

**Programming Guide**  
**Agilent Technologies**  
**83751A/B and 83752A/B Synthesized Sweeper**



Part No. 83750-90005

**Printed in USA**

**July 2001**

Supersedes: March 1997

**Serial Numbers.**

This manual applies directly to instruments with serial prefix 3447A and below.

This manual also applies to firmware revision 2.0 and above. For firmware revisions below 2.0 contact your nearest Agilent Technologies service center for a firmware upgrade.

**Notice.**

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

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

© Copyright Agilent Technologies Inc.1993, 1997, 2001  
All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.  
1400 Fountaingrove Parkway, Santa Rosa, CA 95403-1799, USA

# Certification

Agilent Technologies Inc. certifies that this product met its published specifications at the time of shipment from the factory. Agilent Technologies 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.

## **Regulatory Information.**

The User's Guide contains ISO/IEC regulatory information.

SCPI Conformance Information is found in Chapter 5, "SCPI Conformance Information."

# Warranty

This Agilent Technologies instrument product is warranted against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Agilent Technologies 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 Agilent Technologies. Buyer shall prepay shipping charges Agilent Technologies and Agilent Technologies shall pay shipping charges to return the product to Buyer. However, buyer shall pay all shipping charges, duties, and taxes for products returned to Agilent Technologies from another country. Agilent Technologies warrants that its software and firmware designated by Agilent Technologies for use with an instrument will execute its programming instructions when properly installed on that instrument. Agilent Technologies 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. AGILENT TECHNOLOGIES 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. AGILENT TECHNOLOGIES SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

# Assistance

*Product maintenance agreements and other customer assistance agreements are available for Agilent Technologies products.*

*For any assistance, contact your nearest Agilent Technologies Sales and Service Office.*

# Safety Notes

The following safety notes are used throughout this manual. Familiarize yourself with each of the notes and its meaning before operating this instrument.

---

## CAUTION

---

The *caution* note denotes a hazard. It calls attention to a procedure which, if not correctly performed or adhered to, could result in damage to or destruction of the instrument. Do not proceed beyond a *caution* note until the indicated conditions are fully understood and met.

---

## WARNING

---

The *warning* note denotes a hazard. It calls attention to a procedure which, if not correctly performed or adhered to, could result in injury or loss of life. Do not proceed beyond a *warning* note until the indicated conditions are fully understood and met.

---

**Instruction Manual** The **instruction manual** symbol. The product is marked with this symbol when it is necessary for the user to refer to the instructions in the manual.



# General Safety Considerations

---

**WARNING**

*Before this instrument is switched on*, make sure it has been properly grounded through the protective conductor of the ac power cable to a socket outlet provided with protective earth contact.

Any interruption of the protective (grounding) conductor, inside or outside the instrument, or disconnection of the protective earth terminal can result in personal injury.

---

---

**WARNING**

There are many points in the instrument which can, if contacted, cause personal injury. Be extremely careful.

Any adjustments or service procedures that require operation of the instrument with protective covers removed should be performed only by trained service personnel.

---

---

**CAUTION**

*Before this instrument is switched on*, make sure its primary power circuitry has been adapted to the voltage of the ac power source.

Failure to set the ac power input to the correct voltage could cause damage to the instrument when the ac power cable is plugged in.

---

# How to Use This Guide

## **This guide uses the following conventions.**

- Front-Panel Key** This represents a key physically located on the instrument.
- Softkey** This indicates a “softkey,” a key whose label is determined by the instrument’s firmware.
- Screen Text** This indicates text displayed on the instrument’s screen.



# Contents

## 1. Getting Started Programming

GPIB General Information . . . . .	1-3
Interconnecting Cables . . . . .	1-3
Instrument Addresses . . . . .	1-3
GPIB Instrument Nomenclature . . . . .	1-4
Listener . . . . .	1-4
Talker . . . . .	1-4
Controller . . . . .	1-4
Programming the Sweeper . . . . .	1-4
GPIB Command Statements . . . . .	1-5
Abort . . . . .	1-6
Related statements used by some computers . . . . .	1-6
Remote . . . . .	1-7
Some BASIC examples . . . . .	1-7
Local Lockout . . . . .	1-8
A BASIC example . . . . .	1-8
Local . . . . .	1-8
Some BASIC examples . . . . .	1-8
Clear . . . . .	1-9
Some BASIC examples . . . . .	1-9
Related statements used by some computers . . . . .	1-9
Output . . . . .	1-10
A BASIC example . . . . .	1-11
Related statements used by some computers . . . . .	1-11
Enter . . . . .	1-12
Related statements used by some computers . . . . .	1-13
Getting Started with SCPI . . . . .	1-14
Definitions of Terms . . . . .	1-15
Standard Notation . . . . .	1-16
Command Mnemonics . . . . .	1-16
Angle Brackets . . . . .	1-16
How to Use Examples . . . . .	1-16
Command Examples . . . . .	1-17
Response Examples . . . . .	1-17
Essentials for Beginners . . . . .	1-18
Program and Response Messages . . . . .	1-19

Forgiving Listening and Precise Talking . . . . .	1-19
Types of Commands . . . . .	1-19
Subsystem Command Trees . . . . .	1-21
The Command Tree Structure . . . . .	1-21
Paths Through the Command Tree . . . . .	1-21
Subsystem Command Tables . . . . .	1-24
Reading the Command Table . . . . .	1-25
More About Commands . . . . .	1-26
Query and Event Commands . . . . .	1-26
Implied Commands . . . . .	1-26
Optional Parameters . . . . .	1-26
Program Message Examples . . . . .	1-27
Example 1 . . . . .	1-27
Example 2 . . . . .	1-27
Example 3 . . . . .	1-28
Example 4 . . . . .	1-28
Parameter Types . . . . .	1-29
Numeric Parameters . . . . .	1-29
Extended Numeric Parameters . . . . .	1-30
Discrete Parameters . . . . .	1-31
Boolean Parameters . . . . .	1-31
Reading Instrument Errors . . . . .	1-32
Example Programs . . . . .	1-33
Example Program . . . . .	1-33
Program Comments . . . . .	1-35
Details of Commands and Responses . . . . .	1-36
Program Message Syntax . . . . .	1-37
SCPI Subsystem Command Syntax . . . . .	1-38
Common Command Syntax . . . . .	1-39
Response Message Syntax . . . . .	1-40
SCPI Data Types . . . . .	1-41
Parameter Types . . . . .	1-42
Numeric Parameters . . . . .	1-42
Extended Numeric Parameters . . . . .	1-43
Discrete Parameters . . . . .	1-44
Boolean Parameters . . . . .	1-44
Response Data Types . . . . .	1-45
Real Response Data . . . . .	1-45
Integer Response Data . . . . .	1-45
Discrete Response Data . . . . .	1-46
String Response Data . . . . .	1-46

Programming Typical Measurements . . . . .	1-47
Using the Example Programs . . . . .	1-47
Use of the Command Tables . . . . .	1-48
GPIB Check, Example Program 1 . . . . .	1-51
Program Comments . . . . .	1-51
Local Lockout Demonstration, Example Program 2 . . . . .	1-52
Program Comments . . . . .	1-53
Setting Up A Typical Sweep, Example Program 3 . . . . .	1-54
Program Comments . . . . .	1-55
Queries, Example Program 4 . . . . .	1-56
Program Comments . . . . .	1-56
Saving and Recalling States, Example Program 5 . . . . .	1-58
Program Comments . . . . .	1-59
Looping and Synchronization, Example Program 6 . . . . .	1-60
Program Comments . . . . .	1-61
Using the *WAI Command, Example Program 7 . . . . .	1-62
Program Comments . . . . .	1-63
Using the User Flatness Correction Commands, Example Program 8 . . . . .	1-64
Programming the Status System . . . . .	1-68
General Status Register Model . . . . .	1-69
Condition Register . . . . .	1-69
Transition Filter . . . . .	1-70
Event Register . . . . .	1-70
Enable Register . . . . .	1-70
An Example Sequence . . . . .	1-71
83750 Series Status Register Model . . . . .	1-72
Synthesized Sweeper Status Groups . . . . .	1-72
The Status Byte Group . . . . .	1-72
The Standard Event Status Group . . . . .	1-74
The Standard Operation Status Group . . . . .	1-75
The Questionable Data Status Group . . . . .	1-76
Status Register System Programming Example . . . . .	1-77
Programming the Trigger System . . . . .	1-80
Generalized Trigger Model . . . . .	1-80
Description of Triggering in Sweepers . . . . .	1-82
Advanced Trigger Configurations . . . . .	1-83
Trigger Keyword Definitions . . . . .	1-84
ABORT . . . . .	1-84
IMMediate . . . . .	1-84
SOURce . . . . .	1-84

Related Documents . . . . .	1-85
-----------------------------	------

## 2. Programming Commands

Command Syntax . . . . .	2-3
IEEE 488.2 Common Commands . . . . .	2-4
*CLS (Clear Status Command) . . . . .	2-4
*DMC (Define Macro Command) . . . . .	2-4
*EMC (Enable Macros Command) . . . . .	2-5
Query Syntax . . . . .	2-5
*ESE (Standard Event Status Enable Command) . . . . .	2-5
Query Syntax . . . . .	2-5
*ESR? (Standard Event Status Register Query) . . . . .	2-5
*GMC? (Get Macro Contents Query) . . . . .	2-6
*IDN? (Identification Query) . . . . .	2-6
*LMC? (List Macro Query) . . . . .	2-6
*LRN? (Learn Device Setup Query) . . . . .	2-6
*OPC (Operation Complete Command) . . . . .	2-7
Query Syntax . . . . .	2-7
*OPT? (Option Identification Query) . . . . .	2-7
*PMC (Purge Macros Command) . . . . .	2-8
*PSC (Power-On Status Clear Command) . . . . .	2-8
Example . . . . .	2-8
Query Syntax . . . . .	2-8
*RCL (Recall Command) . . . . .	2-9
*RMC (Remove Macro Command) . . . . .	2-9
*RST (Reset Command) . . . . .	2-9
*SAV (Save Command) . . . . .	2-9
*SRE (Service Request Enable Command) . . . . .	2-10
Query Syntax . . . . .	2-10
*STB? (Read Status Byte Query) . . . . .	2-10
*TRG (Trigger Command) . . . . .	2-10
*TST? (Self-Test Query) . . . . .	2-11
*WAI (Wait-to-Continue Command) . . . . .	2-11
Subsystem Commands . . . . .	2-12
ABORt . . . . .	2-12
AM:STATe . . . . .	2-12
Query Syntax . . . . .	2-12
AM:SOURce . . . . .	2-13
Query Syntax . . . . .	2-13
Calibration Subsystem . . . . .	2-14
CALibration:PEAKing . . . . .	2-14

Query Syntax . . . . .	2-14
CALibration:TRACk . . . . .	2-14
CALibration:PMETer:FLATness:INITiate? . . . . .	2-15
CALibration:PMETer:FLATness:NEXT? . . . . .	2-15
Correction Subsystem . . . . .	2-16
CORRection:FLATness:FREQ . . . . .	2-16
Query Syntax . . . . .	2-16
CORRection:FLATness:AMPL . . . . .	2-17
Query Syntax . . . . .	2-17
CORRection:FLATness:POINts? . . . . .	2-17
CORRection[:STATe] . . . . .	2-18
Query Syntax . . . . .	2-18
CORRection:VOLTs:SCALe . . . . .	2-18
Query Syntax . . . . .	2-18
CORRection:VOLTs:OFFSet . . . . .	2-19
Query Syntax . . . . .	2-19
Diagnostic Subsystem . . . . .	2-20
DIAG:LRNS? . . . . .	2-20
DIAGnostic:TEST:FULLtest? . . . . .	2-20
DIAGnostic:TEST:FULLtest:REPort? . . . . .	2-21
Display Subsystem . . . . .	2-22
DISPlay[:STATe] . . . . .	2-22
Query Syntax . . . . .	2-22
FM Subsystem . . . . .	2-23
FM:COUPLing . . . . .	2-23
Query Syntax . . . . .	2-23
FM:STATe . . . . .	2-23
Query Syntax . . . . .	2-23
FM:SENSitivity . . . . .	2-24
Query Syntax . . . . .	2-24
FM:SOURce . . . . .	2-24
Query Syntax . . . . .	2-24
Frequency Subsystem . . . . .	2-25
FREQUency:CENTer . . . . .	2-25
Query Syntax . . . . .	2-25
Example 1 . . . . .	2-26
Example 2 . . . . .	2-26
Example 3 . . . . .	2-26
FREQUency[:CW]:FIXed] . . . . .	2-27
Query Syntax . . . . .	2-27

FREQUency[:CW]:AUTO and FREQUency[:FIXed]:AUTO . . . . .	2-27
Query Syntax . . . . .	2-27
FREQUency:MANual . . . . .	2-28
Query Syntax . . . . .	2-28
FREQUency:MODE . . . . .	2-29
Query Syntax . . . . .	2-29
FREQUency:MULTIplier . . . . .	2-30
Query Syntax . . . . .	2-30
FREQUency:MULTIplier:STATe . . . . .	2-30
Query Syntax . . . . .	2-30
FREQUency:OFFSet . . . . .	2-31
Query Syntax . . . . .	2-31
FREQUency:OFFSet:STATe . . . . .	2-31
Query Syntax . . . . .	2-31
FREQUency:SPAN . . . . .	2-32
Query Syntax . . . . .	2-32
FREQUency:STARt . . . . .	2-32
Query Syntax . . . . .	2-32
FREQUency:STEP[:INCRement] . . . . .	2-33
Query Syntax . . . . .	2-33
FREQUency:STOP . . . . .	2-33
Query Syntax . . . . .	2-33
Triggering in the Sweeper . . . . .	2-34
INITiate:CONTinuous . . . . .	2-36
Query Syntax . . . . .	2-36
INITiate[:IMMediate] . . . . .	2-36
Marker Subsystem . . . . .	2-37
MARKer[n]:AMPLitude . . . . .	2-37
Query Syntax . . . . .	2-37
MARKer[n]:AOFF . . . . .	2-38
MARKer[n]:FREQUency . . . . .	2-39
Query Syntax . . . . .	2-39
MARKer[n]:MODE . . . . .	2-40
Query Syntax . . . . .	2-40
MARKer[n]:REFerence . . . . .	2-41
Query Syntax . . . . .	2-41
MARKer[n][:STATe] . . . . .	2-42
Query Syntax . . . . .	2-42
Memory Subsystem . . . . .	2-43
MEMory:RAM:INITialize . . . . .	2-43

Output Subsystem . . . . .	2-44
OUTPut:STATe . . . . .	2-44
Query Syntax . . . . .	2-44
OUTPut:IMPedance? . . . . .	2-44
Power Subsystem . . . . .	2-45
POWER:ALC:CFACtor . . . . .	2-45
Query Syntax . . . . .	2-45
POWER:ALC:SOURce . . . . .	2-45
Query Syntax . . . . .	2-45
POWER:ALC[:STATe] . . . . .	2-46
Query Syntax . . . . .	2-46
POWER:ATTenuation . . . . .	2-46
Query Syntax . . . . .	2-46
POWER:ATTenuation:AUTO . . . . .	2-47
Query Syntax . . . . .	2-47
POWER:CENTer . . . . .	2-48
Query Syntax . . . . .	2-48
POWER[:LEVel] . . . . .	2-49
Query Syntax . . . . .	2-49
POWER:MODE FIXed SWEEp . . . . .	2-49
Query Syntax . . . . .	2-49
POWER:OFFSet . . . . .	2-50
Query Syntax . . . . .	2-50
POWER:OFFSet:STATe . . . . .	2-50
Query Syntax . . . . .	2-50
POWER:SLOPe . . . . .	2-51
Query Syntax . . . . .	2-51
POWER:SLOPe:STATe . . . . .	2-51
Query Syntax . . . . .	2-51
POWER:SPAN . . . . .	2-52
Query Syntax . . . . .	2-52
POWER:START . . . . .	2-52
Query Syntax . . . . .	2-52
POWER:STATe . . . . .	2-53
Query Syntax . . . . .	2-53
POWER:STEP[:INCRement] . . . . .	2-53
Query Syntax . . . . .	2-53
POWER:STOP . . . . .	2-54
Query Syntax . . . . .	2-54
Pulse Subsystem . . . . .	2-55
PULSe:PERiod . . . . .	2-55

Query Syntax . . . . .	2-55
PULSe:FREQuency . . . . .	2-56
Query Syntax . . . . .	2-56
PULSe:WIDTh . . . . .	2-56
Query Syntax . . . . .	2-56
PULM:SOURce . . . . .	2-57
Query Syntax . . . . .	2-57
PULM:STATe . . . . .	2-57
Query Syntax . . . . .	2-57
ROSCillator:SOURce . . . . .	2-58
Query Syntax . . . . .	2-58
ROSCillator:SOURce:AUTO . . . . .	2-58
Query Syntax . . . . .	2-58
Status Subsystem . . . . .	2-59
STATus:OPERation:CONDition? . . . . .	2-59
STATus:OPERation:ENABle . . . . .	2-59
Query Syntax . . . . .	2-59
STATus:OPERation[:EVENT]? . . . . .	2-59
STATus:OPERation:NTRansition . . . . .	2-60
Query Syntax . . . . .	2-60
STATus:OPERation:PTRansition . . . . .	2-60
Query Syntax . . . . .	2-60
STATUS:PRESet . . . . .	2-61
STATus:QUEStionable:CONDition? . . . . .	2-61
STATus:QUEStionable:ENABle . . . . .	2-61
Query Syntax . . . . .	2-61
STATus:QUEStionable[:EVENT]? . . . . .	2-62
STATus:QUEStionable:NTRansition . . . . .	2-62
Query Syntax . . . . .	2-62
STATus:QUEStionable:PTRansition . . . . .	2-63
Query Syntax . . . . .	2-63
Sweep Subsystem . . . . .	2-64
SWEep:CONTRol:TYPE . . . . .	2-64
Query Syntax . . . . .	2-64
SWEep:DWELl . . . . .	2-65
Query Syntax . . . . .	2-65
SWEep:DWELl:AUTO . . . . .	2-65
Query Syntax . . . . .	2-65
SWEep:POINts . . . . .	2-66
Query Syntax . . . . .	2-66
SWEep:POWer:STEP . . . . .	2-67



Query Syntax . . . . .	2-67
SWEep[:FREQUency]:STEP . . . . .	2-68
Query Syntax . . . . .	2-68
SWEep:TIME . . . . .	2-69
Query Syntax . . . . .	2-69
SWEep:TIME:AUTO . . . . .	2-70
Query Syntax . . . . .	2-70
SWEep:TIME:LLIMit . . . . .	2-70
Query Syntax . . . . .	2-70
SWEep:GENeration . . . . .	2-72
Query Syntax . . . . .	2-72
SWEep:MODE . . . . .	2-72
Query Syntax . . . . .	2-72
SWEep:MANual[:RELative] . . . . .	2-73
Query Syntax . . . . .	2-73
SWEep:MANual:POINt . . . . .	2-73
Query Syntax . . . . .	2-73
SWEep:MARKer:STATe . . . . .	2-74
Query Syntax . . . . .	2-74
SWEep:MARKer:XFER . . . . .	2-74
SWEep[:POINts]:TRIGger:SOURce . . . . .	2-75
Query Syntax . . . . .	2-75
SWEep:POINts:TRIGger: . . . . .	2-75
System Subsystem . . . . .	2-76
SYSTem:ALTerNate . . . . .	2-76
Query Syntax . . . . .	2-76
SYSTem:ALTerNate:STATe . . . . .	2-76
Query Syntax . . . . .	2-76
SYSTem:COMMunicate:GPIB:ADDRes . . . . .	2-77
SYSTem:COMMunicate:PMETer:ADDRes . . . . .	2-77
Query Syntax . . . . .	2-77
SYSTem:COMMunicate:PMETer:TYPE . . . . .	2-78
Query Syntax . . . . .	2-78
SYSTem:ERRor? . . . . .	2-78
SYSTem:KEY[:CODE] . . . . .	2-79
Query Syntax . . . . .	2-79
SYSTem:KEY:DISable . . . . .	2-81
Query Syntax . . . . .	2-81
SYSTem:KEY:ENABle . . . . .	2-81
SYSTem:LANGUage . . . . .	2-82
Query Syntax . . . . .	2-82

SYSTem:PRESet[:EXECute]	2-82
SYSTem:PRESet:SAVE	2-82
SYSTem:PRESet:TYPE	2-83
Query Syntax	2-83
SYSTem:SECurity:CLEAr	2-83
SYSTem:SECurity:COUNT	2-84
SYSTem:SECurity:KLOCK	2-84
SYSTem:SECurity:ZERO	2-84
SYSTem:VERSion?	2-85
Trigger Subsystem	2-86
TRIGger[:IMMediate]	2-86
TRIGger:SOURce	2-87
Query Syntax	2-87
TSWeep	2-88

### 3. 8350B Compatibility Guide

Introduction	3-1
Data	3-1
Input Syntax	3-1
Function Codes (Prefix Active)	3-2
Numeric Value (Numeric Format)	3-2
Numeric Terminators	3-3
Valid Characters	3-3
Instrument Preset	3-4
Output Data	3-4
Learn String	3-5
Mode String	3-6
Interrogate Function	3-13
Active Function	3-13
Status	3-14
Trigger	3-14
Input Programming Codes	3-15
Clear	3-22
Remote/Local Changes	3-22
Service Request	3-23
Status Byte	3-24
Status Bit	3-24
Pass Control	3-24
Abort	3-24
Interface Function Codes	3-25
83750 Series Status Byte Descriptions	3-26

<b>4. Error Messages</b>	
:ERRor? SYSTem:ERRor . . . . .	4-3
The Error/Event Queue . . . . .	4-4
Error numbers . . . . .	4-5
No Error . . . . .	4-5
SCPI Error Messages . . . . .	4-6
Error Message Description . . . . .	4-6
Example Error . . . . .	4-6
Command Error . . . . .	4-8
Execution Error . . . . .	4-14
Device-Specific Error . . . . .	4-20
Query Error . . . . .	4-22
Instrument Specific Error Messages . . . . .	4-24
Block Transfer Errors . . . . .	4-24
Bus Control Errors . . . . .	4-25
Parsing and Compatibility Errors . . . . .	4-26
Diagnostics and Self-Test Errors . . . . .	4-29
Internal Hardware Errors . . . . .	4-33
Hardware Configuration Errors . . . . .	4-33
Calibration Routine Errors . . . . .	4-34
Loops Unlocked Errors . . . . .	4-36
Miscellaneous Hardware Dependent Errors . . . . .	4-37
<b>5. SCPI Conformance Information</b>	
SCPI Conformance . . . . .	5-3

**Index**

# Figures

1-1. SCPI Command Types . . . . .	1-20
1-2. A Simplified Command Tree . . . . .	1-21
1-3. Proper Use of the Colon and Semicolon . . . . .	1-23
1-4. Simplified SWEEP Command Tree . . . . .	1-24
1-5. Voltage Controlled Oscillator Test . . . . .	1-33
1-6. Simplified Program Message Syntax . . . . .	1-37
1-7. SCPI Simplified Subsystem Command Syntax . . . . .	1-38
1-8. Simplified Common Command Syntax . . . . .	1-39
1-9. Simplified Response Message Syntax . . . . .	1-40
1-10. Generalized Status Register Model . . . . .	1-69
1-11. Typical Status Register Bit Changes . . . . .	1-71
1-12. Status Registers . . . . .	1-78
1-13. The TRIG Trigger Configuration . . . . .	1-81
1-14. Simplified Trigger Model . . . . .	1-82
2-1. Instrument Trigger Model . . . . .	2-34

---

# Tables

1-1. Command Table . . . . .	1-25
1-2. SCPI Data Types . . . . .	1-41
1-3. Sample Sweeper Commands . . . . .	1-49
2-1. Interactions between Dwell, Sweep Time, and Points. . . . .	2-64
2-2. 83750 SCPI Sweep Mode Programming Table . . . . .	2-71
2-3. Sweeper Key Codes . . . . .	2-80
5-1. SCPI Conformance . . . . .	5-4

---

Getting Started  
Programming

# Getting Started Programming

GPIB, the Generic Interface Bus, is the instrument-to-instrument communication system between the sweeper and up to 14 other instruments. Any instrument having GPIB capability can be interfaced to the sweeper, including non-Agilent instruments that have “GPIB,” “IEEE-488,” “ANSI MC1.1,” or “IEC-625” capability (these are common generic terms for GPIB; all are electrically equivalent although IEC-625 uses a unique connector). This portion of the manual specifically describes interfacing the sweeper to a computer.

The first part of this section provides general GPIB information. Later, the Standard Commands for Programmable Instruments language (SCPI) is introduced, and example programs are given.

---

## Interconnecting Cables

The Installation Guide shows the sweeper rear-panel GPIB connector and suitable cables, and describes the procedures and limitations for interconnecting instruments. Cable length restrictions, also described in the Installation Guide, must be observed to prevent transmission line propagation delays that might disrupt GPIB timing cycles.

---

## Instrument Addresses

Each instrument in an GPIB network must have a unique address, an integer ranging in value from 0 to 30. The default address for the sweeper is 19, but this can be changed using the **(SHIFT)** **(LOCAL)** keys or rear panel switch.

---

## GPIB Instrument Nomenclature

An GPIB instrument is categorized as a “listener,” “talker,” or “controller,” depending on its current function in the network.

listener

A listener is a device capable of receiving data or commands from other instruments. Any number of instruments in the GPIB network can be listeners simultaneously.

talker

A talker is a device capable of transmitting data or commands to other instruments. To avoid confusion, an GPIB system allows only one device at a time to be an active talker.

controller

A controller is an instrument, typically a computer, capable of managing the various GPIB activities. Only one device at a time can be an active controller.

---

## Programming the Sweeper

The sweeper can be controlled entirely by a computer (although the line POWER switch must be operated manually). Several functions are possible only by computer (remote) control. Computer programming procedures for the sweeper involve selecting an GPIB command statement, then adding the specific sweeper (SCPI, Analyzer) programming codes to that statement to achieve the desired operating conditions. The programming codes can be categorized into two groups: Those that mimic front panel keystrokes; and those that are unique, and have no front panel equivalent.

In the programming explanations that follow, specific examples are included that are written in a generic dialect of the BASIC language. BASIC was selected because the majority of GPIB computers have BASIC language capability. However, other languages can also be used.



---

## GPIB Command Statements

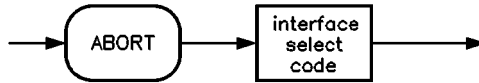
Command statements form the nucleus of GPIB programming; they are understood by all instruments in the network and, when combined with the programming language codes, they provide all management and data communication instructions for the system.

An explanation of the **eight fundamental command statements** follows. However, some computers use a slightly different terminology, or support an extended or enhanced version of these commands. Consider the following explanations as a starting point, but for detailed information consult the BASIC language reference manual, the I/O programming guide, and the GPIB manual for the particular computer used.

Syntax drawings accompany each statement: All items enclosed by a circle or oval are computer specific terms that must be entered exactly as described; items enclosed in a rectangular box are names of parameters used in the statement; and the arrows indicate a path that generates a valid combination of statement elements.

## Abort

Abort abruptly terminates all listener/talker activity on the interface bus, and prepares all instruments to receive a new command from the controller. Typically, this is an initialization command used to place the bus in a known starting condition. The syntax is:



where the interface select code is the computer's GPIB I/O port, which is typically port 7. Some BASIC examples:

```
10 ABORT 7  
100 IF V>20 THEN ABORT 7
```

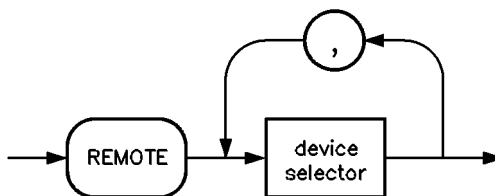
Related statements used  
by some computers

- ABORTIO (used by HP-80 series computers)
- HALT
- RESET

---

## Remote

Remote causes an instrument to change from local control to remote control. In remote control, the front panel keys are disabled (except for the **LOCAL** key and the POWER switch), and the REMOTE annunciator is lighted. The syntax is:



where the device selector is the address of the instrument appended to the GPIB port number. Typically, the GPIB port number is 7, and the default address for the sweeper is 19, so the device selector is 719.

Some BASIC examples

10 **REMOTE 7**

which prepares all GPIB instruments for remote operation (although nothing appears to happen to the instruments until they are addressed to talk), or

10 **REMOTE 719**

which affects the GPIB instrument located at address 19, or

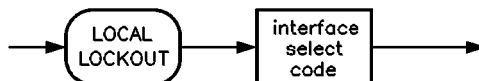
10 **REMOTE 719, 721, 726, 715**

which effects four instruments that have addresses 19, 21, 26, and 15.

---

## Local Lockout

Local Lockout can be used in conjunction with REMOTE to disable the front panel **LOCAL** key. With the **LOCAL** key disabled, only the controller (or a hard reset by the POWER switch) can restore local control. The syntax is:



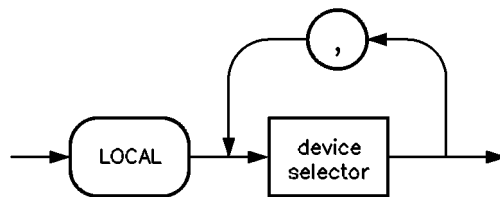
A BASIC example

```
10  REMOTE 719
20  LOCAL LOCKOUT 7
```

---

## Local

Local is the complement to REMOTE, causing an instrument to return to local control with a fully enabled front panel. The syntax is:



Some BASIC examples

```
10  LOCAL 7
    which effects all instruments in the network, or
10  LOCAL 719
    for an addressed instrument (address 19).
```

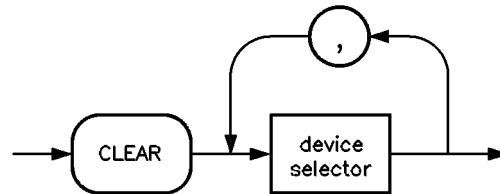
---

## Clear

Clear causes all GPIB instruments, or addressed instruments, to assume a “cleared” condition, with the definition of “cleared” being unique for each device. For the sweeper:

1. All pending output-parameter operations are halted.
2. The parser (the software that interprets the programming codes) is reset, and now expects to receive the first character of a programming code.

The syntax is:



Some BASIC examples

```
10 CLEAR 7
```

to clear all GPIB instruments, or

```
10 CLEAR 719
```

to clear an addressed instrument.

Related statements used  
by some computers

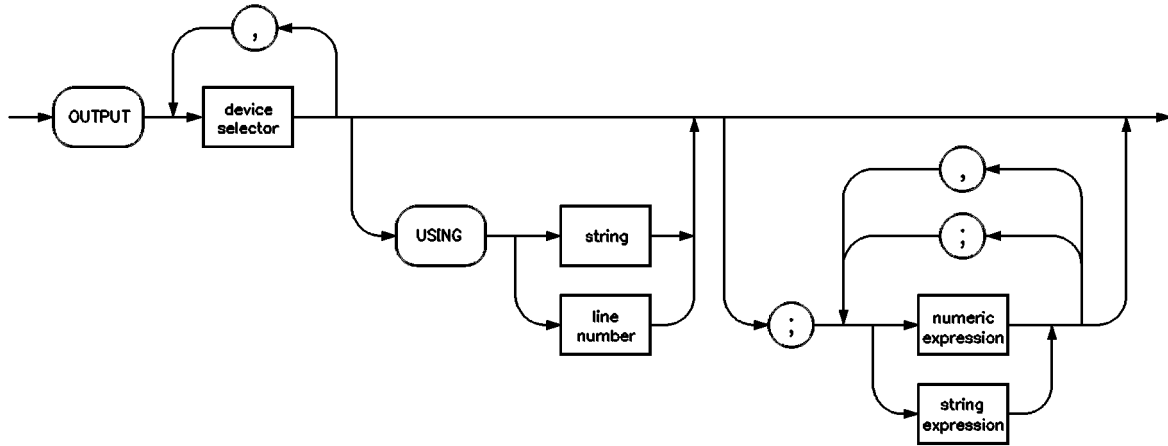
- RESET
- CONTROL
- SEND

The preceding statements are primarily management commands that do not incorporate programming codes. The following two statements do incorporate programming codes, and are used for data communication.

---

## Output

Output is used to send function commands and data commands from the controller to the addressed instrument. The syntax is:



where USING is a secondary command that formats the output in a particular way, such as a binary or ASCII representation of numbers. The USING command is followed by “image items” that precisely define the format of the output; these image items can be a string of code characters, or a reference to a statement line in the computer program. Image items are explained in the programming codes where they are needed. Notice that this syntax is virtually identical to the syntax for the ENTER statement that follows.

A BASIC example            100 OUTPUT 719; "programming codes"

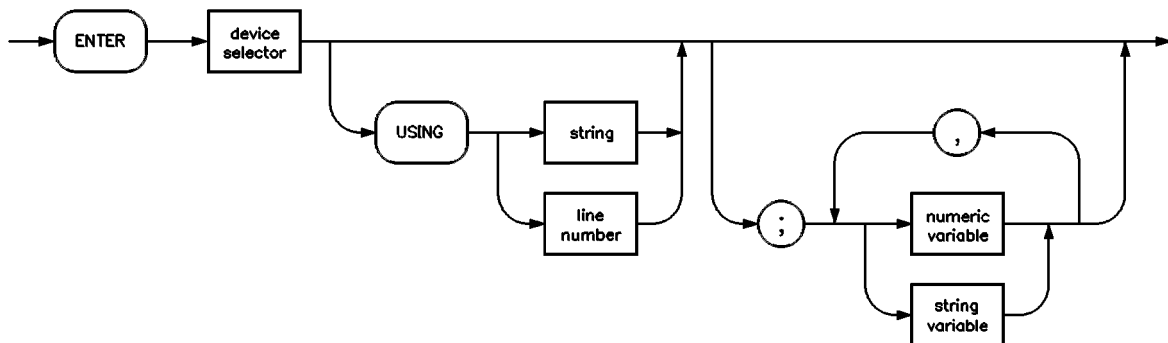
The many programming codes for the sweeper are listed in the "SCPI Command Summary" in chapter 2.

Related statements used  
by some computers

- CONTROL
- CONVERT
- IMAGE
- IOBUFFER
- TRANSFER

## Enter

Enter is the complement of OUTPUT, and is used to transfer data from the addressed instrument to the controller. The syntax is:



ENTER is always used in conjunction with OUTPUT, such as:

```
100 OUTPUT 719; " ... programming codes ... "
110 ENTER 719; " ... response data ... "
```

ENTER statements are commonly formatted, which requires the secondary command USING and the appropriate image items. The most-used image items involve end-of-line (end or identify) suppression, binary inputs, and literal inputs.

Example

```
100 ENTER 719 USING "#, B"; A, B, C
```

suppresses the EOI sequence (#), and indicates that variables A, B, and C are to be filled with binary (B) data. As another example,

```
100 ENTER 719 USING "#, 123A"; A$
```

suppresses EOI, and indicates that string variable A\$ is to be filled with 123 bytes of literal data (123A).



**NOTE**

Be careful when using byte-counting image specifiers. If the requested number of bytes does not match the actual number available, data might be lost, or the program might enter an endless wait state.

The suppression of the EOI sequence is frequently necessary to prevent a premature termination of the data input. When not specified, the typical EOI termination occurs when an ASCII LF (line feed) is received. However, the LF bit pattern could coincidentally occur randomly in a long string of binary data, where it might cause a false termination. Also, the bit patterns for the ASCII CR (carriage return), comma, or semicolon might cause a false termination. Suppression of the EOI causes the computer to accept all bit patterns as data, not commands, and relies on the GPIB EOI (end or identify) line for correct end-of-data termination.

Related statements used  
by some computers

- CONVERT
- IMAGE
- IOBUFFER
- ON TIMEOUT
- SET TIMEOUT
- TRANSFER

This completes the GPIB Command Statements subsection. The following material explains the SCPI programming codes, and shows how they are used with the OUTPUT and ENTER GPIB command statements.

# Getting Started with SCPI

This section of chapter 1 describes the use of the Standard Commands for Programmable Instruments language (SCPI). This section explains how to use SCPI commands in general. The instrument command summary in Chapter 5 lists the specific commands available in the instrument. This section presents only the basics of SCPI. If you want to explore the topic in greater depth, see the paragraph titled, “Related Documents.”

# Definitions of Terms

You need a general understanding of the terms listed below before you continue.

<b>controller</b>	A controller is any computer used to communicate with a SCPI instrument. A controller can be a personal computer, a minicomputer, or a plug-in card in a card cage. Some intelligent instruments can also function as controllers.
<b>instrument</b>	An instrument is any device that implements SCPI. Most instruments are electronic measurement or stimulus devices, but this is not a requirement. Similarly, most instruments use an GPIB interface for communication. The same concepts apply regardless of the instrument function or the type of interface used.
<b>program message</b>	A program message is a combination of one or more properly formatted SCPI commands. Program messages always go from a controller to an instrument. Program messages tell the instrument how to make measurements and output signals.
<b>response message</b>	A response message is a collection of data in specific SCPI formats. Response messages always go from an instrument to a controller or listening instrument. Response messages tell the controller about the internal state of the instrument and about measured values.
<b>command</b>	A command is an instruction in SCPI. You combine commands to form messages that control instruments. In general, a command consists of mnemonics (keywords), parameters, and punctuation.
<b>query</b>	A query is a special type of command. Queries instruct the instrument to make response data available to the controller. Query mnemonics always end with a question mark.

---

## Standard Notation

This section uses several forms of notation that have specific meaning.

Command Mnemonics

Many commands have both a long and a short form, and you must use either one or the other (SCPI does not accept a combination of the two). Consider the **FREQuency** command, for example. The short form is **FREQ** and the long form is **FREQUENCY** (this notation style is a shorthand to document both the long and short form of commands). SCPI is not case sensitive, so **fREquEnCy** is just as valid as **FREQUENCY**, but **FREQ** and **FREQUENCY** are the only valid forms of the **FREQuency** command.

Angle Brackets

Angle brackets indicate that the word or words enclosed represent something other than themselves. For example, **<new line>** represents the ASCII character with the decimal value 10. Similarly, **<^END>** means that EOI is asserted on the GPIB interface. Words in angle brackets have much more rigidly defined meaning than words used in ordinary text. For example, this section uses the word “message” to talk about messages generally. But the bracketed words **<program message>** indicate a precisely defined element of SCPI. If you need them, you can find the exact definitions of words such as **<program message>** in a syntax diagram.

---

## How to Use Examples

It is important to understand that programming with SCPI actually requires knowledge of two languages. You must know the programming language of your controller (BASIC, C, Pascal) as well as the language of your instrument (SCPI). The semantic requirements of your controller’s language determine how the SCPI commands and responses are handled in your application.

## Command Examples

Command examples look like this:

```
:FREQUENCY: CW?
```

This example tells you to put the string `:FREQUENCY: CW?` in the output statement appropriate to your application programming language. If you encounter problems, study the details of how the output statement handles message terminators such as `<new line>`. If you are using simple `OUTPUT` statements in HTBasic, this is taken care of for you. In HTBasic, you type:

```
OUTPUT Source; ":FREQUENCY: CW?"
```

Command examples do not show message terminators because they are used at the end of every program message. “Details of Commands and Responses,” discusses message terminators in more detail.

## Response Examples

Response examples look like this:

```
1.23
```

These are the characters you would read from an instrument after sending a query command. To actually pull them from the instrument into the controller, use the input statement appropriate to your application programming language. If you have problems, study the details of how the input statement operates. In particular, investigate how the input statement handles punctuation characters such as comma and semicolon, and how it handles `<new line>` and EOI. To enter the previous response in HTBasic, you type:

```
ENTER Source; CW_frequency
```

Response examples do not show response message terminators because they are always `<new line> <^END>`. These terminators are typically automatically handled by the input statement. The paragraph titled “Details of Commands and Responses” discusses message terminators in more detail.

# Essentials for Beginners

This subsection discusses elementary concepts critical to first-time users of SCPI. Read and understand this subsection before going on to another. This subsection includes the following topics:

<b>Program and Response Messages</b>	These paragraphs introduce the basic types of messages sent between instruments and controllers.
<b>Subsystem Command Trees</b>	These paragraphs describe the tree structure used in subsystem commands.
<b>Subsystem Command Tables</b>	These paragraphs present the condensed tabular format used for documenting subsystem commands.
<b>Reading Instrument Errors</b>	These paragraphs explain how to read and print an instrument's internal error messages.
<b>Example Programs</b>	These paragraphs contain two simple measurement programs that illustrate basic SCPI programming principles.

---

## Program and Response Messages

To understand how your instrument and controller communicate using SCPI, you must understand the concepts of program and response messages. *Program messages* are the formatted data sent from the controller to the instrument. Conversely, *response messages* are the formatted data sent from the instrument to the controller. Program messages contain one or more commands, and response messages contain one or more responses.

The controller may send commands at any time, but the instrument sends responses only when specifically instructed to do so. The special type of command used to instruct the instrument to send a response message is the *query*. All query mnemonics end with a question mark. Queries return either measured values or internal instrument settings. Any internal setting that can be programmed with SCPI can also be queried.

### Forgiving Listening and Precise Talking

SCPI uses the concept of forgiving listening and precise talking outlined in IEEE 488.2.

*Forgiving listening* means that instruments are very flexible in accepting various command and parameter formats. For example, the sweeper accepts either `:POWer:STATe ON` or `:POWer:STATe 1` to turn RF output on.

*Precise talking* means that the response format for a particular query is always the same. For example, if you query the power state when it is on (using `:POWer:STATe?`), the response is always `1`, regardless of whether you previously sent `:POWer:STATe 1` or `:POWer:STATe ON`.

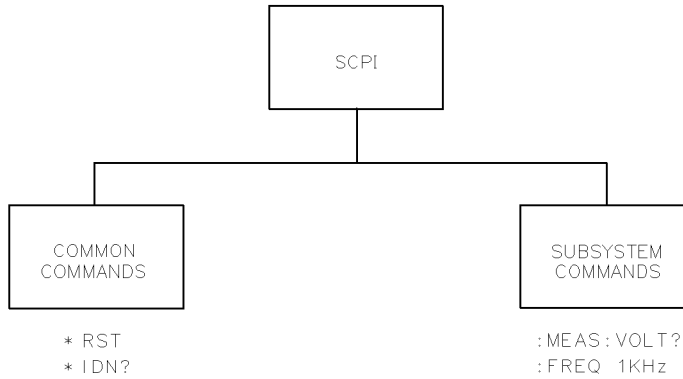
### Types of Commands

Commands can be separated into two groups, common commands and subsystem commands.

*Common commands* are generally not measurement related. They are used to manage macros, status registers, synchronization, and data storage. Common commands are easy to recognize because they all begin with an asterisk, such as `*IDN?`, `*OPC`, and `*RST`. Common commands are defined by IEEE 488.2.

*Subsystem commands* include all measurement functions and some general purpose functions. Subsystem commands are distinguished by the colon used between keywords, as in `:FREQuency:CW?`. Each command subsystem is a

set of commands that roughly corresponds to a functional block inside the instrument. For example, the **POWer** subsystem contains commands for power generation, while the **STATus** subsystem contains commands for accessing status registers.



cg41a

**Figure 1-1. SCPI Command Types**

The remaining paragraphs in this subsection discuss subsystem commands in more detail. Remember, some commands are implemented in one instrument and not in another, depending on its measurement function.

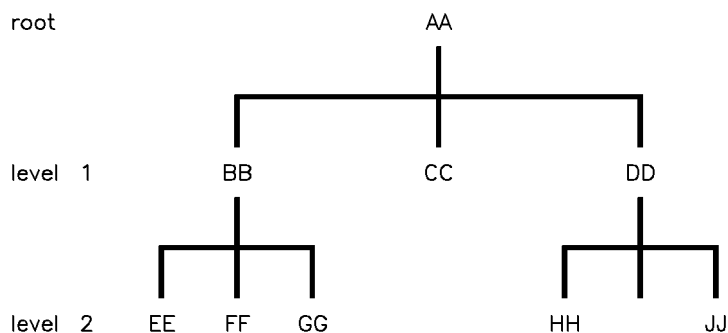


---

## Subsystem Command Trees

### The Command Tree Structure

Most programming tasks involve subsystem commands. SCPI uses a hierarchical structure for subsystem commands similar to the file systems on most computers. In SCPI, this command structure is called a *command tree*.



**Figure 1-2. A Simplified Command Tree**

In the command tree shown in Figure 1-2, the command closest to the top is the *root command*, or simply the *root*. Notice that you must follow a particular *path* to reach lower level subcommands. For example, if you wish to access the **GG** command, you must follow the path **AA** to **BB** to **GG**.

### Paths Through the Command Tree

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

- *Power On and Reset*

After power is cycled or after \*RST, the current path is set to the root.

- *Message Terminators*

A message terminator, such as a <new line> character, sets the current path to the root. Many programming languages have output statements that send message terminators automatically. The paragraph titled, “Details of Commands and Responses,” discusses message terminators in more detail.

- *Colon*

When it is between two command mnemonics, a colon moves the current path down one level in the command tree. For example, the colon in **MEAS:VOLT** specifies that **VOLT** is one level below **MEAS**. When the colon is the first character of a command, it specifies that the next command mnemonic is a root level command. For example, the colon in **:INIT** specifies that **INIT** is a root level command.

- *Semicolon*

A semicolon separates two commands in the same message without changing the current path.

- *Whitespace*

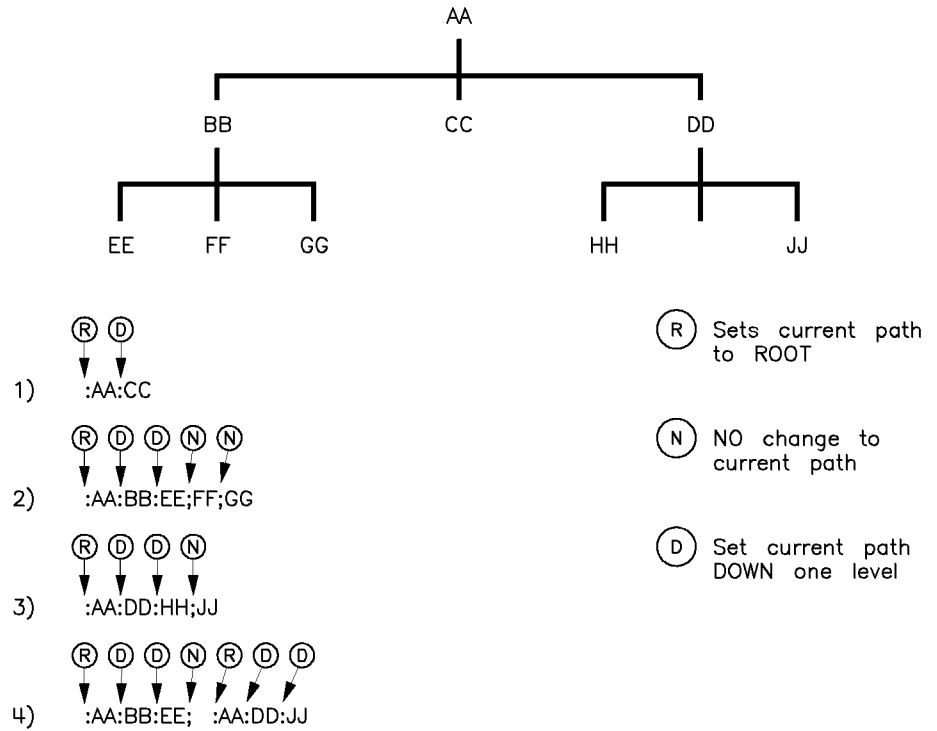
Whitespace characters, such as <tab> and <space>, are generally ignored. There are two important exceptions. Whitespace inside a keyword, such as **:FREQUENCY**, is not allowed. You must use white space to separate parameters from commands. For example, the <space> between **LEVEL** and **6.2** in the command **:POWER:LEVEL 6.2** is mandatory. Whitespace does not affect the current path.

- *Commas*

If a command requires more than one parameter, you must separate adjacent parameters using a comma. Commas do not affect the current path.

- *Common Commands*

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



**Figure 1-3. Proper Use of the Colon and Semicolon**

Figure 1-3 shows examples of how to use the colon and semicolon to navigate efficiently through the command tree. Notice how proper use of the semicolon can save typing.

Sending this message:

`:AA:BB:EE; FF; GG`

Is the same as sending these three messages:

`:AA:BB:EE`

`:AA:BB:FF`

`:AA:BB:GG`

## Subsystem Command Tables

These paragraphs introduce a more complete, compact way of documenting subsystems using a tabular format. The command table contains more information than just the command hierarchy shown in a graphical tree. In particular, these tables list command parameters for each command and response data formats for queries. To begin this exploration of command tables, consider a simplified **SWEep** subsystem for the sweeper in both the graphical and tabular formats.

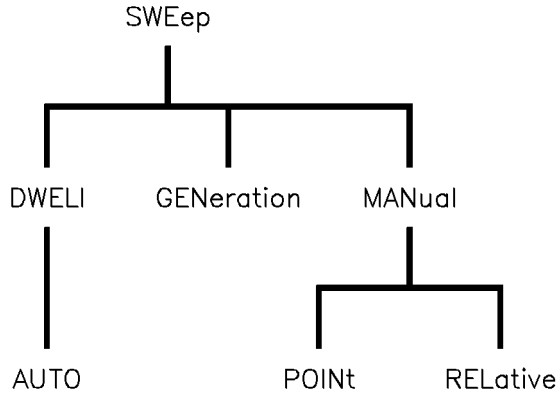


Figure 1-4. Simplified SWEep Command Tree

Table 1-1. Command Table

Command	Parameters	Parameter Type
:SWEep :DWELl :AUTO :GENeration :MANual :POINt [:RELative]	state	Boolean ONCE

## Reading the Command Table

Note the three columns in the command table labeled *Command*, *Parameters*, and *Parameter Type*. Commands closest to the root level are at the top of the table. Commands in square brackets are implied commands, which are discussed in later paragraphs. If a command requires one or more parameters in addition to the keyword, the parameter names are listed adjacent to the command. Parameters in square brackets are optional parameters, which are discussed in later paragraphs. If the parameter is not in square brackets, it is required and you must send a valid setting for it with the matching command. The parameter type is listed adjacent to each named parameter.

## More About Commands

Query and Event  
Commands

Because you can query any value that you can set, the query form of each command is not shown explicitly in the command tables. For example, the presence of the sweeper `:SWEep:DWELL` command implies that a `:SWEep:DWELL?` also exists. If you see a table containing a command ending with a question mark, it is a *query* only command. Some commands are *events*, and cannot be queried. An event has no corresponding setting if it causes something to happen inside the instrument at a particular instant. For example, `:INITiate:IMMEDIATE` causes a certain trigger sequence to initiate. Because it is an event, there is no query form of `:INITiate:IMMEDIATE`.

Implied Commands

*Implied commands* appear in square brackets in the command table. If you send a subcommand immediately preceding an implied command, but do not send the implied command, the instrument assumes you intend to use the implied command, and behaves just as if you had sent it. Note that this means the instrument expects you to include any parameters required by the implied command. The following example illustrates equivalent ways to program the sweeper using explicit and implied commands.

Example sweeper commands with and without an implied commands:

```
:SWEep:MANual:RELative 6  using explicit commands  
:SWEep:MANual 6          using implied commands
```

Optional Parameters

Optional parameter names are enclosed in square brackets in the command table. If you do not send a value for an optional parameter, the instrument chooses a default value. The instrument's command dictionary documents the values used for optional parameters.

---

## Program Message Examples

The following parts of the sweeper SCPI command set will be used to demonstrate how to create complete SCPI program messages:

```
:FREQuency
  [:CW]
    :MULTiplier
      :STATE
:POWER
  [:LEVEL]
```

Example 1            "FREQuency: CW 5 GHZ; MULTiplier 2"

The command is correct and will not cause errors. It is equivalent to sending:

```
"FREQuency: CW 5 GHZ; :FREQuency: MULTiplier 2".
```

Example 2            "FREQuency 5 GHZ; MULTiplier 2"

This command results in a command error. The command makes use of the default [:CW] node. When using a default node, there is no change to the current path position. Since there is no command "MULT" at the root, an error results. A correct way to send this is:

```
"FREQ 5 GHZ; FREQ: MULT 2"
```

or as in example 1.

Example 3

```
"FREQuency:MULTIplier 2; MULTIplier:STATE ON; FREQuency:CW 5  
GHZ"
```

This command results in a command error. The `FREQ:CW` portion of the command is missing a leading colon. The path level is dropped at each colon until it is in the `FREQ:MULT` subsystem. So when the `FREQ:CW` command is sent, it causes confusion because no such node occurs in the `FREQ:MULT` subsystem. By adding a leading colon, the current path is reset to the root. The corrected command is:

```
"FREQuency:MULTIplier 2; MULTIplier:STATE ON; :FREQuency:CW  
5 GHZ".
```

Example 4

```
"FREQ 5 GHZ; POWER 4 DBM"
```

Notice that in this example the keyword short form is used. The command is correct. It utilizes the default nodes of `[:CW]` and `[:LEVEL]`. Since default nodes do not affect the current path, it is not necessary to use a leading colon before `POWER`.



---

## Parameter Types

As you saw in the example command table for **SWEEP**, there are several types of parameters. The parameter type indicates what kind of values are valid instrument settings. The most commonly used parameter types are numeric, extended numeric, discrete, and Boolean. These common types are discussed briefly in the following paragraphs. The paragraph titled “Details of Commands and Responses” explains all parameter types in greater depth.

### Numeric Parameters

Numeric parameters are used in both subsystem commands and common commands. Numeric parameters accept all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation. If an instrument accepts only specific numeric values, such as integers, it automatically rounds numeric parameters to fit its needs.

Examples of numeric parameters:

100	<i>no decimal point required</i>
100.	<i>fractional digits optional</i>
-1.23	<i>leading signs allowed</i>
4.56e<space>3	<i>space allowed after e in exponents</i>
-7.89E-01	<i>use either E or e in exponentials</i>
+256	<i>leading + allowed</i>
.5	<i>digits left of decimal point optional</i>

Examples of numeric parameters in commands:

```
100 OUTPUT @Source;":FREQuency:STARt 1.0E+09"  
110 OUTPUT @Source;":POWer LEVel -5"
```

Most measurement related subsystems use *extended numeric* parameters to specify physical quantities. Extended numeric parameters accept all numeric parameter values and other special values as well. All extended numeric parameters accept **MAXimum** and **MINimum** as values. Other special values, such as **UP** and **DOWN** may be available as documented in the instrument's command summary. Some instruments also let you to send engineering units as suffixes to extended numeric parameters. The SCPI Command Summary lists the suffixes available, if any. Note that extended numeric parameters are not used for common commands or **STATus** subsystem commands.

**Examples of extended numeric parameters:**

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

**Examples of extended numeric parameters in commands:**

```
100 OUTPUT @Source;":FREQUENCY:STOP MAX"
```

## Discrete Parameters

Use discrete parameters to program settings that have a finite number of values. Discrete parameters use mnemonics to represent each valid setting. They have a long and a short form, like command mnemonics. You can use mixed upper and lower case letters for discrete parameters.

Examples of discrete parameters:

```
INTernal  level internally
DIODe    level using an external diode
PMETer   level using an external power meter
MMHead   Level using a mm-wave source module
```

Examples of discrete parameters in commands:

```
100 OUTPUT @Source;":POWER:ALC:SOURce INT"
110 OUTPUT @Source;":POWER:ALC:SOURce mmh"
```

Although discrete parameters values look like command keywords, do not confuse the two. In particular, be sure to use colons and spaces properly. Use a colon to separate command mnemonics from each other. Use a space to separate parameters from command mnemonics.

## Boolean Parameters

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

Examples of Boolean parameters:

```
ON       Boolean TRUE, upper/lower case allowed
OFF      Boolean FALSE, upper/lower case allowed
1        Boolean TRUE
0        Boolean FALSE
```

Examples of Boolean parameters in commands:

```
100 OUTPUT @Source;":FM:STATe On"
110 OUTPUT @Source;":AM:STATe 1"
```

---

## Reading Instrument Errors

When debugging a program, you may want to know if an instrument error has occurred. Some instruments can display error messages on their front panels. If your instrument cannot do this, you can put the following code segment in your program to read and display error messages.

```
10 !
20 ! The rest of your
30 ! variable declarations
40 !
50 DIM Err_msg$(75)
60 INTEGER Err_num
70 !
80 ! Part of your program
90 ! that generates errors
100 !
110 !
200 REPEAT
210   OUTPUT @Box;":SYST:ERR?"
220   ! Query instrument error
230   ENTER @Box;Err_num,Err_msg$
240   ! Read error #, message
250   PRINT Err_num,Err_msg$
260   ! Print error message
270 UNTIL Err_num = 0
280 ! Repeat until no errors
290 !
300 ! The rest of your program
310 !
```

---

## Example Programs

The following is an example program using SCPI compatible instruments. The example is written in HTBasic.

This example is a stimulus and response application. It uses a source and counter to test a voltage controlled oscillator.

This example demonstrates how several SCPI instruments work together to perform a stimulus/response measurement. This program measures the linearity of a voltage controlled oscillator (VCO). A VCO is a device that outputs a frequency proportional to an input signal level. Figure 1-5 shows how the hardware is configured.

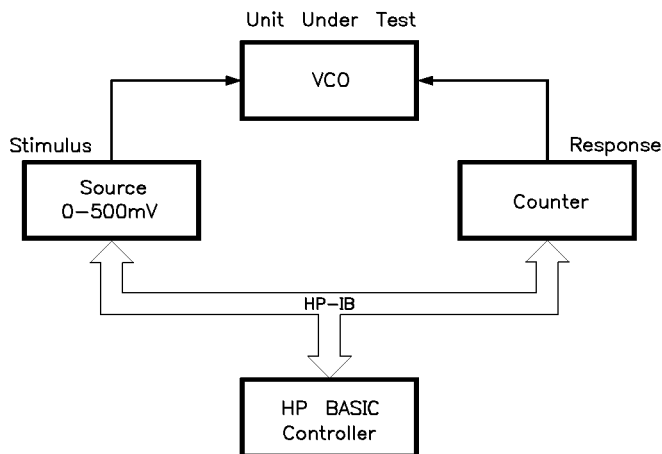


Figure 1-5. Voltage Controlled Oscillator Test

Example Program

```
20 !
30 INTEGER First,Last,Testpoint,Dummy
40 DIM Id$[70]
50 ASSIGN @Stimulus TO 717
60 ASSIGN @Response TO 718
70 !
80 First=0
```

```
90   Last=100
100  !
110  CLEAR @Stimulus
120  CLEAR @Response
130  !
140  OUTPUT @Stimulus;"*RST"
150  OUTPUT @Response;"*RST"
160  !
170  PRINT "Voltage Controlled Oscillator Test"
180  PRINT
190  !
200  PRINT "Source Used ..."
210  OUTPUT @Stimulus;"*IDN?"
220  ENTER @Stimulus;Id$
230  PRINT Id$
240  PRINT
250  !
260  PRINT "Counter Used ..."
270  OUTPUT @Response;"*IDN?"
280  ENTER @Response;Id$
290  PRINT Id$
300  PRINT
310  !
320  OUTPUT @Stimulus;":OUTPUT ON"
330  !
340  PRINT
350  PRINT "INPUT [mv]","OUTPUT [kHz]"
360  PRINT "-----","-----"
370  PRINT
380  !
390  FOR Testpoint=First TO Last
400    OUTPUT @Stimulus;":SOURCE:VOLT ";VAL$(Testpoint/1000);";*0PC?"
410    ENTER @Stimulus;Dummy
420    OUTPUT @Response;":MEAS:FREQ?"
430    ENTER @Response;Reading
440    PRINT Testpoint,Reading/1000
450  NEXT Testpoint
460  !
470  OUTPUT @Source;":OUTPUT OFF"
480  END
```

## Program Comments

Lines 20 to 70: Declare variables and I/O paths for instruments. I/O paths let you use a name for an instrument in **OUTPUT** and **ENTER** statements, instead of a numeric address.

80 to 100: Assign values to the input test limits in mV.

110 to 130: Clear the instrument GPIB interfaces.

140 to 160: Reset each instrument to a known measurement state.

170 to 190: Print the test report title.

200 to 310: Query measurement instruments' identifications for test traceability.

320 to 330: Connect the source output signal to the output terminals.

340 to 380: Print results table header.

390 to 460: This is the main measurement loop. Line 400 contains two commands. **:SOURce:VOLT** sets the output level of the source. **\*OPC?** is used to signal that the preceding command has finished executing. To make an accurate measurement, the source output must be allowed to settle. When the output has settled, **\*OPC?** places a 1 in the source Output Queue. The program waits at line 410 until the 1 returned by **\*OPC?** is entered.

Note that following each **OUTPUT** containing a query is an **ENTER** to retrieve the queried value. If you do not use paired **OUTPUTs** and **ENTERs**, you can overwrite data in the instrument Output Queue and generate instrument errors.

470 to 480: Disconnect output terminals of the instruments from the unit under test, and end the program. All HTBasic programs must have **END** as the last statement of the main program.

# Details of Commands and Responses

This subsection describes the syntax of SCPI commands and responses. It provides many examples of the data types used for command parameters and response data. The following topics are explained:

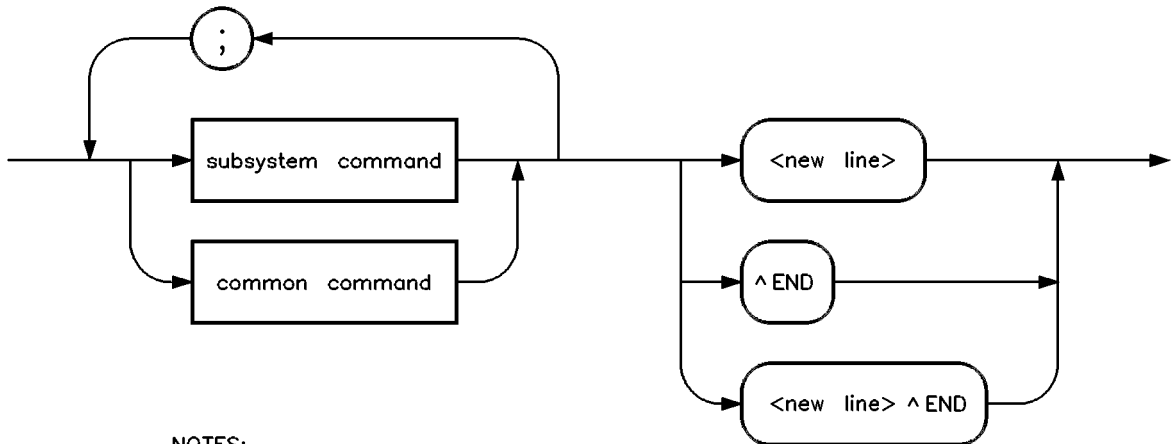
<b>Program Message Syntax</b>	These paragraphs explain how to properly construct the messages you send from the computer to instruments.
<b>Response Message Syntax</b>	These paragraphs discuss the format of messages sent from instruments to the computer.
<b>SCPI Data Types</b>	These paragraphs explain the types of data contained in program and response messages.



---

## Program Message Syntax

These program messages contain commands combined with appropriate punctuation and program message terminators.



**NOTES:**

<new line> = ASCII character decimal 10

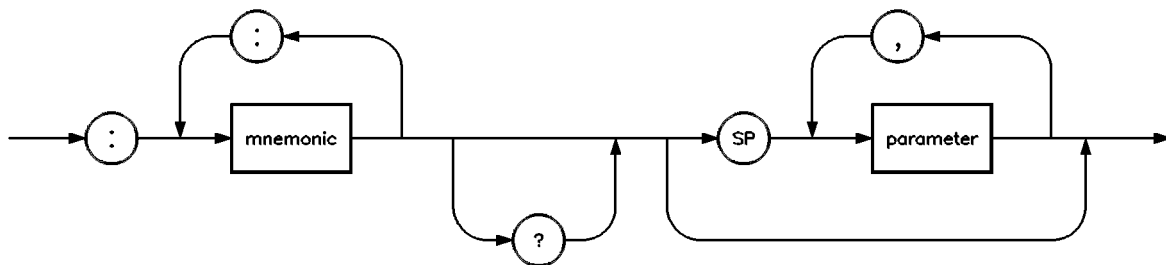
^END = EOI asserted concurrent with last byte

**Figure 1-6. Simplified Program Message Syntax**

As Figure 1-6 shows, you can send common commands and subsystem commands in the same message. If you send more than one command in the same message, you must separate them with a semicolon. You must always end a program message with one of the three program message terminators shown in Figure 1-6. Use <new line>, <^END>, or <new line> <^END> as the program message terminator. The word <^END> means that EOI is asserted on the GPIB interface at the same time the preceding data byte is sent. Most programming languages send these terminators automatically. For example, if you use the HTBasic OUTPUT statement, <new line> is automatically sent after your last data byte. If you are using a PC, you can usually configure the system to send whatever terminator you specify.

---

## SCPI Subsystem Command Syntax



**NOTE:**

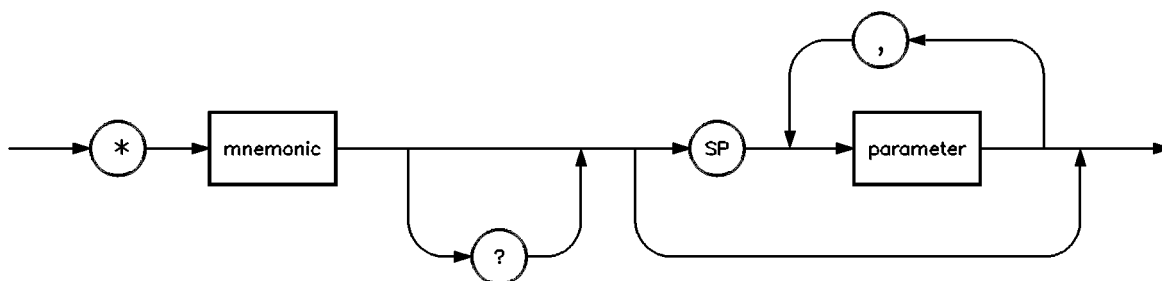
SP = white space, ASCII characters  $0_{10}$  to  $9_{10}$  and  $11_{10}$  to  $32_{10}$

**Figure 1-7. SCPI Simplified Subsystem Command Syntax**

As Figure 1-7 shows, there must be a `<space>` between the last command mnemonic and the first parameter in a subsystem command. This is one of the few places in SCPI where `<space>` is required. Note that if you send more than one parameter with a single command, you must separate adjacent parameters with a comma. Parameter types are explained later in this subsection.

---

## Common Command Syntax

**NOTE:**

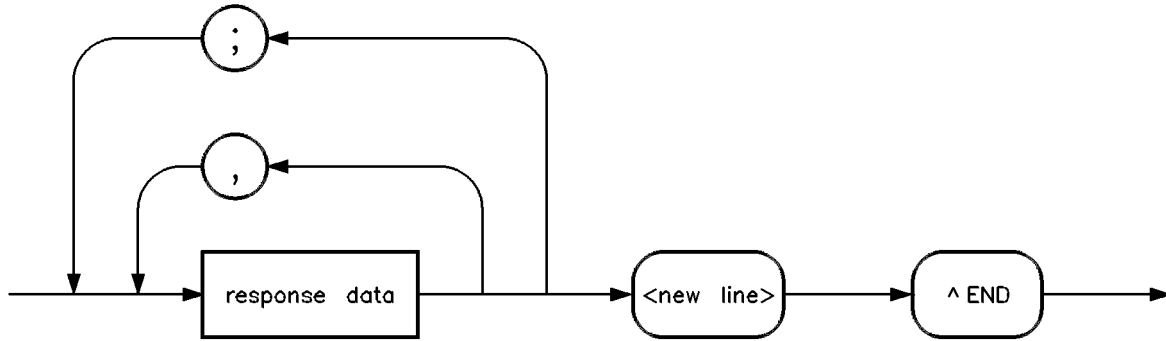
SP = white space, ASCII characters  $0_{10}$  to  $9_{10}$  and  $11_{10}$  to  $32_{10}$

**Figure 1-8. Simplified Common Command Syntax**

As with subsystem commands, use a `<space>` to separate a command mnemonic from subsequent parameters. Separate adjacent parameters with a comma. Parameter types are explained later in this subsection.

---

## Response Message Syntax



**Figure 1-9. Simplified Response Message Syntax**

Response messages can contain both commas and semicolons as separators. When a single query command returns multiple values, a comma separates each data item. When multiple queries are sent in the same message, the groups of data items corresponding to each query are separated by a semicolon. For example, the fictitious query `:QUERY1?:QUERY2?` might return a response message of:

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

Response data types are explained later in this subsection. Note that `<new line><^END>` is always sent as a response message terminator.

---

## SCPI Data Types

These paragraphs explain the data types available for parameters and response data. They list the types available and present examples for each type. SCPI defines different data formats for use in program messages and response messages. It does this to accommodate the principle of forgiving listening and precise talking. Recall that forgiving listening means instruments are flexible, accepting commands and parameters in various formats. Precise talking means an instrument always responds to a particular query in a predefined, rigid format. Parameter data types are designed to be flexible in the spirit of forgiving listening. Conversely, response data types are defined to meet the requirements of precise talking.

**Table 1-2. SCPI Data Types**

Parameter Types	Response Data Types
Numeric	Real or Integer
Extended Numeric	Integer
Discrete	Discrete
Boolean	Numeric Boolean
String	String
Block	Definite Length Block Indefinite Length Block
Non-decimal Numeric	Hexadecimal Octal Binary

Notice that each parameter type has one or more corresponding response data types. For example, a setting that you program using a numeric parameter returns either real or integer response data when queried. Whether real or integer response data is returned depends on the instrument used. However, precise talking requires that the response data type be clearly defined for a particular instrument and query. The instrument command

dictionary generally contains information about data types for individual commands. The following paragraphs explain each parameter and response data type in more detail.

---

## Parameter Types

### Numeric Parameters

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

If an instrument setting programmed with a numeric parameter can only assume a finite number of values, the instrument automatically rounds the parameter. For example, if an instrument has a programmable output impedance of 50 or 75 ohms, and you specified 76.1 for output impedance, the value is rounded to 75. If the instrument setting can only assume integer values, it automatically rounds the value to an integer. For example, sending \*ESE 10.123 is the same as sending \*ESE 10.

Examples of numeric parameters:

100	<i>no decimal point required</i>
100.	<i>fractional digits optional</i>
-1.23	<i>leading signs allowed</i>
4.56e<space>3	<i>space allowed after e in exponentials</i>
-7.89E-01	<i>use either E or e in exponentials</i>
+256	<i>leading + allowed</i>
.5	<i>digits left of decimal point optional</i>

Extended Numeric  
Parameters

Most measurement related subsystems use extended numeric parameters to specify physical quantities. *Extended numeric* parameters accept all numeric parameter values and other special values as well. All extended numeric parameters accept **MAXimum** and **MINimum** as values. Other special values, such as **UP** and **DOWN** may be available as documented in the instrument's command dictionary. Note that **MINimum** and **MAXimum** can be used to set or query values. The query forms are useful for determining the range of values allowed for a given parameter.

In some instruments, extended numeric parameters accept engineering unit suffixes as part of the parameter value. Refer to the command summary to see if this capability exists.

Note that extended numeric parameters are not used for common commands or **STATus** subsystem commands.

Examples of extended numeric parameters:

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

## Discrete Parameters

Use *discrete parameters* to program settings that have a finite number of values. Discrete parameters use mnemonics to represent each valid setting. They have a long and a short form, just like command mnemonics. You can use mixed upper and lower case letters for discrete parameters.

Examples of discrete parameters used with the ROscillator subsystem:

INTernal *internal frequency standard*  
EXTernal *external frequency standard*  
NONE *no frequency standard, free run mode*

Although discrete parameters values look like command keywords, do not confuse the two. In particular, be sure to use colons and spaces properly. Use a colon to separate command mnemonics from each other. Use a space to separate parameters from command mnemonics.

## Boolean Parameters

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

Examples of Boolean parameters:

ON *Boolean TRUE, upper/lower case allowed*  
OFF *Boolean FALSE, upper/lower case allowed*  
1 *Boolean TRUE*  
0 *Boolean FALSE*



---

## Response Data Types

### Real Response Data

A large portion of all measurement data are formatted as *real* response data. Real response data are decimal numbers in either fixed decimal notation or scientific notation. In general, you do not need to worry about the rules for formatting real data, or whether fixed decimal or scientific notation is used. Most high level programming languages that support instrument I/O handle either type transparently.

Examples of real response data:

```
1.23E+0
-1.0E+2
+1.0E+2
0.5E+0
1.23
-100.0
+100.0
0.5
```

### Integer Response Data

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

Examples of integer response data:

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

**Details of Commands and Responses**

## Discrete Response Data

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

Examples of discrete response data:

INTernal	<i>level internally</i>
DIODe	<i>level using an external diode</i>
PMETer	<i>level using an external power meter</i>
MMHead	<i>level using a mm-wave source module</i>

## String Response Data

*String* response data are similar to string parameters. The main difference is that string response data use only double quotes as delimiters, rather than single quotes. Embedded double quotes may be present in string response data. Embedded quotes appear as two adjacent double quotes with no characters between them.

Examples of string response data:

```
"This IS valid"  
"SO IS THIS "" "  
"I said, ""Hello!"""
```

This subsection illustrates how the general SCPI concepts presented in previous subsections apply to programming real measurements. To introduce you to programming with SCPI, we must list the commands for the sweeper. We will begin with a simplified example.

---

## Using the Example Programs

The example programs are interactive. They require active participation by the operator. If you desire to get an understanding of the principles without following all of the instructions, read the “Program Comments” paragraphs to follow the programmed activity.

The GPIB select code is assumed to be preset to 7. All example programs in this section expect the sweeper’s GPIB address to be decimal 19.

To find the present GPIB address use the front panel.

Press **SHIFT** **LOCAL**.

The active entry area indicates the present decimal address. If the number displayed is not 19, press **1** **9** **ENTER** to reset it to 19.

Now check that the interface language is set to SCPI. Press **SHIFT** **RECALL** **1** **5** **ENTER**. The selected interface language is then shown, use the up and down keys to change the language.

---

## Use of the Command Tables

In Table 1-3, notice that a new column titled “Allowed Values” has been added to the command table. This column lists the specific values or range of values allowed for each parameter. A vertical bar (|) separates values in a list from which you must choose one value. The commands listed in the table are only part of all the available SCPI commands of the sweeper. For a complete listing of the programming codes see Chapter 2.

**Table 1-3. Sample Sweeper Commands**

<b>Command</b>	<b>Parameters</b>	<b>Parameter Type</b>	<b>Allowed Values</b>
:CALibration :PMEter :FLATness :INITiate? :NEXT?	flatness array to cal measured power	discrete extended numeric	USER DIODE PMEter MMHead INternal EXternal <num> [lvl suffix]
:CORRection :FLATness	Up to 801 freq- correction pairs	extended numeric	{ <num> [freq suffix]
:FREQuency :CENter [:CW] :AUTO :MODE :STARt :STEP [:INCRement] :STOP	center freq CW freq coupled to center freq free mode start freq freq step stop freq	extended numeric extended numeric Boolean discrete extended numeric extended numeric extended numeric	specified freq range or MAXimum MINimum UP DOWN specified freq range or MAXimum MINimum UP DOWN ON OFF 1 0 CW SWEep FIX SW CW specified freq range or MAXimum MINimum UP DOWN <num> [freq suffix] or MAXimum MINimum specified freq range or MAXimum MINimum UP DOWN
[n] is 0 to 9, 1 is the default			
:MARKer[n] :FREQuency	marker frequency	extended numeric	specified freq range or MAXimum MINimum

Table 1-3. Sample Sweeper Commands (continued)

Command	Parameters	Parameter Type	Allowed Values
:POWer			
:ATTenuation	atten setting	extended numeric	0 to 70 [DB] or MAXimum MINimum UP DOWN
:AUTO	coupled atten	Boolean	ON OFF 1 0
[:LEVel]	output level	extended numeric	specified power range or MAXimum MINimum UP DOWN
:STATe	RF on/off	Boolean	ON OFF 1 0
:SWEEp			
:GENeration	type of sweep	discrete	STEPped ANALog
:TIME	sweep time	extended numeric	<num> [time suffix] MAXimum MINimum
:AUTO	auto sweep time switch	Boolean	ON OFF 1 0
:LLIMit	fastest sweep time	extended numeric	<num> [time suffix] or MAXimum MINimum

---

## GPIB Check, Example Program 1

This first program is to verify that the GPIB connections and interface are functional. Connect a controller to the sweeper via an GPIB cable. Clear and reset the controller and type in the following program:

```
10 Source=719
20 ABORT 7
30 LOCAL Source
40 CLEAR Source
50 REMOTE Source
60 CLS
70 PRINT "The source should now be in REMOTE."
80 PRINT "Verify that the 'REMOTE' LED is on."
90 END
```

Run the program and verify that the REMOTE LED is lit on the sweeper. If it is not, verify that the sweeper address is set to 19 and that the interface cable is properly connected.

If the controller display indicates an error message, it is possible that the program was entered in incorrectly. If the controller accepts the REMOTE statement but the sweeper REMOTE LED does not turn on, perform the operational checks as outlined in the respective Operating and Service Manuals to find the defective device.

### Program Comments

- 10: Setup a variable to contain the GPIB address of the source.
- 20: Abort any bus activity and return the GPIB interfaces to their reset states.
- 30: Place the source into LOCAL to cancel any Local Lockouts that may have been setup.
- 40: Reset the source's parser and clear any pending output from the source. Prepare the source to receive new commands.
- 50: Place the source into REMOTE.
- 60: Clear the display of the computer.
- 70: Print a message to the computer's display.

---

## Local Lockout Demonstration, Example Program 2

When the sweeper is in REMOTE mode, all the front panel keys are disabled except the LOCAL key. But, when the LOCAL LOCKOUT command is set on the bus, even the LOCAL key is disabled. The LOCAL command, executed from the controller, is then the only way to return all (or selected) instruments to front panel control.

Continue example program 1. Delete line 90 END and type in the following commands:

```

90  PRINT "Verify that all keys are ignored,
      except the 'LOCAL' key."
100 PRINT "Verify that 'LOCAL' causes the
      REMOTE LED to go OFF."
110 PRINT " . . . . . press CONTINUE"
120 PAUSE
130 REMOTE Source
140 LOCAL LOCKOUT 7
150 PRINT
160 PRINT "Source should now be in LOCAL LOCKOUT mode."
170 PRINT "Verify that all keys (including 'LOCAL')
      have no effect."
180 PRINT " . . . . . press CONTINUE"
190 PAUSE
200 LOCAL Source
210 PRINT
220 PRINT "Source should now be in LOCAL mode."
230 PRINT "Verify that the sweeper's keyboard
      is functional."
240  END

```

To verify and investigate the different remote modes do the following:

1. Reset the controller.
2. On the sweeper: Press **PRESET**.
3. Clear the controller display and run the program.
4. Verify that the REMOTE LED on the sweeper is lit.



5. From the front panel, attempt to change the start frequency and verify that this is impossible.
6. Verify that all keys except **LOCAL** are disabled.
7. Now press the **LOCAL** key and verify that the sweeper REMOTE LED is off and that you can modify any of the sweep functions.
8. Execute a “continue” on the controller. With the controller displaying “LOCAL LOCKOUT mode”, verify that the sweeper REMOTE LED is again lit.
9. Attempt to change the start frequency and press **PRESET**. Verify that this is impossible.
10. Now press the sweeper **LOCAL** key and verify that still no action is taken.
11. Execute a “continue” on the controller. With the controller displaying “LOCAL mode”, verify that the sweeper REMOTE LED is off. Also verify that all sweep functions now can be modified via the front panel controls.

#### HINT

Note that the sweeper **LOCAL** key produces the same result as programming LOCAL 719 or LOCAL 7. Be careful because the LOCAL 7 command places all instruments on the GPIB in the local state as opposed to just the sweeper.

Program Comments	90 to 120:	Print a message on the computer's display, then pause.
	130:	Place the source into REMOTE.
	140:	Place the source into LOCAL LOCKOUT mode.
	150 to 190:	Print a message on the computer's display, then pause.
	200:	Return the source to local control.
	210 to 230:	Print a message on the computer's display.

---

## Setting Up A Typical Sweep, Example Program 3

In swept operation, the sweeper is programmed for the proper sweep frequency range, sweep time, power level, and marker frequencies for a test measurement. This program sets up the sweeper for a general purpose situation. The instrument is the same as in program 1. Clear and reset the controller and type in the following program:

```
10 Source=719
20 ABORT 7
30 LOCAL 7
40 CLEAR Source
50 REMOTE Source
60 OUTPUT Source;"*RST"
70 OUTPUT Source;"FREQuency:MODE SWEEp"
80 OUTPUT Source;"FREQuency:START 4 GHZ"
90 OUTPUT Source;"FREQ:STOP 7 GHz"
100 OUTPUT Source;"POWer:LEVel -5 DBM"
110 OUTPUT Source;"SWEEp:TIME 500MS"
120 OUTPUT Source;":MARKer1:STATe ON;FREQuency 4.5GHZ"
130 OUTPUT Source;"MARKer2:STATe ON;FREQuency 6111E6"
140 OUTPUT Source;"*OPC?"
150 ENTER Source;X
160 OUTPUT Source;"POWer:STATe ON"
170 OUTPUT Source;"INITiate:CONTinuous ON"
180 CLS
190 PRINT "Source setup complete."
200 PRINT "Verify that the source is sweeping from"
210 PRINT "4 GHz to 7 GHz at a power of -5 dBm,"
220 PRINT "with a sweeptime of 0.5 seconds."
230 END
```

Run the program.

Program Comments	10:	Assign the source's GPIB address to a variable.
	20 to 50:	Abort any GPIB activity and initialize the GPIB interface.
	60:	Set the source to its initial state for programming. The *RST state is <i>not</i> the same as the PRESET state. For complete details of the instrument state at *RST, see "SCPI Command Summary," in Chapter 2.
	70:	Select the frequency mode to be SWEEP instead of the default sweep mode of "CW" that was selected with *RST.
	80:	Set the source start frequency to 4 GHz.
	90:	Set the source stop frequency to 7 GHz. Note the optional usage of the short-form mnemonic, "FREQ".
	100:	Set the source's power level to $-5$ dBm.
	110:	Set the sweep time to 500 ms. Notice that upper/lower case in commands does not matter. Also spaces before the suffix ("MS") are not required in SCPI.
	120 and 130:	Set markers 1 and 2 to a fixed value. Notice that the value for marker 2 does not end with a frequency suffix. Hertz is a default terminator and is understood.
	140:	Wait until the source has completed setting up the commands that have been sent so far before turning on the output.
	150:	The ENTER statement causes the program to wait here until the source responds to the previous *OPC? with a '1'.
	160:	The source has now completed processing the commands. The RF frequency, power, and markers are at their programmed values. Turn on the RF output of the source.
	170:	Select a continuously initiated sweep instead of the default mode of non-continuous that was selected with *RST.
	180:	Clear the computer's display.
	190 to 220:	Print a message on the computer's display.

---

## Queries, Example Program 4

The following example demonstrates the use of query commands and response data formats. Clear and reset the controller and type in the following program:

```
10 Source=719
20 ABORT 7
30 LOCAL 7
40 CLEAR Source
50 REMOTE Source
60 CLS
70 OUTPUT Source;"*RST"
80 OUTPUT Source;"POWER:LEVEL -5 dBm;STATE ON"
90 OUTPUT Source;"FREQ:CW?"
100 ENTER Source;F
110 PRINT "Present source CW frequency is : ";F/1.E+6;"MHz"
120 OUTPUT Source;"POWER:STATE?"
130 ENTER Source;W
140 PRINT "Present power ON/OFF state is : ";W
150 OUTPUT Source;"FREQ:MODE?"
160 DIM A$[10]
170 ENTER Source;A$
180 PRINT "Source's frequency mode is : "&A$
190 OUTPUT Source;"FREQ:CW? MIN"
200 ENTER Source;A
210 PRINT "Minimum source CW frequency is : ";A/1.E+6;"MHz"
220 OUTPUT Source;"FREQ:START?;STOP?"
230 ENTER Source;X,Y
240 PRINT "Swept frequency limits : "
250 PRINT "  Start ";X/1.E+6;"MHz"
260 PRINT "  Stop  ";Y/1.E+6;"MHz"
270 END
```

Run the program.

Program Comments

- 10: Assign the source's GPIB address to a variable.
- 20 to 50: Abort any GPIB activity and initialize the GPIB interface.

- 60: Clear the computer's display.
- 70: Set the source to its initial state for programming.
- 80: Setup the source power level using a compound message.
- 90: Query the value of the source's CW frequency.
- 100: Enter the query response into the variable 'F'. The response always is returned in fundamental units, Hz in the case of frequency.
- 110: Print the CW Frequency in MHz on the computer display.
- 120: Query the value of a boolean function, POWER:STATE.
- 130: Enter the query response into a variable 'W'. Boolean responses are always '1' for ON and '0' for OFF.
- 140: Print the value of the POWER:STATE on the computer display.
- 150: Query the value of a discrete function (FREQ:MODE).
- 160: Dimension a string variable to contain the response.
- 170: Enter the response into A\$. The response will be a string that represents the function's present value.
- 180: Print the value of A\$ on the computer display.
- 190: Example usage of a MIN query. This will request the minimum value that the FREQ:CW function can be programmed to. This has no effect on the current value.
- 200: Enter the numeric response into the variable A.
- 210: Print the value of A on the computer display.
- 220: This is compound query. Up to 8 parameters can be queried from the sweeper at one time using this method. In this example, the start and stop frequencies are interrogated.
- 230: The responses are read back into the variables X and Y. The order of the responses is the same as the order of the queries. X will contain the START frequency and Y will contain the STOP.
- 240 to 260: Print the START/STOP frequencies on the display.

---

## Saving and Recalling States, Example Program 5

When a typical sweep, like example program 3, is set up, the complete front panel state may be saved for later use in non-volatile memories called registers 1 through 9. This can be done remotely as a part of a program. Clear and reset the controller and type in the following program:

```
10  Source=719
20  ABORT 7
30  LOCAL 7
40  CLEAR Source
50  REMOTE Source
60  CLS
70  OUTPUT Source;"*RST;FREQ:MODE SWE;STAR 4GHZ
    ;STOP 5GHZ;;INIT:CONT ON"
80  OUTPUT Source;"*SAV 1"
90  CLS
100 PRINT "A sweeping state has been saved in REGISTER 1."
110 OUTPUT Source;"*RST;FREQ:CW 1.23456GHZ;;POW:LEV -1DBM"
120 OUTPUT Source;"*SAV 2"
130 PRINT "A CW state has been saved in REGISTER 2."
140 PRINT "..... Press Continue"
150 PAUSE
160 OUTPUT Source;"*RCL 1"
170 PRINT "Register 1 recalled. Verify source is sweeping."
180 PRINT "Press Continue."
190 PAUSE
200 OUTPUT Source;"*RCL 2"
210 PRINT "Register 2 recalled."
220 PRINT "Verify source is in CW mode."
230 END
```

Run the program.

Program Comments	10:	Assign the source's GPIB address to a variable.
	20 to 50:	Abort any GPIB activity and initialize the GPIB interface.
	60:	Clear the computer's display.
	70:	Setup the source for a sweeping state. Note the combination of several commands into a single message. This single line is equivalent to the following lines :
		<pre> OUTPUT Source; "*RST" OUTPUT Source; "FREQ:MODE SWEEp" OUTPUT Source; "FREQ:START 4 GHZ" OUTPUT Source; "FREQ:STOP 5 GHZ" OUTPUT Source; "INIT:CONT ON" </pre>
	80:	Save this state into storage register 1.
	90:	Clear the computer display.
	100:	Print a message on the computer display.
	110:	Setup the source for a CW state. Note the combination of several commands into a single message. This single line is equivalent to the following lines :
		<pre> OUTPUT Source; "*RST" OUTPUT Source; "FREQ:CW 1.23456 GHZ" OUTPUT Source; "POWer:LEVel -1 DBM" </pre>
	120:	Save this state into storage register 2.
	130 to 150:	Print a message on the computer display and pause.
	160:	Recall the instrument state from register 1. It should contain the sweeping state.
	170 to 190:	Print a message on the computer display and pause.
	200:	Recall the instrument state from register 2. It should contain the CW state.
	210 and 220:	Print messages on the computer display.

---

## Looping and Synchronization, Example Program 6

Clear and reset the controller and type in the following program:

```
10  Source=719
20  ABORT 7
30  LOCAL 7
40  CLEAR Source
50  REMOTE Source
60  CLS
70  OUTPUT Source;"*RST"
80  OUTPUT Source;"FREQ:START 4 GHZ; STOP 5 GHZ; MODE SWEEP"
90  OUTPUT Source;"POWER:LEVEL -1 DBM; STATE ON"
100 OUTPUT Source;"SWEEP:TIME 1"
110 OUTPUT Source;"*OPC?"
120 ENTER Source;X
130 REPEAT
140     DISP "Enter number of sweeps to take : [0 to exit]";
150     INPUT N
160     IF N>0 THEN
170         FOR I=1 TO N
180             DISP "Taking sweep number : ";I
190             OUTPUT Source;"INIT:IMM;*OPC?"
200             ENTER Source;X
210         NEXT I
220     END IF
230 UNTIL N=0
240 END
```

Run the program.



Program Comments	10:	Assign the source's GPIB address to a variable.
	20 to 50:	Abort any GPIB activity and initialize the GPIB interface.
	60:	Clear the computer's display.
	70:	Set the source to its initial state for programming.
	80:	Setup the frequency parameters using a compound message.
	90:	Setup the source's power level and state using a compound message.
	100:	Setup the source's sweep time to 1 second.
	110:	Send the *OPC? command to the source to ensure that the previous commands are completed and the source is ready to begin controlled sweeps.
	120:	Enter the response to the *OPC? into the variable X. The response should be a '1'.
	130:	Start of the loop.
	140 and 150:	Prompt the operator for the number of sweeps to take. The number of sweeps to take is stored in the variable N. Enter 0 to quit the program.
	160:	Don't take any sweeps if N is less than or equal to 0.
	170:	Start a FOR/NEXT loop to take N sweeps.
	180:	Display the number of this sweep on the computer display.
	190:	Initiate a single sweep on the source and then wait until the pending operation is complete. Return a '1' when the sweep completes.
	200:	Enter the response to the *OPC? into the variable X. The program execution will halt on this ENTER statement until the sweep is finished.
	210:	Repeat the INIT:IMM sequence N times.
	220:	End of the IF statement to skip sweeps if N is less than or equal to 0.
	230:	Exit the program if the value of N is 0.

---

## Using the \*WAI Command, Example Program 7

The following example illustrates the use of the \*WAI command to cause the sweeper to perform a synchronous sweep.

```
10 Source=719
20 ABORT 7
30 LOCAL 7
40 CLEAR Source
50 REMOTE Source
60 CLS
70 OUTPUT Source;"*RST"
80 OUTPUT Source;"FREQ:STAR 4GHZ; STOP 5GHZ; MODE SWE"
90 OUTPUT Source;"SWE:TIME 2"
100 OUTPUT Source;"*OPC?"
110 ENTER Source;X
120 FOR I=1 TO 4
130     OUTPUT Source;"INIT"
140     OUTPUT Source;"*WAI"
150     OUTPUT Source;"POW:STAT ON"
160     OUTPUT Source;"INIT"
170     OUTPUT Source;"*WAI"
180     OUTPUT Source;"POW:STAT OFF"
190 NEXT I
200 PRINT "Finished sending commands to source."
210 PRINT "Note that execution is continuing for four cycles."
220 END
```

Run the program.

Program Comments	10:	Assign the source's GPIB address to a variable.
	20 to 50:	Abort any GPIB activity and initialize the GPIB interface.
	60:	Clear the computer's display.
	70:	Set the source to its initial state for programming.
	80:	Set the source up for a sweep, from 4 GHz to 5 GHz.
	90:	Set the sweep time to 2 second. In SCPI, suffixes are optional if you program in fundamental units (for sweep time, that would be seconds).
	100:	Send an *OPC? to the source.
	110:	Enter the query response to the *OPC? into a variable "X". The program execution will halt here until the source has finished processing all the commands up to this point. Once complete, the source will respond to the *OPC? with a "1".
	120:	Begin a FOR/NEXT loop that is repeated four times.
	130:	Initiate a sweep on the source.
	140:	Send a *WAI command to the source. This command causes the source to stop executing new commands until all prior commands and operations have completed execution. In this case, there is a sweep in progress, so no further commands will be executed until the sweep finishes.
	150:	Turn the RF output of the source ON.
	160:	Initiate a sweep on the source.
	170:	Send another *WAI to the source. Although the *WAI command causes EXECUTION of commands to be held off, it has no effect on the transfer of commands over the GPIB. The commands continue to be accepted by the source and are buffered until they can be executed.
	180:	Toggle the RF STATE to OFF.
	190:	Repeat the sample exercise.
	200 and 210:	Print messages on the computer display.

---

## Using the User Flatness Correction Commands, Example Program 8

The following program interrogates the sweeper and an 437B power meter for frequency and power information respectively. The sweeper is programmed to sweep from 2 to 20 GHz, with frequency-correction pairs every 100 MHz and 0 dBm leveled output power. For this example, we assume that the path losses do not exceed 10 dB and that the 437B power meter already has its power sensor's calibration factors stored in sensor data table 0. If another power meter is used, the power sensor's calibration factors will have to be stored in a look-up table. Modify the program to suit your particular measurement requirements. Up to 801 points may be entered in the user flatness correction table with this program. SCPI commands are used to set up the source parameters and enter correction frequencies and data into the correction table.

```
10 !ASSIGN THE ADDRESS TO THE SOURCE AND POWER METER
20 DIM A$[5000],B$[5000]
30 ASSIGN @Source TO 719
40 ASSIGN @Meter TO 713
50 INTEGER Error_flag
60 ABORT 7
70 !
80 !SET UP SOURCE
90 OUTPUT @Source;"*RST"
100 OUTPUT @Source;"FREQ:MODE SWE;STAR 2 GHZ;STOP 20 GHZ"
110 OUTPUT @Source;"SWEEP:TIME 200 MS"
120 OUTPUT @Source;"POW:LEV 5 DBM;:INIT:CONT ON"
130 OUTPUT @Source;"*OPC?"
140 ENTER @Source;Done
150 !
160 !SET UP POWER METER
170 OUTPUT @Meter;"PR"
180 OUTPUT @Meter;"FA"
190 OUTPUT @Meter;"TRO"
200 !
210 !ZERO POWER METER
220 OUTPUT @Source;"POW:STAT OFF"
```

```
230 Zero_meter(@Meter,@Source,Error_flag)
240 IF Error_flag THEN
250     BEEP
260     CLEAR SCREEN
270     PRINT "ERROR:METER DID NOT COMPLETE ZEROING OPERATION!"
280 ELSE
290 !
300 !SET UP CORRECTION FREQUENCIES IN USER FLATNESS CORRECTION TABLE
310 !OUTPUT @Source;"CORR:FLAT ";
320     Start_freq=2
330     Stop_freq=20
340     Increment=1
350     N=(((Stop_freq-Start_freq)/Increment)+1)
360     Freq=Start_freq
370     FOR I=1 TO N
380     A$=A$&VAL$(Freq)&"GHZ,Odb,"
390     Freq=Freq+Increment
400     NEXT I
410     B=LEN(A$)
420     B=B-1
430     B$=A$[1,B]
440     OUTPUT @Source;"CORR:FLAT ";B$
450 ! OUTPUT @Source;"POW:STAT ON"
460 !
470 !ENTER DATA IN USER CORRECTION TABLE
480     OUTPUT @Source;"CAL:PMET:FLAT:INIT? USER"
490     ENTER @Source;Freq
500     WHILE Freq>0
510     Power=FNRead_meter(@Meter,Freq)
520     OUTPUT @Source;"CAL:PMET:FLAT:NEXT? ";VAL$(Power);"DBM"
530     ENTER @Source;Freq
540     END WHILE
550 END IF
560 END
570 !
580 SUB Zero_meter(@Meter,@Source,INTEGER Error_flag)
590 OUTPUT @Source;"Pow:stat off"
600     OUTPUT @Meter;"CS"
610     OUTPUT @Meter;"ZE"
620     Max_attempts=30
630     Attempts=0
```

## Programming Typical Measurements

```

640     Zeroing=1590
650     Finished=0
660     WHILE Zeroing AND NOT Finished
670         Attempts=Attempts+1
680         Meter_stat=SPOLL(@Meter)
690         IF Attrmpts>Max_attempts THEN Zeroing=0
700         IF BIT(Meter_stat,1) THEN Finished=1
710         WAIT 1
720     END WHILE
730     OUTPUT @Source;"Pow:stat on"
740     IF NOT Zeroing THEN
750         Error_flag=1
760     ELSE
770         Error_flag=0
780     END IF
790     SUBEND
800     !
810     DEF FNRead_meter(@Meter,Freq)
820         OUTPUT @Meter;"SEOEN"
830         Freq$=VAL$(Freq)
840         OUTPUT @Meter;"FR"&Freq$&"GZ"i
850         OUTPUT @Meter;"TR2"
860         ENTER @Meter;Power$
870         PO=VAL(Power$)
880         Flips=0
890         Slope=0
900         REPEAT
910             OUTPUT @Meter;"TR2"
920             ENTER @Meter;Power$
930             P1=VAL(Power$)
940             Slope2=SGN(PO-P1)
950             IF Slope2Slope THEN
960                 Flips=Flips+1
970                 Slope2=Slope
980             ELSE
990                 IF Slope2=0 THEN Flips=Flips+.2
1000            END IF

```

```
1010      P0=P1
1020      UNTIL Flips>=3
1030      Power=(P0+P1)/2
1040      RETURN Power
1050  FNEND
```

# Programming the Status System

This section discusses the structure of the status system used in SCPI instruments, and explains how to program status registers. An important feature of SCPI instruments is that they all implement status registers the same way. The status system is explained in the following paragraphs:

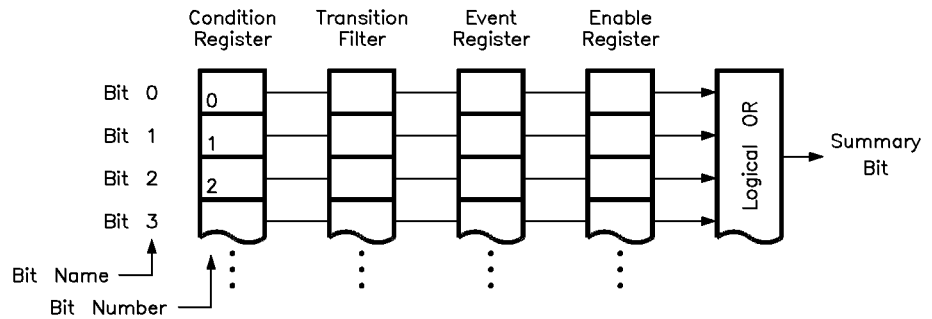
- General Status Register Model** These paragraphs explain the way that status registers are structured in SCPI instruments. It also contains an example of how bits in the various registers change with different input conditions.
- 83750 Series Status Register Model** These paragraphs describe how the status system works in the 83750 Series synthesized sweepers.



## General Status Register Model

The generalized status register model shown in Figure 1-10 is the building block of the SCPI status system. This model consists of a condition register, a transition filter, an event register and an enable register. A set of these registers is called a *status group*.

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



**Figure 1-10. Generalized Status Register Model**

### Condition Register

The *condition register* continuously monitors the hardware and firmware status of the instrument. There is no latching or buffering for this register; it is updated in real time. Condition registers are read-only.

There may or may not be a command to read a particular condition register.

**Programming the Status System**

Transition Filter	The <i>transition filter</i> specifies which types of bit state changes in the condition register will set corresponding bits in the event register. Transition filter bits may be set for positive transitions ( <b>PTR</b> ), negative transitions ( <b>NTR</b> ), or both. Positive means a condition bit changes from 0 to 1. Negative means a condition bit changes from 1 to 0. Transition filters are read-write. Transition filters are unaffected by <b>*CLS</b> (clear status) or queries. They are set to instrument dependent values at power on and after <b>*RST</b> .
Event Register	The <i>event register</i> latches transition events from the condition register, as specified by the transition filter. Bits in the event register are latched, and once set they remain set until cleared by a query or a <b>*CLS</b> (clear status). There is no buffering, so while an event bit is set, subsequent events corresponding to that bit are ignored. Event registers are read-only.
Enable Register	The <i>enable register</i> specifies the bits in the event register that can generate a summary bit. The instrument logically ANDs corresponding bits in the event and enable registers, and ORs all the resulting bits to obtain a summary bit. Summary bits are in turn recorded in the Status Byte. Enable registers are read-write. Querying an enable register does not affect it. There is always a command to read and write to the enable register of a particular status group.

## An Example Sequence

Figure 1-11 illustrates the response of a single bit position in a typical status group for various settings. The changing state of the condition in question is shown at the bottom of the figure. A small binary table shows the state of the chosen bit in each status register at the selected times T1 to T5.

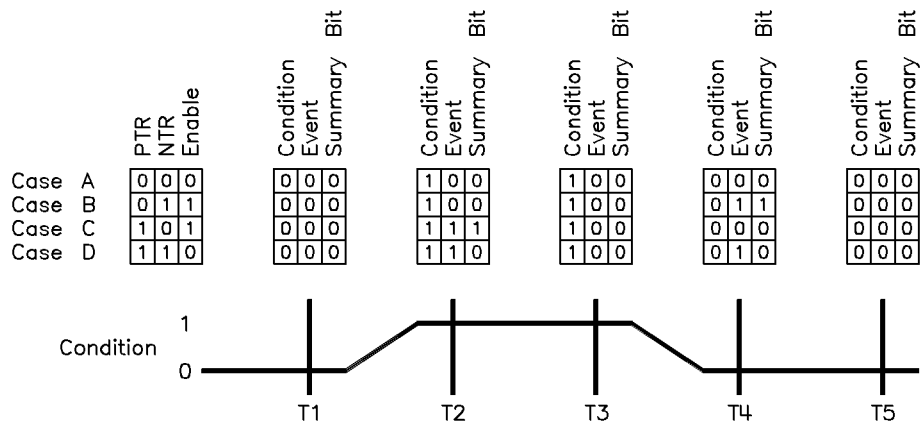


Figure 1-11. Typical Status Register Bit Changes

## 83750 Series Status Register Model

The state of certain instrument hardware and firmware events and conditions in the instrument may be determined by programming the status register system. The status register system is arranged in a hierarchical order. Refer to Figure 1-12. Three lower status groups provide information to the status byte group. The status byte group is used to determine the general nature of an event and the lower status groups are used to determine the specific nature of the event. A “status group” is a set of related registers whose contents are programmed in order to produce status summary bits.

Corresponding bits in the Condition Register are filtered by the Negative and Positive Transition Registers and stored in the Event Register. The contents of the Event Register are logically ANDed with the contents of the Enable Register and the result is logically ORed to produce a status summary bit.

---

## Synthesized Sweeper Status Groups

The synthesized sweeper status register system consists of the Status Byte group and three other status groups that provide input to the Status Byte group. The hierarchy of the 83750 status register system is shown in Figure 1-12. The following paragraphs explain the information that is provided by each status group.

The Status Byte Group

IEEE 488.1 originally defined the Status Byte and provide the Serial Poll to allow controllers to read it. However, other than the RQS bit (bit 6), it did not define how the bits are set or cleared, and also left the definition of the bits up to the device designer.

IEEE 488.2 (to which the Synthesize Sweeper conforms) further defined the Status Byte bit positions, specifically bits 4 and 5. In addition, it defines more commands that allow the user to access the Status Byte and associated data structures. It's important to note that the Serial Poll DOES NOT clear the Status Byte, even though it does clear the RQS bit (bit 6). The byte is cleared by clearing the related status structures. Further, IEEE 488.2 provides the \*CLS command (clear status) which clears all the Event Registers. This causes the bits in the Status Byte to be cleared.

The Status Byte group is used to determine the general nature of an instrument event or condition. The Status Byte group consists of the Service Request Enable register and the Status Byte. The bits in the Status Byte provide you with the following information:

Bit	Description
0 to 2	These bits are always set to 0.
3	1 in this bit position indicates that the Questionable Data summary bit has been set. The Questionable Event register can then be read to determine the specific condition that caused this bit to be set.
4	1 in this bit position indicates that the synthesized sweeper has data ready in its output queue. Note that there are no lower status groups that provide input to this bit.
5	1 in this bit position indicates that the Standard Event summary bit has been set. The Standard Event Status register can then be read to determine the specific event that caused this bit to be set.
6	1 in this bit position indicates that the instrument has at least one reason to require service. This bit is also called the Master Summary Status Bit (MSS). The individual bits in the Status Byte are individually ANDed with their corresponding Service Request Enable Register, then each individual bit value is ORed and input to this bit. See Figure 1-10.
7	1 in this bit position indicates that the Standard Operation summary bit has been set. The Operation Event register can then be read to determine the specific condition that caused this bit to be set.

**Programming the Status System**

The Standard Event Status Group

The Standard Event Status group is used to determine the specific event that set bit 5 in the Status Byte. The Standard Event Status group consists of the Standard Event Status register (an Event register) and the Standard Event Status Enable register. The bits in the Standard Event Status register provide you with the following information:

Bit	Description
0	1 in this bit position indicates that all pending synthesized sweeper operations were completed following execution of the “*OPC” command.
1	This bit is always set to 0. (The 83750 does not request control.)
2	1 in this bit position indicates that a query error has occurred. Query errors have SCPI error numbers from –499 to –400.
3	1 in this bit position indicates that a device dependent error has occurred. Device dependent errors have SCPI error numbers from –399 to –300 and 1 to 32767.
4	1 in this bit position indicates that an execution error has occurred. Execution errors have SCPI error numbers from –299 to –200.
5	1 in this bit position indicates that a command error has occurred. Command errors have SCPI error numbers from –199 to –100.
6	1 in this bit position indicates that at least one front panel key (except the LINE switch) has been pressed. This is true even if the synthesized sweeper is in Local Lockout (LLO) mode.
7	1 in this bit position indicates that the synthesized sweeper has been turned off and then on.

### The Standard Operation Status Group

The Standard Operation status group is used to determine the specific condition that set bit 7 in the Status Byte. The Standard Operation status group consists of the Operation Condition register, Operation Negative Transition register, Operation Positive Transition register, Operation Event register, and Operation Event Enable register. The bits in the Operation Event register provide you with the following information:

Bit	Description
0	1 in this bit position indicates that a calibration is being performed (auto tracking, power meter cal, YIG oscillator cal).
1	1 in this bit position indicates that the synthesized sweeper hardware is settling (for example, the power level is changing).
2	This bit is always set to 0.
3	1 in this bit position indicates that the synthesized sweeper is sweeping (in the process of completing a sweep).
4 to 7	These bits are always set to 0.
8	1 in this bit position indicates that the synthesized sweeper self-test data is ready.
9	1 in this bit position indicates that the synthesized sweeper's display has changed.
10	1 in this bit position indicates that the synthesized sweeper self-test is in progress.
11 to 15	These bits are always set to 0.

The Questionable Data  
Status Group

The Questionable Data status group is used to determine the specific condition that set bit 3 in the Status Byte. The Questionable Data status group consists of the Questionable Condition register, Questionable Negative Transition register, Questionable Positive Transition register, Questionable Event register, and Questionable Event Enable register. The bits in the Questionable Event register provide you with the following information:

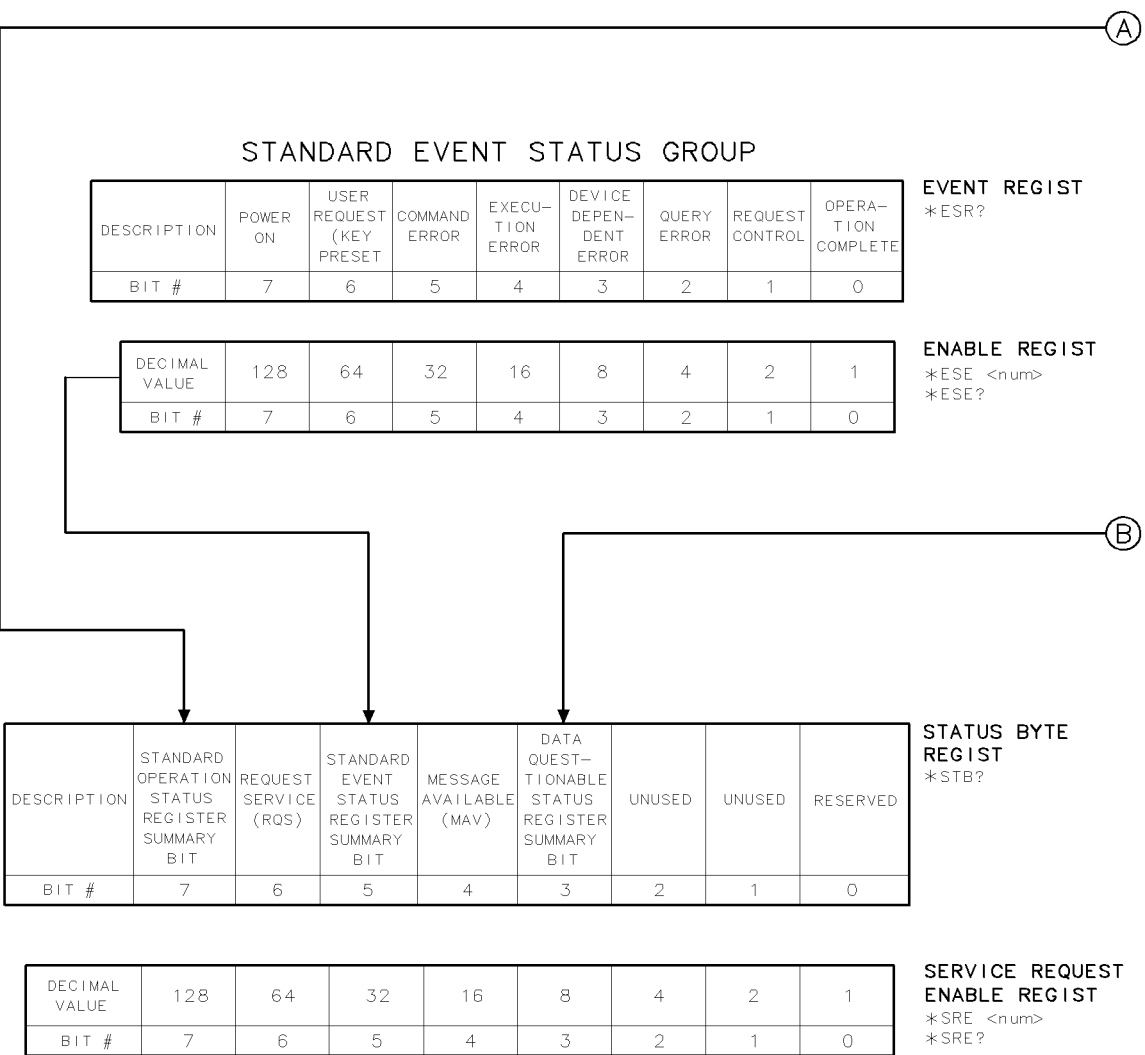
Bit	Description
0 to 2	These bits are always set to 0.
3	1 in this bit position indicates that the automatic leveling control (ALC) is unable to maintain a leveled RF power output (i.e., the RF is unlevelled).
4	1 in this bit position indicates that the internal frequency reference oven is cold.
5	1 in this bit position indicates that one of the synthesized sweeper phaselock loops is unlocked (Fractional N, Reference, or YO Loop).
6 to 7	These bits are always set to 0.
8	1 in this bit position indicates that there has been an error in calibration (auto tracking, power meter cal, or YIG oscillator cal).
9	This bit is set to 1 whenever the self-test has failed.
10	This bit is set to 1 whenever there is a hardware fault detected.
11 to 15	These bits are always set to 0.



Status Register System  
Programming Example

In the following example, the Status Register System is programmed to set bit 6 of the status byte (the SRQ bit) high after the synthesized sweeper hardware has settled. Bit 6 is monitored and, once it is set high, the controller prints "HARDWARE IS SETTLED" on its screen.

```
10 OUTPUT 719;"STAT:OPER:PTR 0"  
20 OUTPUT 719;"STAT:OPER:NTR 2"  
30 OUTPUT 719;"STAT:OPER:ENAB 2"  
40 OUTPUT 719;"*SRE 128"  
50 PRINT "SRQ IS SET UP"  
60 OUTPUT 719;"*CLS"  
70 A=SPOLL(719)  
80 OUTPUT 719;"FREQ 2.123GHz;POW -1.23dBm"  
90 Wait4srq: A=SPOLL(719)  
100     IF A=0 THEN GOTO Wait4srq  
110 PRINT "HARDWARE IS SETTLED"  
120 END
```



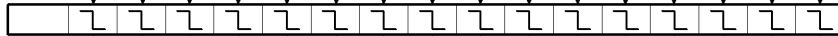
cg42a

**Figure 1-12. Status Registers**

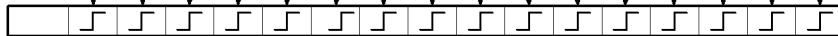
**STANDARD OPERATION STATUS GROUP**

DESCRIPTION	RESERVED FOR TMSL DEFINITION	UNUSED	UNUSED	UNUSED	SELF TEST IN PROGRESS	DIS-PLAY CHANGE	SELF TEST DATA READY	UNUSED ALWAYS (0)	UNUSED ALWAYS (0)	UNUSED ALWAYS (0)	UNUSED ALWAYS (0)	SWEEP	UNUSED ALWAYS (0)	SETTLING	CALIBRATE	
BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

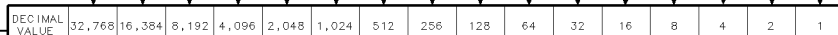
**CONDIT REGISTER**  
 STAT:OPER:COND?



**NEG TRANS FLTR**  
 STAT:OPER:NTR  
 STAT:OPER:NTR?



**POS TRANS FLTR**  
 STAT:OPER:PTR  
 STAT:OPER:P?



(A)

DECIMAL VALUE	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

**EVENT REGISTER**  
 STAT:OPER?

DECIMAL VALUE	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

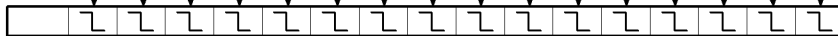
**ENABLE REGISTER**  
 STAT:OPER:ENAB  
 STAT:OPER:E

**NOTE:** STAT:PRES  
 THIS COMMAND PRESETS THE FOLLOWING ENABLE & TRANSITION REGISTERS: OPER. & QUES.

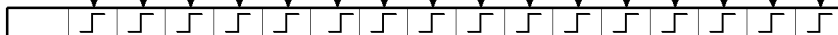
**DATA QUESTIONABLE STATUS GROUP**

DESCRIPTION	RESERVED FOR TMSL DEFINITION	UNUSED	UNUSED	UNUSED	HARDWARE ERROR (FAULT)	SELF TEST FAILED	* CALIBRATION ERROR	MODULATION ERROR (OVERMOD)	UNUSED ALWAYS (0)	FREQUENCY ERROR (UNLOCKED)	TEMP ERROR (OVEN COOL)	POWER ERROR (UN-LEVELLED)	UNUSED ALWAYS (0)	UNUSED ALWAYS (0)	UNUSED ALWAYS (0)	
BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

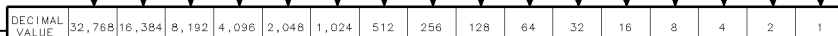
**CONDIT REGISTER**  
 STAT:QUES:COND?



**NEG TRANS FLTR**  
 STAT:QUES:NTR  
 STAT:QUES:NTR?



**POS TRANS FLTR**  
 STAT:QUES:PTR  
 STAT:QUES:P?



(B)

DECIMAL VALUE	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

**ENABLE REGISTER**  
 STAT:QUES:ENAB  
 STAT:QUES:ENAB?

DECIMAL VALUE	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

**EVENT REGISTER**  
 STAT:QUES:EVENTJ?

cg43a

This section discusses a trigger model used in SCPI instruments. Trigger system topics are explained in the following paragraphs:

<b>Generalized Trigger Model</b>	This paragraph explains the structure and components of the trigger model used in SCPI instruments.
<b>Trigger Command Definitions</b>	These paragraphs provide condensed definitions for the keywords used in this section.

---

## Generalized Trigger Model

An instrument trigger system synchronizes instrument actions with specified events. An instrument action may be making a measurement or sourcing an output signal. The events used to synchronize these actions include software trigger commands, changing signal levels, and pulses on BNC connectors. The trigger system also lets you specify the number of times to repeat certain actions, and delays between actions. Figure 1-13 is an example of a SCPI trigger model.

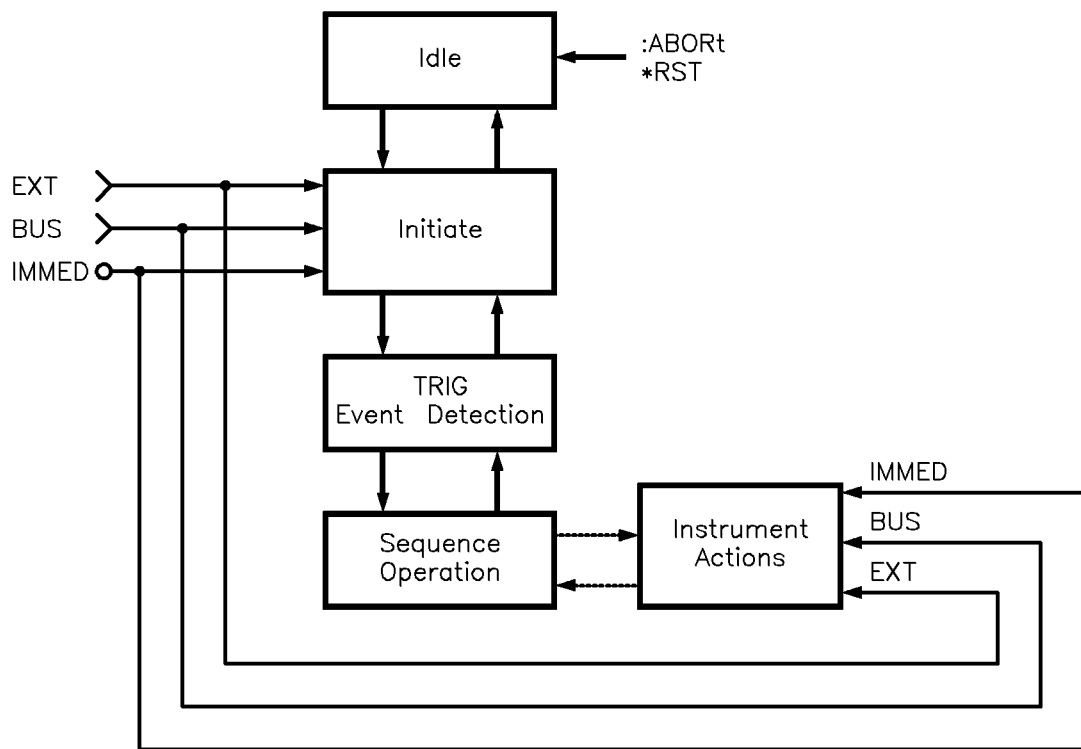
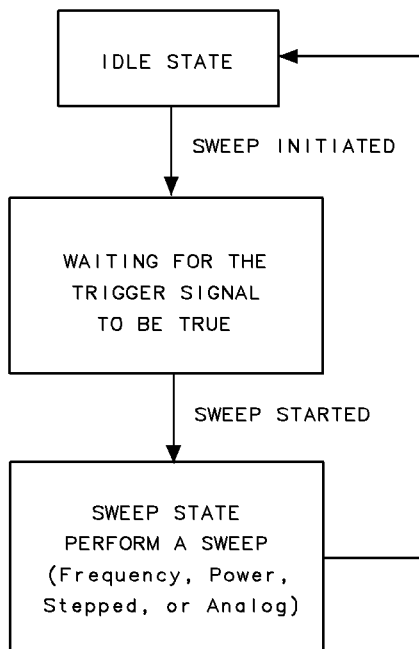


Figure 1-13. The TRIG Trigger Configuration

## Description of Triggering in Sweepers

The sweepers follow the SCPI model of triggering. It is a layered model with the structure shown in Figure 1-14.



Trigmod

**Figure 1-14. Simplified Trigger Model**

The process of sweeping involves all three of these states. The IDLE state is where the sweep begins. The IDLE state is left when the sweep is initiated. This can happen on a continuous basis (`INIT:CONT ON`) or on a demand basis (`INIT:CONT OFF`). The functions of continuous and single sweeps are handled by this command. When the `INIT:CONT ON` command is given, the sweep is continuously re-initiated. When in the `OFF` state, the sweep is initiated with the `INIT:IMMEDIATE` command.

Once initiated, the *wait for trigger* state is entered. Here, the trigger signal selected by the **TRIG:SOURce** command is examined until a TRUE condition is detected. These trigger signals are :

IMMediate	This signal is always TRUE.
EXTErnal	This is the external trigger input jack. A positive transition on this jack constitutes a TRUE signal.
BUS	This signal is the GPIB <get> (Group Execute Trigger) message or a *TRG command.

When a TRUE signal is found, the sweep is actually started. The act of producing the sweep in some cases involves the use of trigger signals. For example, the stepped sweeps have modes that allow triggering for point-to-point advancement through the sweep. These trigger signals are selected by individual **TRIG:SOURce** commands in the appropriate subsystems (for example, **SWEep:TRIGger:SOURce**). The definition of these signals in the sweeper cause the sweep to jump to the next point when the signal becomes TRUE, therefore the first point in the stepped sweep is produced immediately upon starting the sweep. Receiving a trigger signal at the last point causes the IDLE state to be re-entered. Analog sweeps do not use the trigger signals during the sweep (although the trigger signals are needed to start the sweep as described). The **ABORt** command resets any sweep in progress and immediately returns the instrument to the IDLE state. The **\*WAI**, **\*OPC** and **\*OPC?** commands indicate a complete operation at the end of the sweep upon re-entry into the IDLE state.

#### Advanced Trigger Configurations

Because the SCPI layered trigger model is expandable, many more complex trigger configurations are possible.

## Trigger Keyword Definitions

The following paragraphs contain condensed definitions of the keywords used in the command tables. Many of the commands in trigger related subsystems are *event commands*. Remember that event commands cannot be queried. Similarly, event commands have no related **\*RST** actions or settings. Event commands cause a particular action to take place inside the sweeper.

ABORt

The **ABORt** command forces the trigger system to the idle state. Any measurement or output sequence in process is aborted as quickly as possible. *ABORt* does not alter the settings programmed by other commands, unlike **\*RST**. *ABORt* is a root level event command and cannot be queried.

IMMediate

The **IMMediate** command provides a one-time override of the normal downward path in an event-detection state. The instrument must be in the specified event detection state when **IMMediate** is received, or an error is generated and the command has no effect. For example, the instrument must be in the TRIG state for **TRIGger:IMMediate** to work properly. If the instrument is in the idle state, the command has no effect, and an error would be generated. **IMMediate** is an event command and cannot be queried.

SOURce

The **SOURce** command selects the trigger source for an event-detection state. Only one source can be specified at a time, and all others are ignored. Sending **\*RST** sets **SOURce** to **IMMediate**. The most commonly used sources are:

- BUS**            The event detector is satisfied by either Group Execute Trigger (<GET>) or a **\*TRG** command. <GET> is a low level GPIB message that can be sent using the **TRIGGER** command in HTBasic.
- EXTernal**      An external signal connector is selected as the source.
- IMMediate**     Qualified events are generated automatically. There is no waiting for a qualified event.



# Related Documents

*IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.* The International Institute of Electrical and Electronics Engineers, New York, NY, 1987.

This standard defines the technical details required to design and build an GPIB interface (IEEE 488.1). This standard contains electrical specifications and information on protocol that is beyond the needs of most programmers. However, it can be useful to clarify formal definitions of certain terms used in related documents.

*IEEE Standard 488.2-1987, IEEE Standard Codes, Formats, Protocols, and Common Commands For Use with ANSI/IEEE Std 488.1-1987.* The International Institute of Electrical and Electronics Engineers, New York, NY, 1987.

This document describes the underlying message formats and data types used in SCPI. It is intended more for instrument firmware engineers than for instrument user/programmers. However, you may find it useful if you need to know the precise definition of certain message formats, data types, or common commands.

## **NOTE**

To obtain a copy of either of these documents, write to:

The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street New York, NY 10017  
USA

*BASIC 5.0/5.1 Interfacing Techniques. Vol. 2, Specific Interfaces.* Agilent Technologies Inc. 1987.

This HTBasic manual contains a good non-technical description of the GPIB (IEEE 488.1) interface in chapter 12, "The GPIB Interface." Subsequent revisions of HTBasic may use a slightly different title for this manual or

**Related Documents**

chapter. This manual is the best reference on instrument I/O for HTBasic programmers.

*Tutorial Description of the Agilent Technologies Inc. Interface Bus* Agilent Technologies Inc., 1987

This book provides a thorough overview of GPIB basics for the GPIB system designer, programmer, or user.

**NOTE**

To obtain a copy of either of these documents, contact the Agilent Technologies representative listed in your telephone directory.

---

## Programming Commands

# Programming Commands

This chapter contains information on all the programming commands used by the sweeper.

# Command Syntax

Following the heading for each programming command entry is a syntax statement showing the proper syntax for the command. An example syntax statement is shown below:

```
POWer [ :LEVel ] MAXimum | MINimum | UP | DOWN
```

Syntax statements read from left to right. In the above example, the “:LEVel” portion of the statement immediately follows the “POWer” portion of the statement with no separating space. A separating space is legal only between the command and its argument. In the above example, the portion following the “[ :LEVel ]” portion of the statement is the argument. Additional conventions used in the syntax statements are defined as follows:

- *italics* are used to symbolize a program code parameter or query response.
- ::= means “is defined as”.
- | (vertical bar) indicates a choice of one element from a list. For example, <A> | <B> indicates <A> or <B> but not both.
- [ ] (square brackets) indicate that the enclosed items are optional.
- Upper-Case Lettering (FREQuency) indicates that the upper-case portion of the command is the minimum required for the command.
- Lower-Case Lettering (FREQuency) indicates that the lower-case portion of the command is optional; it can either be included with the upper-case portion of the command or omitted.
- ? after a subsystem command indicates that the command syntax is a query.

## **NOTE**

SCPI is not case sensitive.

Common commands are generally not measurement related, but are used to manage macros, status registers, synchronization, and data storage. All common commands begin with an asterisk. The common commands are defined by IEEE 488.2

---

## \*CLS (Clear Status Command)

Clear the status byte, the Data Questionable Event Register, the Standard Event Status Register, the Standard Operation Status register and any other registers that are summarized in the Status Byte.

---

## \*DMC (Define Macro Command)

```
*DMC "<label>","<cmds>"
```

Allows the programmer to assign a sequence of elements to a macro label.

---

## \*EMC (Enable Macros Command)

\*EMC 1|0

Query Syntax

\*ESE?

Allows the programmer to query whether the macros are enabled. A return value of 0 indicates that the macros are disabled. A return value of 1 indicates that the macros are enabled.

---

## \*ESE (Standard Event Status Enable Command)

\*ESE <num>

Query Syntax

\*ESE?

Sets and queries the Standard Event Status Enable Register.

---

## \*ESR? (Standard Event Status Register Query)

Queries the value of the Standard Event Status Register. This is a destructive read.

### **\*GMC? (Get Macro Contents Query)**

**\*GMC? <label>**

Returns the current definition of the macro.

---

### **\*IDN? (Identification Query)**

Outputs an identifying string to the GPIB. The response for the sweeper will be "HEWLETT-PACKARD,83750A,2415A00123,REV A.01.00" where the actual model number, serial number and firmware revision will be substituted in.

---

### **\*LMC? (List Macro Query)**

Returns the currently defined macro labels.

---

### **\*LRN? (Learn Device Setup Query)**

This returns a huge string of device specific characters that, when sent back to the sweeper, will restore the instrument state. Note that this implies that the command to re-digest the string is included in the query response to \*LRN? Since we use SYST:SET <huge data block> as our command to re-enter the learn string, then the \*LRN? response begins with "SYST:SET .....".

---



---

## \*OPC (Operation Complete Command)

Operation complete command. The sweeper will set bit 0 in the Standard Event Status Register when all pending operations have finished.

In CW mode, this is defined as RF settled. In an analog swept mode, this is defined as the beginning of a new sweep. In step sweep mode, this is also defined as the beginning of a new sweep unless in point-to-point triggering; then it is defined as RF settled at each point. Many commands do not disturb the RF output energy, and thus, when executed alone, are effectively complete immediately.

### Query Syntax

**\*OPC?**

Operation Complete query. The sweeper will return an ASCII '1' when all pending operations have finished.

---

## \*OPT? (Option Identification Query)

Outputs a string identifying any device options. The response for the sweeper will be "1E1,1E5, . . . " with a list of actual options inserted.

## **\*PMC (Purge Macros Command)**

Deletes all macros that have been previously defined using the \*DMC command.

---

## **\*PSC (Power-On Status Clear Command)**

**\*PSC { 0 | 1 }**

This command controls the automatic power-on clearing of the Service Request Enable Register and the Standard Event Status Enable Register. Setting the power-on-clear flag TRUE causes the registers to be cleared at power-on, thus preventing the device from requesting service. Sending a <Decimal Numeric Program Data> element that rounds to the integer value 0 makes the flag FALSE and thereby potentially allows the device to request service at power-on. Sending any value other than 0, in the range -32767 to 32767 sets the flag TRUE.

Example

**\*PSC 0;\*SRE 32;\*ESE 128**

This sequence allows a device to request service at power-on.

Query Syntax

**\*PSC?**

The query reads the status of the power-on-clear flag. A value of 0 indicates that the flag is FALSE. A value of 1 indicates that the flag is TRUE.

---

## \*RCL (Recall Command)

\*RCL <num>

The instrument state is recalled from the specified memory register. Range is 1 through 9.

---

## \*RMC (Remove Macro Command)

\*RMC <label>

Deletes a single macro.

---

## \*RST (Reset Command)

The instrument is set to a pre-defined condition. These conditions are explained under each command.

---

## \*SAV (Save Command)

\*SAV <num>

The present instrument state is stored in the specified memory register. Range is 1 through 9.

---

### \*SRE (Service Request Enable Command)

\*SRE <num>

\*SRE?

Sets and queries the value of the Service Request Enable Register.

---

### \*STB? (Read Status Byte Query)

Queries the status byte. This is non-destructive.

---

### \*TRG (Trigger Command)

Performs the same function as the Group Execute Trigger command defined by IEEE 488.1.

---

### **\*TST? (Self-Test Query)**

A full self-test is performed , without data logging or looping, and returns one of the following error codes :

- 0 = Passed (no tests failed and at least one test passed)
- 1 = Failed (one or more tests failed)
- 2 = Skipped (all tests are skipped or can't do - doubtful )
- 3 = Can't Do (all tests are can't do - highly unlikely)
- 4 = Not Run

---

### **\*WAI (Wait-to-Continue Command)**

This causes the device to not execute any commands until the pending commands are completed. In our instrument, this will allow synchronous sweep operation by giving TSWep;\*WAI which will start a sweep and then wait until it completes. See \*OPC for a more detailed description.

# Subsystem Commands

Subsystem commands include all measurement functions and some general purpose functions. Subsystem commands are distinguished by the colon used between keywords, as in **POWER:SLOPe**. Each subsystem is a set of commands that roughly corresponds to a functional block inside the instrument.

---

## ABORt

This causes the sweep in progress to abort and reset. If the INIT:CONT is ON it will immediately re-initiate a new sweep.

---

## AM:STATe

**AM:STATe ON|OFF|1|0**

**AM:STATe?**

This sets and queries the status of the AM modulation.

\*RST setting is OFF.

---

## AM:SOURce

AM:SOURce { EXTernal }

Query Syntax

AM:SOURce?

This sets and queries the AM modulation source. This is always EXTernal.

## CALibration:PEAKing

```
CALibration:PEAKing[:EXECute]
```

Query Syntax

```
CALibration:PEAKing[:EXECute]? <dac_va>
```

Peaking is used to obtain the maximum available power and spectral purity; and the best pulse and FM envelopes at a given frequency. `CALibration:PEAKing[:EXECute]` causes a peak power calibration to occur for a CW frequency. In general `-1` is returned if there is a problem.

---

## CALibration:PMETER:FLATness:INITiate?

```
CALibration:PMETER:FLATness:INITiate? USER
```

Initiates the user flatness calibration. This calibration requires the use of an external power measurement. Once initiated, the sweeper will setup for the first point to be measured, and will respond to the query with the frequency at which the power is to be measured.

Refer to “Using the User Flatness Correction Commands, Example Program 8” in Chapter 1 for specific examples of this command.



---

## CALibration:PMETer:FLATness:NEXT?

`CALibration:PMETer:FLATness:NEXT? <num> [lvl suffix]`

The parameter is the measured power that is currently being produced by the sweeper. The user is to supply this parameter after measuring the power using his/her own power meter. The query response will be issued after the sweeper has processed the supplied parameter and has settled on the next point to be measured. The query response will be:

>0 ::= the frequency [in Hz] that is now being produced.

0 ::= this means that the calibration is complete.

<0 ::= an error has occurred and the calibration is to be aborted.

Refer to “Using the User Flatness Correction Commands, Example Program 8” in Chapter 1 for specific examples of this command.

---

## CALibration:TRACk

`CALibration:TRACk`

Causes an automatic tracking calibration procedure to be performed. The instrument initiates a peaking algorithm which automatically aligns the YIG tracking filter at a series of frequencies over its entire range, to optimize RF output power. This procedure is also called Autotracking. Pressing the **PRESET** key will abort the autotracking procedure.

## **CALibration:SECurity:CODE**

**CALibration:SECurity:CODE <OldPasswd> <NewPasswd>**

Changes the current password to a new one. The password must be five numerical digits and may not start with zero (0). Alphabetic and special characters are not allowed. The command is sent once; you do not verify it by sending the command a second time.

The old password is used for verification only. If the old password is incorrect, an error message will show up in the SCPI message queue and the new password will be rejected.

---

## **CALibration:SECurity:PASSword**

**CALibration:SECurity:PASSword <passwd>**

This command is used to supply the current password which then allows programmer to make changes in the password-protected areas of calibration constants and diagnostics.

---

## CORRection:FLATness:FREQ

```
CORRection:FLATness:FREQ {<num>[freq suffix],<num>[freq  
suffix]...}
```

### Query Syntax

```
CORRection:FLATness:FREQ?
```

Sets and queries an array of up to 801 frequency-correction elements. This correction information will be used to create a correction array that will be added to the internal calibration array. The array can be of arbitrary spacing. At every instantaneous frequency linear interpolation is used to determine an amplitude correction. If a value of START or STOP frequency is specified that is outside the limits of the specified frequencies, the correction applied for those points will be 0 dB.

At cold power-up, the array has two values: 0 Hz, and 20.5 Hz.

## CORRection:FLATness:AMPL

```
CORRection:FLATness:AMPL {<num> [DB], <num> [DB] . . . }
```

```
CORRection:FLATness:AMPL?
```

Sets and queries an array of up to 801 amplitude correction elements. This correction information will be used to create a correction array that will be added to the internal calibration array. This array is used in conjunction with CORR:FLAT:FREQ on a one to one basis.

At cold power-up, the array has two values: 0 dB and 0 dB.

Query Syntax

---

## CORRection:FLATness:POINTs?

```
CORRection:FLATness:POINTs? [MAXimum|MINimum]
```

Returns the number of frequency-power pairs entered using the CORR:FLAT:AMPL and CORR:FLAT:FREQ commands. If they differ in number, the smaller is used.

At cold power-up, the value is 2.

---

## CORRection[:STATe]

`CORRection[:STATe]ON|OFF|1|0`

Query Syntax

`CORRection[:STATe]?`

Sets and queries the switch on the users ALC correction system. This switch prevents the User Correction data from being added to the internal CALibration data.

\*RST value is OFF.

---

## CORRection:VOLTs:SCALE

`CORRection:VOLTs:SCALE`

Query Syntax

`CORRection:VOLTs:SCALE?`

Sets and queries the rear panel “V/GHZ” scaling factor.

---

## CORRection:VOLTs:OFFSet

`CORRection:VOLTs:OFFSet`

`CORRection:VOLTs:OFFSet?`

Sets and queries the rear panel “V/GHZ” offset factor.

Query Syntax

---

## DIAG:LRNS?

This returns a small string (that is, <1 kbytes) of device specific characters that, when sent back to the sweeper, will restore the instrument state with the exception of user ALC arrays. User Alc arrays will be zeroed out. Note that this implies that the command to re-digest the string is included in the query response to DIAG:LRNS? Since we use SYST:SET <huge data block> as our command to re-enter the learn string, then the DIAG:LRNS? response begins with "SYST:SET . . . . .".

---

## DIAGnostic:TEST:FULLtest?

This query command will execute all available self-tests in diagnostic test subsystem. The instrument state is restored upon completion of self-test. This is an IEEE 488.2 requirement for \*TST which this is a synonym or an alias of. The result is one of the following four values:

- 0 = Passed (no tests failed and at least one test passed)
- 1 = Failed (one or more tests failed)
- 2 = Skipped (all tests are skipped or can't do - doubtful )
- 3 = Can't Do (all tests are can't do - highly unlikely)
- 4 = Not Run

## DIAGnostic:TEST:FULLtest:REPort?

This query command will return the status of the fulltest or failure data on the failed test most likely to have caused additional failures.

Status = NOTRUN|PASSED

or

Fail Data Format =

<name> <status> <minValue> <actualData> <maxValue>

### **NOTE**

If any individual test is not run then **NOTRUN** will be returned regardless of other failures.



## DISPlay[:STATe]

DISPlay[:STATe]ON|OFF|1|0

Query Syntax

DISPlay[:STATe]?

Sets and queries the display ON/OFF switch. Once State is set OFF, only an \*RST or SYST:PRES can set it back to ON.

\*RST value is 1.

# FM Subsystem

---

## FM:COUPling

FM:COUPling AC|DC

Query Syntax

FM:COUPling?

Sets and queries the FM input coupling mode.

\*RST value is set to DC.

---

## FM:STATe

FM:STATe ON|OFF|1|0

Query Syntax

FM:STATe?

Sets and queries the FM state. This command will turn frequency modulation on or off.

\*RST value is set to OFF.

---

---

## FM:SENSitivity

FM:SENSitivity <-20|-6> <MHz/V>

Query Syntax

FM:SENSitivity?

Sets and queries the FM Sensitivity. This allows only two different settings: either  $-20$  MHz/V or  $-6$  MHz/V. The unit will return in term of Hz/V which also is the SPCI default unit.

\*RST value is set to  $-20$  MHz/V.

---

## FM:SOURce

FM:SOURce { EXTernal }

Query Syntax

FM:SOURce?

Queries only the FM Source. This is always external in the sweeper, but a SPCI command is provided for better compatibility.

---

## FREQuency:CENTer

```
FREQuency:CENTer <num> [freq suffix]
| MAXimum | MINimum | UP | DOWN
```

```
FREQuency:CENTer? [MAXimum | MINimum]
```

Sets and queries the center frequency.

\*RST value is  $(MAX + MIN)/2$ .

Any two frequency setting headers (START, STOP, CENTER, or SPAN) may be sent in a single message and the resulting sweep will be what was requested. The order in the message will not make any difference in the final result. When a message has been completed, coupling equations will be used to fix the unset parameters to the correct values. These equations specify that :

$$CENTer = (START + STOP) / 2$$

$$SPAN = (STOP - START)$$

If more than two are sent, then the last two in the message will be used to determine the sweep and no errors will be given.

If only one header is sent in a message, then the assumed pairs will be CENTER/SPAN and START/STOP. In other words, if only CENTER is sent, then SPAN will be kept constant (if possible) while adjusting CENTER to the requested value. The START/STOP frequencies would then be updated to reflect the changes based on the coupling equations.

It is OK for the sweeper to use “bumping” to move unspecified frequency parameters, but if the final value of any of the frequency headers is the result of bumping, then an error should be generated since the user is NOT getting what was specified. This means that to guarantee sequence independence requires sending the frequency pairs in a single message.

- Example 1            Present state: START=5 GHZ STOP=6 GHZ)  
                       "FREQ:START 10 GHZ"            *an error results since stop was bumped*  
                       "FREQ:STOP 12 GHZ"            *the final sweep is OK though (10 to 12)*
- Example 2            Present state: START=5 GHZ STOP=6 GHZ)  
                       "FREQ:STOP 12 GHZ"            *NO error generated, START unchanged*  
                       "FREQ:START 10 GHZ"            *still no error*
- Example 3            Present state: START=5 GHZ STOP=6 GHZ)  
                       "FREQ:START 10 GHZ;STOP 12 GHZ"            *both are fine, no errors*  
                       "FREQ:STOP 12 GHZ;START 10 GHZ"

## FREQuency[:CW]:FIXed]

```
FREQuency [ :CW | :FIXed ] <num> [ freq suffix ]  
| MAXimum | MINimum | UP | DOWN
```

```
FREQuency [ :CW ] ? [ MAXimum | MINimum ]
```

```
FREQuency [ :FIXed ] ? [ MAXimum | MINimum ]
```

Sets and queries the CW frequency. This does not change the swept/cw mode switch.

\*RST value is  $(MAX + MIN)/2$  . See `FREQ:CENTER` for more information.

Query Syntax

---

## FREQuency[:CW]:AUTO and FREQuency[:FIXed]:AUTO

```
FREQuency [ :CW ] : AUTO ON | OFF | 1 | 0
```

```
FREQuency [ :FIXed ] : AUTO ON | OFF | 1 | 0
```

```
FREQuency [ :CW ] : AUTO ?
```

```
FREQuency [ :FIXed ] : AUTO ?
```

Couples the CW frequency to the center frequency. Explicitly setting a value for `FREQ:CW` set `FREQ:CW:AUTO` to OFF.

\*RST setting is ON.

Query Syntax

---

## FREQuency:MANual

```
FREQuency:MANual <num> [freq suffix]  
|MAXimum|MINimum|UP|DOWN
```

### Query Syntax

```
FREQuency:MANual? [MAXimum|MINimum]
```

Sets and queries the MANual frequency. This controls the output frequency in swept manual mode (FREQ:MODE SWEEP and SWEEP:MODE MANUAL). The limits are START and STOP.

\*RST value is the same as FREQ:CENTER. See FREQ:CENTER for more information.

See also: SWEEP:MANual[:RELative]

## FREQuency:MODE

FREQuency:MODE FIXed|CW|SWEep|SWCW

FREQuency:MODE?

Sets and queries the switch that selects either swept, CW or swept CW operation.

CW	The output frequency is controlled by <code>FREQ:CW</code> .
SWEep	The output frequency is controlled by the <code>START,STOP,CENTER,SPAN,MAN</code> commands (and the <code>SWE:</code> subsystem).
SWCW	Sweep generator is active, but only the value of <code>FREQ:CW</code> is used. Frequency is constant.

`FIXed` is an alias for `CW`.

\*RST value is `CW`.



---

## FREQuency:MULTiplier

`FREQuency:MULTiplier <num> | MAXimum | MINimum`

Query Syntax

`FREQuency:MULTiplier? [ MAXimum | MINimum ]`

Sets and queries the frequency multiplier. <num> will be rounded to the nearest integer. This function changes mapping of frequency parameters on input to and output from the sweeper. Changing this does not affect the output frequency of the instrument, only the displayed parameters and query responses. The equation implied by this is :

$\text{ENTERED|DISPLAYED FREQ} = (\text{Hardware Freq} * \text{Multiplier}) + \text{Offset}$

$\text{DISPLAYED FREQ} = (\text{Hardware Freq} * \text{Multiplier}) + \text{Offset}$

\*RST value is 1.

---

## FREQuency:MULTiplier:STATe

`FREQuency:MULTiplier:STATe ON|OFF|1|0`

Query Syntax

`FREQuency:MULTiplier:STATe?`

This command queries and turns the frequency multiplier off and on.

\*RST setting is OFF.

## FREQuency:OFFSet

FREQuency:OFFSet <num>|MAXimum|MINimum

Query Syntax

FREQuency:OFFSet? [MAXimum|MINimum]

Sets and queries the frequency offset. This function changes mapping of frequency parameters on input to and output from the sweeper. Changing this does not affect the output frequency of the instrument, only the displayed parameters and query responses. The equation implied by this is :

ENTERED|DISPLAYED FREQ = (Hardware Freq \* Multiplier) + Offset  
\*RST value is 0.

---

## FREQuency:OFFSet:STATe

FREQuency:OFFSet:STATe ON|OFF|1|0

Query Syntax

FREQuency:OFFSet:STATe?

This command queries and turns the frequency offset off and on.

\*RST setting is OFF.

---

## FREQuency:SPAN

FREQuency:SPAN <num> [freq suffix] | MAXimum | MINimum | UP | DOWN

Query Syntax

FREQuency:SPAN? [MAXimum | MINimum]

Sets and queries the frequency span. See `FREQ:CENTER` for more information.

---

## FREQuency:START

FREQuency:START <num> [freq suffix] | MAXimum | MINimum | UP | DOWN

Query Syntax

FREQuency:START? [MAXimum | MINimum]

Sets and queries the START Frequency. See `FREQ:CENTER` for more information.

\*RST setting is MIN.

## FREQuency:STEP[:INCRement]

```
FREQuency:STEP [:INCRement] <num> [freq suffix]  
|MAXimum|MINimum
```

Query Syntax

```
FREQuency:STEP [:INCRement] ?
```

This sets and queries the frequency step size to be used for any node in the  
FREQ: tree that allows UP and DOWN as parameters.

\*RST setting is automatically calculated from SPAN.

---

## FREQuency:STOP

```
FREQuency:STOP <num> [freq suffix] |MAXimum|MINimum|UP|DOWN
```

Query Syntax

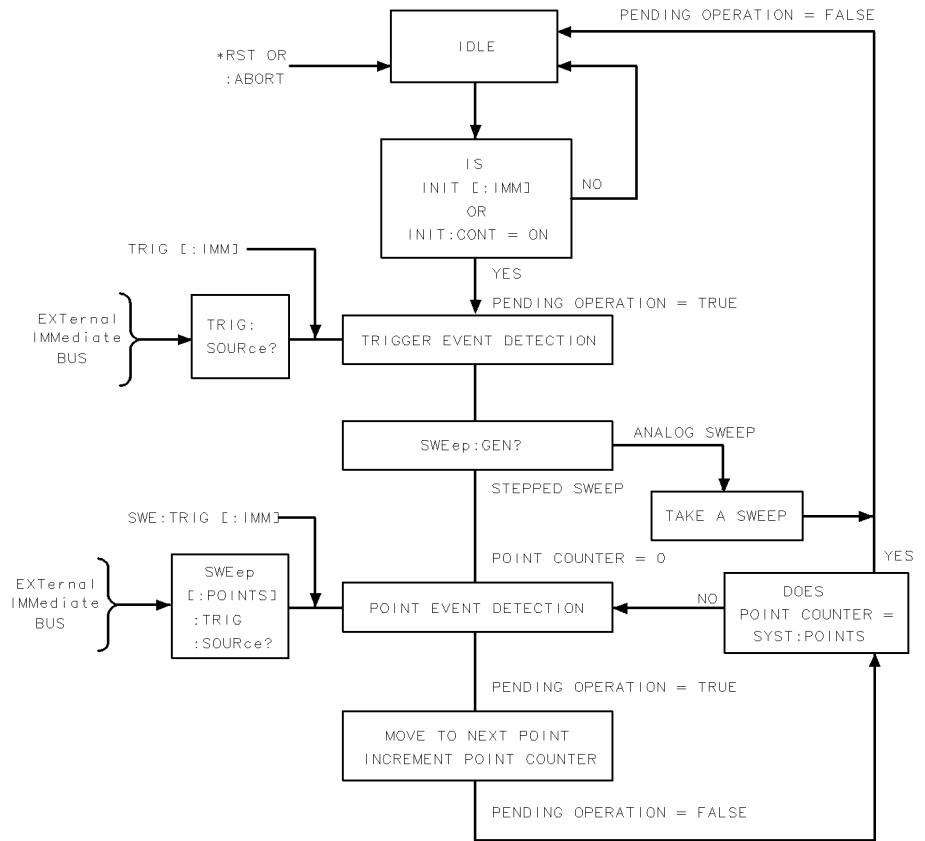
```
FREQuency:STOP? [MAXimum|MINimum]
```

Sets and queries the STOP Frequency. See FREQ:CENTER for more  
information.

\*RST setting is MAX.

---

# Triggering in the Sweeper



cg44a

Figure 2-1. Instrument Trigger Model

**Triggering in the Sweeper**

The process of sweeping involves all three of these states. The IDLE state is where it all begins. The IDLE state is left when the sweep becomes initiated. This can happen on a continuous basis (INIT:CONT ON) or on demand (INIT:CONT OFF). The functions of continuous and single sweeps are handled by this command. When INIT:CONT ON, the sweep is continuously re-initiated. When OFF, the sweep will be initiated with the INIT:IMMediate command.

Once initiated, the “Wait for Trigger” state is entered. Here, the trigger signal selected by the TRIG:SOURce switch is examined until a TRUE condition is detected. These trigger signals are :

IMMediate ::	this signal is always TRUE
EXTernal ::	this is the external trigger input connector. A positive transition on this connector constitutes a TRUE signal.
BUS ::	this signal is the GPIB <get> (Group Execute Trigger) message or a *TRG command.
HOLD ::	this signal is never TRUE

The command TRIGger:IMMediate forces a TRUE signal regardless of the SOURCE position. When a TRUE condition is found, the sweep is actually started.

The act of producing the sweep in some cases involves the use of trigger signals. For example, the stepped sweep has a mode that allows triggering of the point-to-point advancement through the sweep. These trigger signals are selected by individual TRIG:SOURce switches in the appropriate subsystems (for example, SWEEp:TRIGger:SOURce). The definition of these signals in the sweeper is to cause the sweep to jump to the next point when the signal becomes TRUE, therefore the first point in the stepped sweep will be produced immediately upon starting the sweep. Receiving a trigger signal at the last point causes the IDLE state to be re-entered. Analog sweeps do not use the trigger signals during the sweep (although they need them to start the sweep as described above)

The ABORt command resets any sweep in progress and immediately returns the instrument to the IDLE state.

The \*WAI, \*OPC and \*OPC? commands indicate a complete operation at the end of the sweep upon re-entry into the IDLE state.

---

## INITiate:CONTInuous

INITiate:CONTInuous ON|OFF|1|0

### Query Syntax

INITiate:CONTInuous?

Sets and queries the state of the CONTINUOUS initiation switch. This is more commonly known as SINGLE or CONTINUOUS sweep, but this is how all triggered SCPI instruments will be initiated. This does not affect a sweep in progress.

ON 1	The sweep will be re-started at the end of sweep automatically.
OFF 0	The sweep will wait until an INIT[:IMMEDIATE] is sent to re-initiate a sweep.

\*RST setting is OFF.

---

## INITiate[:IMMEDIATE]

Causes the initiation of a sweep. Useful mainly in the INIT:CONT OFF mode of operation (single sweep). By combining the \*OPC, \*WAI facilities with the INIT:IMM command, the functionality of the old "TAKE SWEEP" command and the "SINGLE SWEEP" command can be achieved.

# Marker Subsystem

## Usage of the <n> in MARKER headers

A single digit may be appended to any of the MARKER headers, as shown in the commands. This specifies which of the 10 markers (0 to 9) is being altered. In some cases, the function is global to all 10 markers and is not really something that is an attribute of a single marker, such as amplitude markers. Either markers are all amplitude markers or none of them are. In these cases, the <n> in the header is accepted as a convenience to the programmer and ignored.

---

## MARKer[n]:AMPLitude

```
MARKer[n]:AMPLitude ON|OFF|1|0
```

```
MARKer[n]:AMPLitude?
```

Sets and queries the amplitude marker ON/OFF switch. While [n] may be used, there is really only a single switch for all the markers.

When MARKer[n]:AMPLitude[:STATE] ON the front panel display message is **Marker=Amplitude**.

When MARKer[n]:AMPLitude[:STATE] OFF the front panel display message is **Marker=Intensity**.

\*RST value is OFF or Marker=Intensity.



---

## MARKer[n]:AOFF

This turns all the markers to OFF at once. While [n] may be used, there is really only a single switch to turn all the markers off.

This also turns marker delta mode to OFF if it was ON.

This also turns M1->M2 SWEEP mode to OFF if it was ON.

---

## MARKer[n]:FREQuency

MARKer [n]:FREQuency <num> [freq suffix] |MAXimum|MINimum

MARKer [n]:FREQuency? [MAXimum|MINimum]

Sets and queries the specified marker frequency (marker number one is the default if [n] is not specified). The value is interpreted differently based on the value of MARKer[n]:MODE.

MARKer[n]:MODE	How the frequency of the marker is determined.
FREQuency	Absolute frequency is used. The limits are confined to the present START and STOP frequency limits. For Front panel, if a value is set outside current START and STOP limits, it will confine to whichever is the closer frequency (either start or stop frequency with respect to the new entered value.) For SPCI command, the value is granted as user request.
DELTA	Absolute frequency is used for setting the value of the specified Marker. However, querying the specified Marker value will result with respect to DELTA Reference Marker. For example, if DELTA Reference Marker is set to 1 GHz, the specified Marker is set to 5 GHz. Querying the specified Marker value will return 4 Ghz. The limits are also confined to the present START and STOP frequency limits.

Reference Marker can be set using

MARKer[n]:REFerence <n>.

\*RST values are the same as the FREQ:CENT \*RST value.

---

## MARKer[n]:MODE

MARKer [n] :MODE FREQUency|DELTA

### Query Syntax

MARKer [n] :MODE?

This sets and queries the mode of the specified marker. While [n] may be used, there is really only a single switch to set all the markers to either FREQUency mode or to DELTA mode.

Setting one marker to DELTA turns all other MARKer[n]:MODEs to DELTA and the same for setting one marker to FREQUency mode.

Querying one marker mode showing the mode of all markers. If [n] is not specified, the default is one.

\*RST value is FREQUency.

## MARKer[n]:REFerence

**MARKer**[n]:REFerence <n>

**MARKer**[n]:REFerence?

This sets and queries which marker is the reference marker for use in the DELTA mode. While [n] may be used, there is really only a single reference for all the markers.

**MARKer1:REFerence 5;** and

**MARKer2:REFerence 5;** both set marker 5 as the reference.

**MARKer:REFerence <n>** is turned on and cannot be turned off if the marker's current mode is DELTA mode.

\*RST value for marker reference is 1

---

## MARKer[n][:STATe]

MARKer [n] [:STATe] ON|OFF|1|0

### Query Syntax

MARKer [n] [:STATe]?

This sets and queries the state of the specified marker. Marker number one is the defaulted if [n] is not specified.

The specified Marker cannot be turned off if it is the DELTA Reference Marker and Marker DELTA Mode is on.

If Marker M1 or M2 is requested to be off while the instrument is in M1->M2 SWEEP mode, then M1, M2, and M1->M2 SWEEP mode will all be turned off. The instrument will reside in the condition of M1=START and M2=STOP, or in CW mode. The previous condition of the instrument prior to entering M1->M2 SWEEP mode can not be recovered.

\*RST value for all markers if OFF.

## MEMory:RAM:INITialize

`MEMory:RAM:INITialize[:ALL]`

This command clears and initializes the entire content of RAM to all zeros. This clears all of the save/recall registers. The number of times that memory is cleared and the RAMs are set to zeros is set by `SYSTEM:SECurity:COUNT <num>`. After the RAMs are cleared, the instrument is set to preset conditions.

---

## OUTPut:STATe

OUTPut:STATe ON|OFF|1|0

Query Syntax

OUTPut:STATe

Sets and queries the output state, also known as RF ON/OFF

\*RST value is OFF.

---

## OUTPut:IMPedance?

Queries the output impedance: nominally 50 ohms. This is never set able, but only configurable from the calibration constants.

# Power Subsystem

Any place where dBm is accepted as a suffix, any level suffix will also be accepted. In the absence of a suffix, the units will be assumed to be determined by the setting of UNIT:POW.

---

## POWer:ALC:CFACTOR

```
POWer:ALC:CFACTOR <num> [DB] | MAXimum | MINimum | UP | DOWN
```

Query Syntax

```
POWer:ALC:CFACTOR? [MINimum | MAXimum]
```

Sets and queries the coupling factor to be used when POWer:ALC[:SOURce] is set to DIODE or PMETER.

---

## POWer:ALC:SOURce

```
POWer:ALC:SOURce INTernal | DIODE | PMETER | MMHead
```

Query Syntax

```
POWer:ALC:SOURce?
```

Sets and queries the ALC leveling source selection switch.

\*RST value is INTernal.



---

## POWer:ALC[:STATe]

```
POWer:ALC[:STATe] ON|OFF|1|0
```

### Query Syntax

```
POWer:ALC[:STATe]?
```

Sets and queries the state switch of the ALC. The positions are :

ON	normal ALC operation
OFF	open loop ALC mode

When ON, the POWER can be programmed in fundamental units as selected by the UNIT:POWer command.

When OFF, the POWER is no longer calibrated in absolute units and is set in units of dB of arbitrary modulator setting.

---

## POWer:ATTenuation

```
POWer:ATTenuation <num> [DB] | MAXimum | MINimum | UP | DOWN
```

### Query Syntax

```
POWer:ATTenuation? [MAXimum | MINimum]
```

Sets and queries the output ATTenuation level. Note that when setting the attenuator level to 10 dB the output power will be decreased by 10 dB. Programming a specified attenuation sets POWer:ATTenuation:AUTO OFF.

---

## POWER:ATTenuation:AUTO

POWER:ATTenuation:AUTO ON|OFF|1|0

POWER:ATTenuation:AUTO?

Sets and queries the state of the RF attenuator coupling switch. Programming a specified attenuation sets POWER:ATTenuation:AUTO OFF.

- ON       insures that the amplitude level of the ALC is kept within optimal limits.
- OFF       the attenuator setting is set to the value of POW:ATT and left there.

\*RST value is ON.

---

## POWer:CENTer

```
POWer:CENTer <num> [lvl suffix] | MAXimum | MINimum | UP | DOWN
```

### Query Syntax

```
POWer:CENTer? [MAXimum | MINimum]
```

Sets and queries the center power for power sweep. Default units (and units for query response) are determined by UNIT:POWER.

The coupling equations for power sweep are exactly analogous to those for FREQ sweep. Power sweep is allowed to be negative though, unlike Freq sweeps. See FREQ:CENt for a description.

See POWer:LEVel for an explanation of MAX|MIN.

\*RST value is 0 dBm.

---

## POWer[:LEVel]

POWer[:LEVel]<num>[lvl suffix]|MAXimum|MINimum|UP|DOWN

POWer[:LEVel]? [MAXimum|MINimum]

Sets and queries the output level. Default units (and units for query response) are determined by UNIT:POWer. MAXimum and MINimum levels refer to the leveling mode at the time the command is sent. For instance:

\*RST;POWer:LEVel MIN; ALC:SOURce MMHead

will have different effects than

\*RST;POWer:ALC:SOURce MMHead; POWer:LEVel MIN

\*RST value is 0 dBm.

---

## POWer:MODE FIXed|SWEep

POWer:MODE?

Sets and queries the setting of the power sweep mode switch. If POW:MODE SWEep then the output level is controlled by the START,STOP,CENTER and SPAN functions. If POW:MODE is FIX then the output is controlled by POW[:LEVEL].

\*RST value is FIXed.

---

## POWer:OFFSet

```
POWer:OFFSet <num> [DB] | MAXimum | MINimum | UP | DOWN
```

Query Syntax

```
POWer:OFFSet? [MAXimum | MINimum]
```

Sets and queries the power offset. This function changes mapping of absolute power parameters on input to and output from the sweeper. Changing this does not affect the output power of the instrument, only the displayed parameters and query responses. The equation implied by this is :

ENTERED|DISPLAYED POWER = Hardware Power + Offset

\*RST value is 0.

---

## POWer:OFFSet:STATe

```
POWer:OFFSet:STATe ON | OFF | 1 | 0
```

Query Syntax

```
POWer:OFFSet:STATe?
```

This command queries and turns the power offset off and on.

\*RST setting is OFF.

---

## POWer:SLOPe

POWer:SLOPe <num> [DB/freq suffix] | MAXimum | MINimum | UP | DOWN

POWer:SLOPe? [MAXimum | MINimum]

Sets and queries the RF slope setting (dB per Hz).

### FREQ:MODE

CW Rotates around 0 Hz.

SWEep or STEP Rotates around the start frequency.

\*RST value is 0.

---

## POWer:SLOPe:STATe

POWer:SLOPe:STATe ON | OFF | 1 | 0

POWer:SLOPe:STATe?

Sets and queries the power slope state.

\*RST value is 0.

---

## POWer:SPAN

POWer:SPAN <num> [DB] | MAXimum | MINimum | UP | DOWN

### Query Syntax

POWer:SPAN? [MAXimum | MINimum]

The coupling equations for power sweep are exactly analogous to those for FREQ sweep. Power sweep is allowed to be negative though, unlike frequency sweeps. See FREQ:CENT for a description. See POWer:LEVel for an explanation of MAX|MIN.

\*RST value is 0.

---

## POWer:STARt

POWer:STARt <num> [lvl suffix] | MAXimum | MINimum | UP | DOWN

### Query Syntax

POWer:STARt? [MAXimum | MINimum]

Default units (and units for query response) are determined by UNIT:POWer. The coupling equations for power sweep are exactly analogous to those for FREQ sweep. Power sweep is allowed to be negative though, unlike Freq sweeps. See FREQ:CENT for a description. See POWer:LEVel for an explanation of MAX|MIN.

\*RST value is 0 dBm.

---

## POWer:STATe

```
POWer:STATe ON|OFF|1|0
```

```
POWer:STATe?
```

Sets and queries the output power ON|OFF state.

\*RST value is OFF.

Query Syntax

---

## POWer:STEP[:INCRement]

```
POWer:STEP [:INCRement] <num> [DB] | MAXimum | MINimum
```

```
POWer:STEP [:INCRement]? [MAXimum|MINimum]
```

This command sets and queries the power step size to be used for any node in the POWER: tree that allows UP and DOWN as parameters.

\*RST setting is 1.0 dB.

Query Syntax



---

## POWer:STOP

```
POWer:STOP <num> [lvl suffix] | MAXimum | MINimum | UP | DOWN
```

### Query Syntax

```
POWer:STOP? [MAXimum | MINimum]
```

Set and queries the ending power for a power sweep. Default units (and units for query response) are determined by UNIT:POWer. The coupling equations for power sweep are exactly analogous to those for FREQ sweep. Power sweep is allowed to be negative though, unlike Freq sweeps. See FREQ:CENT for a description. See POWer:LEVel for an explanation of MAX|MIN.

\*RST value is 0 dBm.

Since FREQUENCY and PERIOD are inversely related, if both are sent in the same message, only the last one will be applied. If WIDTH and either FREQUENCY or PERIOD are sent in the same message, they must be accepted without error if the resulting pulse is possible.

---

## PULSe:PERIOD

PULSe:PERIOD <num> [time suffix] | MAXimum | MINimum

PULSe:PERIOD? [MAXimum | MINimum]

Sets and queries the period of the internal pulse generator. The resolution of this is 1  $\mu$ s.

\*RST value is 2  $\mu$ s.

Query Syntax

---

## PULSe:FREQuency

PULSe:FREQuency <num> [freq suffix] | MAXimum | MINimum

### Query Syntax

PULSe:FREQuency? [MAXimum | MINimum]

Sets and queries the frequency of the internal pulse generator. This is always the reciprocal of the period.

\*RST value is 500 KHz.

---

## PULSe:WIDTh

PULSe:WIDTh <num> [time suffix] | MAXimum | MINimum

### Query Syntax

PULSe:WIDTh? [MAXimum | MINimum]

Sets and queries the width of the internal pulse generator.

\*RST value is 1  $\mu$ s.

---

## PULM:SOURce

PULM:SOURce { INTernal | EXTernal | SCALar | SQ1K }

PULM:SOURce?

Sets and queries the source for the pulse modulation control signal.

INTernal internal pulse generator

EXTernal external pulse connector

SCALar 27.777 kHz square wave. This is used with scaler analyzers.

SQ1KHz 1.0 KHz square wave.

\*RST value is INTernal.

---

## PULM:STATe

PULM:STATe ON | OFF | 1 | 0

PULM:STATe?

Sets and queries the state of pulse modulation.

\*RST value is OFF.

---

## ROSCillator:SOURce

`ROSCillator:SOURce INTernal|EXTernal|NONE`

Query Syntax

`ROSCillator:SOURce?`

Sets and queries the reference oscillator selection switch. The command to set the switch will cause `ROSC:SOUR:AUTO OFF` to be done also.

\*RST value is automatically determined.

---

## ROSCillator:SOURce:AUTO

`ROSCillator:SOURce:AUTO ON|OFF|1|0`

Query Syntax

`ROSCillator:SOURce:AUTO?`

Set and queries the automatic reference selection switch.

\*RST value is 1.

## STATus:OPERation:CONDition?

Queries the Standard Operation Condition register.

---

## STATus:OPERation:ENABle

STATus:OPERation:ENABle <num>

STATus:OPERation:ENABle?

Sets and queries the Standard Operation enable register.

The STATus:PRESet value is 0.

\*RST does not affect this register.

---

## STATus:OPERation[:EVENT]?

Queries the Standard Operation Event Register. This is a destructive read.

---

---

## STATus:OPERation:NTRansition

STATus:OPERation:NTRansition <num>

Query Syntax

STATus:OPERation:NTRansition?

Sets and queries the Standard Operation Negative transition filter.

The STATus:PRESet value is 0.

\*RST has no effect.

---

## STATus:OPERation:PTRansition

STATus:OPERation:PTRansition <num>

Query Syntax

STATus:OPERation:PTRansition?

Sets and queries the Standard Operation positive transition filter.

After STATus:PRESet, all used bits are set to 1's.

\*RST has no effect.

---

## STATUS:PRESet

This command presets the following enable and transition registers:  
OPERation and QUEStionable.

ENABle            is set to all 0's.

NTRansition    is set to all 0's.

PTRansition    all bits that are used are set to 1's.

Unused bits remain 0's.

---

## STATus:QUEStionable:CONDition?

Queries the Data Questionable Condition Register.

---

## STATus:QUEStionable:ENABle

`STATus:QUEStionable:ENABle <num>`

`STATus:QUEStionable:ENABle?`

Sets and queries the Data Questionable SRQ ENABle register.

The STATus:PRESet value is 0.

\*RST has no effect.

Query Syntax



---

## STATUS:QUESTIONABLE[:EVENT]?

Queries the Data Questionable Event Register. This is a destructive read.

---

## STATUS:QUESTIONABLE:NTRANSITION

STATUS:QUESTIONABLE:NTRANSITION <num>

Query Syntax

STATUS:QUESTIONABLE:NTRANSITION?

Sets and queries the Negative TRANSITION filter for the Data Questionable Status register.

The STATUS:PRESET value is 0.

\*RST has no effect.

---

## STATus:QUEStionable:PTRansition

STATus:QUEStionable:PTRansition <num>

Query Syntax

STATus:QUEStionable:PTRansition?

Sets and queries the Positive TRansition filter for the Data Questionable Status register.

After STATus:PRESet, all used bits are set to 1's.

\*RST has no effect.

# Sweep Subsystem

**Table 2-1. Interactions between Dwell, Sweep Time, and Points.**

SWEep:xx:AUTO switches		Interaction
DWELI	TIME	
OFF	OFF	No coupling between SWEep:DWELI, SWEep:TIME and SWEep:POINts.
OFF	ON	No coupling between SWEep:DWELI, SWEep:TIME and SWEep:POINts. SWEep:TIME is always made minimum with the restriction from 10 ms lower limit, SWEep:TIME:LLIM feature and FREQ:SPAN to SWEep:TIME hardware relation.
ON	OFF	When SWEep:TIME or SWEep:POINts are changed, $SWEep:DWELI = \lfloor SWEep:TIME / SWEep:POINts \rfloor$ . SWEep:DWELI will be limited to 6 $\mu$ s, minimum.
ON	ON	Both conditions above apply.

---

## SWEep:CONTRol:TYPE

SWEep:CONTRol:TYPE MASTer|SLAVe

Query Syntax

SWEep:CONTRol:TYPE?

Sets and queries whether the sweeper is in master or slave mode. This applies in a dual source mode.

\*RST value is MASTer.

---

## SWEep:DWELL

SWEep:DWELL <num> [time suffix] | MAXimum | MINimum

Query Syntax

SWEep:DWELL? [MAXimum | MINimum]

Sets and queries the amount of time in seconds that the sweeper will dwell at each step after reporting a Source Settled SRQ and pulsing the “Trigger Out” line low. This one value will be used at each step when in the SWE:TRIG:SOUR IMM mode of a stepped sweep. Setting SWEep:DWELL sets SWEep:DWELL:AUTO OFF.

\*RST value is 1  $\mu$ s.

---

## SWEep:DWELL:AUTO

SWEep:DWELL:AUTO ON|OFF|1|0

Query Syntax

SWEep:DWELL:AUTO?

Sets and queries the state of the automatic DWELL calculation switch. Setting SWEep:DWELL sets SWEep:DWELL:AUTO OFF.

ON	See table above.
OFF	No coupling between SWEep:DWELL, SWEep:TIME and SWEep:POINTS.

\*RST state is ON.

---

## SWEep:POINTs

SWEep:POINTs <num>|MAXimum|MINimum

Query Syntax

SWEep:POINTs? [MAXimum|MINimum]

Sets and queries the number of points in a step sweep. When POINTs is changed, SWEep:STEP is modified by the equation  $STEP = SPAN / (POINTS - 1)$ . SPAN is normally an independent variable but will be changed to  $STEP * (POINTS - 1)$  if both of these parameters are changed in the same message.

\*RST value is 10.

---

## SWEep:POWer:STEP

SWEep:POWer:STEP <num> [lev suffix] | MAXimum | MINimum

SWEep:POWer:STEP? [MAXimum | MINimum]

Sets and queries the size of each power step. :STEP is governed by the equation:

$$\text{STEP} = \text{POWER SPAN}/(\text{POINTS}-1).$$

- If you change STEP size then the number of POINTS will be changed to:  
 $\text{POWER SPAN}/\text{STEP} + 1$   
and a Parameter Bumped execution error will be reported.
  - If POWER SPAN or POINTS are changed then:  
 $\text{POWER STEP} = \text{SPAN}/(\text{POINTS}-1)$
  - SPAN is normally an independent variable but will be changed to  
 $\text{STEP} * (\text{POINTS}-1)$   
if both of these parameters are changed in the same message.
- \*RST value is 0.

---

## SWEep[:FREQuency]:STEP

SWEep[:FREQuency]:STEP <num>[freq suffix] | MAXimum | MINimum

### Query Syntax

SWEep[:FREQuency]:STEP? [MAXimum | MINimum]

Sets and queries the size of each frequency step. :STEP is governed by the equation:

$$\text{STEP} = \text{SPAN}/(\text{POINTS}-1)$$

IF you change STEP size then the number of POINTS will be changed to SPAN/STEP+1 and a Parameter Bumped execution error will be reported. If SPAN or POINTS are changed then:

$$\text{STEP} = \text{SPAN}/(\text{POINTS}-1)$$

The above creates a coupling with SWEEPTIME also. If POINTS is changed through this coupling and DWELL:AUTO is ON and TIME:AUTO is ON then DWELL will be changed to SWEEPTIME/POINTS. SPAN is normally an independent variable but will be changed to STEP \* (POINTS-1) if both of these parameters are changed in the same message.

\*RST value is StopMax-StartMin/10.

---

## SWEEp:TIME

SWEEp:TIME <num> [time suffix] | MAXimum | MINimum

Query Syntax

SWEEp:TIME? [MAXimum | MINimum]

Sets and queries the current sweep time. Dwell can be coupled to SWEEPTIME if SWE:DWEL:AUTO is ON. DWELL is then governed by the equation:

$$DWELL = SWEEPTIME/POINTS$$

Changing either SWEEPTIME or POINTS will cause DWELL to be recalculated but will not cause an error. If you attempt to change DWELL then :AUTO will be set to OFF. If DWEL:AUTO is OFF then SWEEPTIME is independent of DWELL and POINTS.

SWEEp:TIME may also be restricted by SWEEp:TIME:LLIM and by the current setting of FREQ:SPAN. In analog ramp mode, SWEEp:TIME is always greater than or equal to FREQ:SPAN divided by calibration constant SWPMax (normally 400 MHz/ms). In addition, in step sweep mode, sweep time will be limited to always maintain a dwell time of 1  $\mu$ s minimum.

\*RST value is MIN.



---

## SWEep:TIME:AUTO

SWEep:TIME:AUTO ON|OFF|1|0

Query Syntax

SWEep:TIME:AUTO?

Sets and queries the automatic sweep time switch.

- |     |   |
|-----|---|
| ON  | The value of sweep time will be AUTOMATICALLY to SWEep:TIME? MIN      |
| OFF | Will remain a current setting unless bumped upward by other features. |

\*RST state is ON.

---

## SWEep:TIME:LLIMit

SWEep:TIME:LLIMit <num> [time suffix] |MAXimum|MINimum

Query Syntax

SWEep:TIME:LLIMit? [MAXimum|MINimum]

Sets and queries the lower sweep time limit. This value will specify the fastest sweep time that the user wants the sweeper to allow either on input or when calculated internally when in AUTO ON mode. This value must be greater than 10  $\mu$ s.

\*RST value is 10  $\mu$ s.

**Table 2-2. 83750 SCPI Sweep Mode Programming Table**

Mode			Sweep	Description of 83750 Sweep Condition
FREQ:	POW:	SWE:	:GEN	
CW	FIX	ignored	ignored	CW Non-swept
SWE	FIX	AUTO	ANALOG	Analog Freq Sweep
SWE	FIX	MAN	ANALOG	Manual Analog Freq Sweep
SWE	FIX	AUTO	STEP	Stepped Freq Sweep
SWE	FIX	MAN	STEP	Manual Step Freq Sweep
CW SWCW	SWE	AUTO	ANALOG	CW with Analog Power Sweep
CW SWCW	SWE	MAN	ANALOG	CW with Manual Analog Power Sweep
CW SWCW	SWE	AUTO	STEP	CW with Stepped Power Sweep
CWSWCW	SWE	MAN	STEP	CW with Manual Stepped Power Sweep
SWE	SWE	AUTO	ANALOG	Analog Freq + Power Sweep
SWE	SWE	MAN	ANALOG	Manual Analog Freq + Power Sweep
SWE	SWE	AUTO	STEP	Stepped Freq + Power Sweep
SWE	SWE	MAN	STEP	Manual Step Freq + Power Sweep
SWCW	FIX	AUTO	ANALOG	Analog Swept-CW sweep
SWCW	FIX	MAN	ANALOG	Manual analog Swept-CW sweep
SWCW	FIX	AUTO	STEP	Stepped Swept-CW sweep
SWCW	FIX	MAN	STEP	Manual stepped Swept-CW sweep

---

## SWEep:GENeration

SWEep:GENeration STEPped|ANALog

Query Syntax

SWEep:GENeration?

Sets and queries the type of sweep to be generated: an analog sweep or a digitally stepped sweep. In either case, all of the other sweep subsystem functions apply such as MANual, AUTO, INITiate:CONTInuous ON|OFF, etc.

\*RST is ANALog.

---

## SWEep:MODE

SWEep:MODE AUTO|MANual

Query Syntax

SWEep:MODE?

This selects and queries the manual sweep mode switch.

AUTO      the sweep is under the control of the INIT and SWEEP subsystems.

MANual    FREq:MANual, and SWEep:MANual[:RELative] control the output.

\*RST value is AUTO.

## **SWEep:MANual[:RELative]**

**SWEep:MANual [ :RELative ] <num>**

Query Syntax

**SWEep:MANual [ :RELative ]?**

Sets and queries a percent of sweep to go to and lock. This command will have no effect unless SWEep:MODE is set to MANual

\*RST value is 0.50.

---

## **SWEep:MANual:POINT**

**SWEep:MANual:POINT <num>**

Query Syntax

**SWEep:MANual:POINT?**

Sets and queries the position of manual sweep in terms of number of sweep:points. This command will have no effect unless SWEep:MODE is set to MANual.

\*RST value is 6.

---

## SWEep:MARKer:STATe

SWEep:MARKer:STATe ON|OFF|1|0

### Query Syntax

SWEep:MARKer:STATe?

Sets and queries the state of marker sweep. When this state is ON, the frequency sweep limits are taken to be from position of marker 1 to position of marker 2. If marker 1 was previously set to be greater than marker 2, their values will be permanently interchanged so that the instrument sweeps up in frequency.

Setting SWEep:MARKer:STATe to ON will turn marker 1 and marker 2 on.

\*RST value is 0.

---

## SWEep:MARKer:XFER

This transfers the values of marker 1 and marker 2 frequencies into `FREQ:START` and `FREQ:STOP`, respectively. If marker 1 was previously set to be greater than marker 2, their values will be interchanged so that the instrument sweeps up in frequency.

---

## SWEep[:POINTs]:TRIGger:SOURce

SWEep[:POINTs]:TRIGger:SOURce IMMEDIATE|BUS|EXTernal

SWEep[:POINTs]:TRIGger:SOURce?

Query Syntax

Sets and queries the stepped sweep point-to-point trigger source. This only applies when SWEep:GEN is set to STEPPed.

IMMEDIATE	Each new frequency point is stepped to automatically, after waiting the specified DWELL time.
BUS	Wait for a <GET> or *TRG over the GPIB before advancing to the next frequency in the sweep.
EXTernal	Wait for a signal to be received on the external connector.
HOLD	Do not proceed or wait for any trigger event

---

## SWEep:POINTs:TRIGger:

SWEep:POINTs:TRIGger:[IMMEDIATE]

Executes an immediate point to point event when in step sweep mode.

---

## SYSTEM:ALternate

SYSTEM:ALternate <num>|MAXimum|MINimum

Query Syntax

SYSTEM:ALternate? [MAXimum|MINimum]

Sets and queries the save/recall register number with which to alternate the foreground state of the instrument with.

\*RST value is 1.

---

## SYSTEM:ALternate:STATE

SYSTEM:ALternate:STATE ON|OFF|1|0

Query Syntax

SYSTEM:ALternate:STATE?

Sets and queries the state of the Alternate State function.

\*RST setting is OFF.

---

## SYSTem:COMMunicate:GPIB:ADDRess

SYSTem:COMMunicate:GPIB:ADDRess <n> MAX|MIN

This command changes the GPIB's (General Purpose Interface Bus) address.

Allowable values are 0 through 30.

---

## SYSTem:COMMunicate:PMETer:ADDRess

SYSTem:COMMunicate:PMETer:ADDRess <num>

SYSTem:COMMunicate:PMETer:ADDRess?

Sets and queries the GPIB address to be used for the power meter during sweeper calibration routines.

Allowable values are 0 through 30.

Query Syntax



---

## SYSTem:COMMunicate:PMETer:TYPE

```
SYSTem:COMMunicate:PMETer:TYPE { SCPI| 70100A| 437B| 438A }
```

Query Syntax

```
SYSTem:COMMunicate:PMETer:TYPE?
```

Sets and queries the mode type of power meter expected over the GPIB to be used for the power meter during sweeper calibration routines.

---

## SYSTem:ERRor?

Returns the next message in the error queue. The format of the response is :

```
<error number>,<error string>
```

where error number is defined in SCPI section 21.81.4 and error string is :

```
"<Generic SCPI string>;<More specific information>"
```

An example response to SYST:ERR? is

```
-123,"EXPONENT TOO LARGE"
```

## SYSTEM:KEY[:CODE]

SYSTEM:KEY[:CODE]<num>

Query Syntax

SYSTEM:KEY[:CODE]?

This accomplishes the equivalent of pressing a front panel key specified by the <num> code. The query form returns the key code of the last pressed key.

```
! Push front panel keys remotely
```

```
OUTPUT 719;"SYSTEM:KEY:CODE 23"
```

```
OUTPUT 719;"SYSTEM:KEY:CODE 47"
```

```
OUTPUT 719;"SYSTEM:KEY:CODE 56"
```

```
OUTPUT 719;"SYSTEM:KEY:CODE 49"
```

```
OUTPUT 719;"SYSTEM:KEY:CODE 34"
```

```
! Power Level 1 . 3 dBm
```

```
END
```

Refer to Table 2-3.

**Table 2-3. Sweeper Key Codes**

Key Name	Key Code	Key Name	Key Code
<b>Instrument State Keys</b>		(SINGLE/TRIG)	31
(SHIFT)	0	<b>Power Keys</b>	
(PRESET)	1	(POWER LEVEL)	23
(SAVE)	4	(POWER/SWEEP)	32
(RECALL)	5	(ALC MODE $\updownarrow$ )	102
(LOCAL)	8	(FLTNESS ON/OFF)	103
(MSG)	4		
<b>Marker Keys</b>		<b>Entry Keys</b>	
(MKR n)	2	( $\uparrow$ )	12
(OFF)	6	( $\downarrow$ )	38
(MKR $\Delta$ )	10	(STEP SIZE)	46
		( $\leftarrow$ )	54
<b>Modulation Keys</b>		(0)	55
(PULSE MODE $\updownarrow$ )	3	(1)	47
(AM MODE $\updownarrow$ )	7	(2)	48
(FM MODE $\updownarrow$ )	99	(3)	49
		(4)	39
<b>Frequency Keys</b>		(5)	40
(START)	20	(6)	41
(STOP)	21	(7)	13
(CF)	100	(8)	14
(SPAN)	25	(9)	33
(CW)	101	(.)	56
(M1 $\rightarrow$ M2/SWEEP)	30	(-)	57
		(GHz/dBm/db)	34
<b>Sweep Keys</b>		(MHz/ $\mu$ s)	42
(TIME)	22	(kHz/ms)	50
(TRIG MODE $\updownarrow$ )	26		

---

## SYSTEM:KEY:DISable

**SYSTEM:KEY:DISable SAVE**

The SAVE key grouping is disabled. This also disables the SAVE STATE feature (Save Lock).

**SYSTEM:KEY:DISable? SAVE**

Returns 1 if the save key is disabled, otherwise it returns 0.

---

## SYSTEM:KEY:ENABLE

**SYSTEM:KEY:ENABLE? SAVE**

Returns 0 if save key is disable, otherwise 1.

**SYSTEM:KEY:ENABLE SAVE**

This unlocks the SAVE registers.

\*RST value is for the SAVE registers to be enabled.

---

## SYSTEM:LANGUage

SYSTEM:LANGUage "SCPI" | "TMSL" | "COMP"

Query Syntax

SYSTEM:LANGUage?

This command causes the instrument to perform a language switch to another language system. TMSL is an alias for SCPI. For 8360 Series compatibility, the unquoted forms are also accepted, however, queries are always quoted.

---

## SYSTEM:PRESet[:EXECute]

This command sets the instrument state to either a factory or user defined state depending on the setting of SYSTEM:PRESet:TYPE. This is the same as pressing the front panel green **PRESET** key. There is no corresponding query.

---

## SYSTEM:PRESet:SAVE

This command saves the present state for use whenever the command SYSTEM:PRESet[:EXECute] is executed, or the front panel “green” key is pressed.

---

## SYSTem:PRESet:TYPE

SYSTem:PRESet:TYPE FACTory|USER

SYSTem:PRESet:TYPE?

This command sets and queries the type of preset to execute when the SYSTem:PRESet[:EXECute] command is given. FACTory preset defaults all values to factory specified values. USER defined preset defaults all values to a specified state of the instrument that the user has saved with SYSTem:PRESet:SAVE.

---

## SYSTem:SECurity:CLEar

SYSTem:SECurity:CLEar

This command clears and initializes the entire content of RAM to all zeros. This clears all of the save/recall registers. The number of times that memory is cleared and the RAMs are set to zeros is set by SYSTem:SECurity:COUNT <num>. After the RAMs are cleared, the instrument is set to preset conditions.

---

## SYSTEM:SECURITY:COUNT

SYSTEM:SECURITY:COUNT <num>|MIN|MAX

This sets the number of times that the RAMs will be cleared and initialized to zeros with the clear memory function. Values between and including 1 through 20 are accepted.

\*RST value is 1.

---

## SYSTEM:SECURITY:KLOCK

SYSTEM:SECURITY:KLOCK ON|OFF|0|1

This command locks the front panel keyboard against any entry except for the **PRESET** key and the line power switch.

\*RST value is OFF.

---

## SYSTEM:SECURITY:ZERO

SYSTEM:SECURITY:ZERO ON|OFF|1|0

This command replaces the frequency and markers displayed on the front panel with zeros. Each frequency is displayed as 0.00000000Hz. If markers have been set, they are also displayed as zeros. Annunciators, such as **SWEEP** and **CW**, are *not* blanked.

This function cannot be executed when the instrument is connected to an 8757 or when the instrument is speaking 8350 compatibility language.

\*RST value is OFF.

---

## SYSTEM:VERSion?

This query returns a formatted numeric value corresponding to the SCPI version number for which the instrument complies. The response shall have the form YYYY.V. The Ys represent the year version (for example, 1990) and the V represents an approved revision number for that year.

This is a query only and therefore does not have an associated \*RST state.



---

## TRIGger[:IMMediate]

This command causes the trigger event to occur regardless of other settings in the subsystem. This event does not affect any other settings in this subsystem.

This command has no effect unless the instrument is in the Wait for TRIG state. If the instrument is in the Wait for TRIG state, it performs its trigger action.

This is an event and has no \*RST condition.

---

## TRIGger:SOURce

TRIGger:SOURce IMMEDIATE|BUS|EXTERNAL|HOLD

TRIGger:SOURce?

The command sets and queries the source of the trigger event. The various settings have the following meanings.

IMMEDIATE	The trigger signal is always true.
BUS	The trigger source is the group execute trigger from GPIB. A trigger will occur when either a <GET>, or a *TRG command is received.
EXTERNAL	The trigger signal source is the external connector.
HOLD	Do not trigger on any event.

Query Syntax

---

## TSweep

This is a convenience command that does the equivalent of

```
"ABORt;INITiate[:IMMediate]"
```

By combing TSW with \*WAI, \*OPC and \*OPC?, the functionality of “single sweep” (S2) and “take sweep” (TS) can be achieved. To get something similar to the old TS command, use TSW;\*WAI to cause the parsing of commands to wait until the sweep is restarted and completed. (Notice it said “parsing”, not handshaking. The commands are still taken off of the bus since the bus is NOT held up like TS used to do).

For example :

```
POWER 5 DBM;TSW;*WAI;POWER 10 DBM
```

will set the power to 5 dBm, reset the sweep and start a new one, wait until the sweep completes, and then set the power to 10 dBm.

For example :

```
POWER 5 DBM;TSW;POWER 10 DBM
```

will set the power to 5 dBm, reset and start the sweep and then change the power to 10 dBm without waiting for the sweep to complete.

\*OPC? should be used to synchronize the sweep with other instruments. For example, you want to sweep the sweeper and then read a voltmeter once the sweep is finished. \*WAI will not help since the sweeper will let go of the GPIB immediately after the sweep begins. Here is how to do it :

- Set up the DVM.
- Set up the sweeper.
- Send to the sweeper;“TSW;\*OPC?”
- Enter from the sweeper;DONE

The program will wait here forever if necessary, until the sweeper completes the sweep and responds to the \*OPC? query with a 1.

- Now the sweep is finished, read the voltmeter.

---

## **8350B Compatibility Guide**

---

## Introduction

This chapter explains the remote operation of the **HP 83751A/B** and **HP 83752A/B** synthesized sweepers when used as a replacement for the **HP 8350B** sweep oscillator and **HP 83500** Series plug-ins. This is intended for use by those familiar with HP-IB programming and the basic functions of the **HP 8350B** sweep oscillator. For complete **programming** information refer to the **HP 8350B** Operating and Service manual.

---

## Data

The HP 83750 Series of synthesized sweepers also accepts HP-IB commands in the same language as used by the **HP 8350B** sweep oscillator and **HP 83500** Series plug-ins. This language is selected by the SCPI command “**SYSTEM: LANGUage COMPAtible,**” or from the front panel, SHIFT SPECIAL 15 ENTER. Use the arrow keys to select Remote **Lang=8350**. The programming data string consists of a string of ASCII coded characters composed of one or more the following control fields:

- Sweep Mode/Limits
- Frequency Markers
- Sweep Trigger
- Modulation
- Step Size
- Instrument State/Registers
- Power Level
- Power Control
- ALC Modes
- Special HP-IB Only Functions

---

## Input Syntax

The HP 83750 Series responds to program codes in the order in which they are received. Each function is programmed with a string of ASCII coded characters that follow one of the following sequences.

- **[Function Code]** [Numeric Value] **[Numeric terminator]**
- [Function Code]

The HP-IB program code sequence typically mirrors that of the local front panel keystroke sequence.

---

## Function Codes (Prefix Activate)

Function codes are typically 2 to 4 character mnemonics. For a function that has a numeric value associated with it, passing the function code only will enable and activate the function for further data entry.

---

## Numeric Value (Numeric Format)

These are either a single decimal digit, a set of characters or less representing a number, or a string of binary bytes. A string of characters can be expressed in exponential, decimal, or integer form. Acceptable numeric formats are referenced in later sections by the following format syntax:

Format #1: Exponential  $\pm d^{**}d.d^{***}dE\pm dd$

Format #2: Decimal  $\pm d^{***}d.d^{***}d$

Format #3: Integer  $\pm d^{***}d$

Format #4: Single Digit  $d$

Format #5: Double Digit  $dd$

Format #6: Binary String  $b^{***}b$

Format #7: Binary Byte  $b$

The character 'd' indicates a leading or trailing zero, a space, or a numeric digit (0 through 9). The character 'b' indicates an **8-bit** binary byte. The characters "\*\*\*\*" indicate a variable number of the previous character. Numeric values that are not binary in nature are scaled by the appropriate numeric terminator.

---

## Numeric Terminators

Numeric terminators are of 2 types, mnemonic and fundamental terminators. Mnemonic terminators are **2-character** codes that terminate and scale the associated numeric value. Thus, frequency values can be entered in **GHz** (GZ), MHz (MZ), **kHz** (**KZ**), or Hz (HZ); sweep time values can be entered in seconds (SC) or milliseconds (MS) and power values can be entered in **dB** or **dBm** (**DB** or DM). Fundamental terminators consist of the ASCII characters Line Feed or Next Line (LF or NL, decimal **10**), semicolon (;, decimal **59**), or comma (', decimal **44**), and may be used in lieu of a mnemonic terminator. However, when this is done the HP **8350B** assumes the numeric value is in the fundamental units of Hz, seconds, or **dB**, depending on the active function.

---

## Valid Characters

The alpha program codes can be either upper or lower case (they can be interchanged). Spaces, leading zeroes, and carriage returns (CR) are ignored. Characters containing a parity bit will have that bit cleared by the HP 83750 Series.

---

## Instrument Preset

Instrument Preset turns off all functions then sets the following:

- Sweep Mode: Start/Stop
  - Start= minimum specified frequency
  - Stop= maximum specified frequency
- Sweep Type: Timed, minimum sweep time
- Sweep Trigger: Internal
- Vernier/Offset: set to 0 MHz
- Markers: all values set to center of frequency span, all off
- Frequency Step Size: set to default value (100 MHz)
- Status Bytes: cleared
- Display Multiplier: set to 1
- Display Offset: set to 0 MHz
- Power Level: 0 **dBm**
- Power Sweep/Slope: set to 0 **dB**
- **RF: on**
- FM Sensitivity: -6 **MHz/V**
- Power Step Size: set to default value (1 **dB**)

Instrument Preset does not affect Storage Registers, HP-IB address, or Service Request Mask

---

## Output Data

The sweeper has several output modes that allow the user to learn and interrogate the present instrument state. The following output modes are available:

- **Learn** String
- Mode String
- Interrogate Function
- Active Function
- Status



The program codes and syntax to enable each function are shown in Table 1. The Learn String, Mode String, and Status functions send a Data message consisting of a string of **8-bit** binary bytes. These messages are terminated by asserting the EOI signal in parallel with the last byte of the message to be sent. The Interrogate and Active functions send a Data message consisting of an ASCII string representing the numeric value in exponential or decimal form terminated with a Line Feed (LF).

Binary Syntax: [**b\*\*\*b**] [EOI]

Numeric Syntax: [ $\pm$ d.dddddE $\pm$ dd] or d\*\*\*d.d\*\*\*d [LF] [EOI]

Where the character 'b' indicates an **8-bit** binary byte and 'd' indicates a decimal digit (0 through 9). Note that the binary output format could have bytes that may be misinterpreted as Line Feeds so the user should defeat the ASCII LF as a valid character string terminator and rely on the byte count.

---

## Learn String

Selected with the “**OL**” program code, the sweeper outputs a Learn String of less than 1024 bytes in length. This binary data string completely describes the present instrument state except for the user ALC flatness arrays. The information is packed and encoded for minimal storage requirements thereby making data analysis difficult. When stored in an ASCII character data string, the Learn String can later be input to the sweeper to restore that instrument state (See Table 1 for Input Learn String information). The length of the Learn String is **fixed** and independent of the functions selected.

Format: [**b\*\*\*b**] [EOI]

## Mode String

Selected with the "OM" program code, the sweeper outputs a Mode String of 8 bytes in length. This binary data string describes presently active functions. The information passed includes only the active functions with no numeric values included. Use the Active or Interrogate Function if numeric values are desired. The length of the Mode String is **fixed** independent of the functions selected and the Plug-in used.

Format: 8 [8 bit bytes] [EOI]

### 83750 Series MODE STRING DEFINITION

**NOTE:** In all bit number references mentioned below, bit 0 is the least significant bit and bit 7 is the most significant bit.

In bytes 1 and 2 the numeric value of the entire byte indicates function.

#### BYTE 1

Numeric Byte Value	Front Panel Key Codes
0-9	0-9 Numerical Keys
10	•
11	- Minus Key
12	Backspace
13	Step Up (up arrow)
14	Step Down (down arrow)
15	Marker to CF
16	Permanent Marker Sweep
17	Instrument Preset
66	[single]
67	[manual]
68	M1
69	M2
70	M3
71	M4
72	M5
76	Marker Sweep
77	Off
78	Marker delta
80	Local
81	Save

BYTE 1 (continued)

Numeric Byte Value	Front Panel Key Codes
82	Recall
83	Alt
84	FM
85	Pulse
86	AM
87	Entry Off
97	Start Freq
98	Stop Freq
99	CF
100	$\Delta F$
101	CW
102	Power Level
103	Sweep Time
104	Shift
105	GHz/dB(m)
106	MHz/ $\mu$ sec
107	kHz/msec
108	Hz/sec
112	Ampt/Mkr
113	Peak
114	Pwr Swp
115	Slope
116	RF
141	Shift Off
144	HP-IB Address
152	Step Down
153	Step Up
161	Freq Mult
162	Freq Offset

**BYTE 1 (continued)**

<b>Numeric Byte Value</b>	<b>Front Panel Key Codes</b>
181	[SHIFT] [0]
182	[SHIFT] [1]
183	[SHIFT] [2]
184	[SHIFT] [3]
185	[SHIFT] [4]
186	[SHIFT] [5]
187	[SHIFT] [6]
188	[SHIFT] [7]
189	[SHIFT] [8]
190	[SHIFT] [9]

**BYTE 2**

Numeric Byte Value	Current Active Function Code
1	Save
2	Recall
3	Alt
6	Power LVL Step Size
7	Power Level
8	Sweep Time
10	CW
11	CF
12	Delta frequency
13	Start
14	Stop
15	Marker 1
16	Marker 2
17	Marker 3
18	Marker 4
19	Marker 5
21	Frequency step size
22	Calibration constants accessed
23	HP-IB Address [Shift Local]
26	Manual sweep
27	Frequency Offset
28	Frequency Multiplier
29	RF Slope
31	Bucket Pulses
32	Number of steps
33	Power Sweep
35	ALC
36	Attenuator
43	Sweep Time Limit
60	Vernier
61	RF Offset

**BYTE 3**

Byte 3 is separated into 3 functional parts. Bits 0, 1, and 2 contain a number that represents the Active Marker. Bits 3, 4, and 5 contain a binary number that represents the Delta Reference Marker. Bits 6 and 7 are not used.

<b>Bits</b>	<b>Definition</b>
0-2	Active Marker Binary number corresponds to marker number)
3-5	Delta Reference Marker (Binary number corresponds to marker number)
6, 7	Not used

**BYTE 4**

Each of the 8 bits that make up byte 4 independently represents the status of the frequency Markers and Marker Modes. A logic one in any bit indicates active function.

<b>Bit</b>	<b>Definition</b>
0	Marker Sweep
1	Marker 1
2	Marker 2
3	Marker 3
4	Marker 4
5	Marker 5
6	Not used (always=0)
7	Marker Delta Mode

**BYTE 5**

Byte 5 is separated into 3 functional parts. Bits 0 and 1 contain a binary number that indicates the Sweep Trigger mode. Bits 2, 3, and 4 contain a binary number that indicates the Sweep Source. Bits 5, 6, and 7 contain a binary number that indicates Sweep Mode.

<b>Bits</b>	<b>Definition</b>
0-1	Sweep Trigger 0 Internal Free Run 2 External
2-4	Sweep Source 0 Continuous Analog Sweep ("Time") 1 Single Analog Sweep 2 Manual

BYTE 5 (continued)	
Bits	Definition
	4 Continuous Step Sweep 5 Single Step Sweep
BYTE 6	
Each of the bits that make up byte 6 independently represents the status of the function listed. A logic one in any bit represents active function.	
Bits	Definition
0	Amplitude Markers
1	Not used (Always=1)
2	Not used (Always=1)
3	Not used (Always=0)
4	Not used (Always=0)
5	Save Lock
6	Alt. Sweep Mode
7	Keyboard Shifted
BYTE 7	
Bits 0 and 1 of byte 7 contain a binary number that indicates ALC Leveling Mode. Bits 2, 3, 4, and 5 independently represent the status of the functions listed (a logic one in any one of these bits indicates active function). Bits 6 and 7 are not used.	
Bit(s)	Definition/Function
0-1	ALC Leveling Mode
0	0=Internal
1	1=External
2	2=Power Meter
2	Not used (Always=0)
3	Power Sweep
4	Power Slope
5	RF Power Output
6, 7	Not used

**BYTE 8**

Each of the bits in byte 8 independently represents the status of the functions listed. A logic one in any bit indicates active function.

Bit	Definition
0	Not used (Always=0)
1	Not used (Always=0)
2	Not used (Always=1)
3	Pulse Modulation
4	Frequency Modulation
5	Amplitude Modulation
6	YTM Peaking (always 0)
7	Penlift at Bandcross (always 0)

**NOTE**

If the command “**EM1**” is sent to the 83570, (Default/PRESET condition is EMO), the mode string will be 14 bytes long instead of the standard 8 bytes, and will contain information about all 10 markers instead of just markers 1-5. The **14-byte** mode string differs from the **8-byte** mode string in the following manner:

**BYTE 2:** If in extended marker mode, different active function **codes** are **used** for the markers.

<b>Marker 0:</b>	236	<b>Marker 5:</b>	241
<b>Marker 1:</b>	237	<b>Marker 6:</b>	242
<b>Marker 2:</b>	238	<b>Marker 7:</b>	243
<b>Marker 3:</b>	239	<b>Marker 8:</b>	244
<b>Marker 4:</b>	240	<b>Marker 9:</b>	245

The original active function codes (15-19) for markers 1-5 are not used in extended marker mode.

**BYTE 3** (Active Marker/Delta Reference Marker) is set to hex FF in extended marker mode. This serves as an indicator to the receiver that the 8370 is in extended marker mode.

**BYTE 9** contains the number of the currently active marker.



BYTE 10 contains the number of the Delta reference Marker.

BYTE 11-14: Marker On/Off status

	Bit Number								
	7	6	5	4	3	2	1	A	
BYTE 11	7	6	5	4	3	2	1	A	
BYTE 12	15	14	13	12	11	10	9	8	
BYTE 13	23	22	21	20	19	18	17	16	
BYTE 14	31	30	29	28	27	26	25	24	
	Marker Numbers								
	bit=0 (Marker off)				bit= 1 (Marker on)				

---

## Interrogate Function

Selected with the "OP" program code and the program code for the function to be interrogated, the sweeper will output the present numeric value of the selected function. The units of the output data are Hz, **dBm**, **dB**, or sec., implied with the function selected.

Format: [ $\pm$ d.dddddE $\pm$ dd] [LF] [EOI] or [ $\pm$ d\*\*\*d.d\*\*\*d] [LF] [EOI]

---

## Active Function

Selected with the "OA" program code, the sweeper will output the present numeric value of the presently active function (ie. enabled for modification from the keyboard or step keys). The units of the output data are Hz, **dBm**, **dB**, or sec., implied with the function selected.

Format: [ $\pm$ d.dddddE $\pm$ dd] [LF] [EOI] or [ $\pm$ d\*\*\*d.d\*\*\*d] [LF] [EOI]

---

## Status

Selected with the "OS" program code, the sweeper will output 3 sequential bytes, 8 bits wide, giving the present instrument status. The first status byte is equivalent to the Status Byte of the Serial Poll, the second and third status bytes are the Extended Status Bytes which provide additional information. See Table 2 for a description of each Status Byte. Status Byte values are cleared upon execution of a Serial Poll (Status Byte message), Device Clear Message, Power On, and/or the "CS" (Clear Status Byte) program code.

---

## Trigger

The sweeper responds to **HP-IB** Commands Group Execute Trigger (GET) and Selective Device Trigger (SDT) when it is in the SINGLE SWEEP mode. Receipt of either command causes the 83750 Series to start a sweep if the sweep had been previously reset; if not, the command is ignored. The Trigger commands are primarily used to begin a sweep when the 83750 Series is in SINGLE SWEEP mode.

**Table 1. Input Programming Codes (1 of 8)**

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE					
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE				
<b>SWEEP LIMITS/MODE</b>										
start/stop sweep	START	FA	1, 2, 3, 4	GZ	GHz	Sweeper Frequency Limits				
	STOP	FB								
Center Frequency /Δ Sweep	CF	CF								
	AF	DF								
CW Frequency	c w	CW								
	SWEPT CW	SHCW								
Frequency Offset	OFFSET	SHVR								
Frequency Vernier	VERNIER	VR								
Display Offset	DISPLAY OFFSET	SHFB								±99.9 GHz
Display Multiplier	DISPLAY MULTIPLIER	SHFA					3			1-36
Lock Display Multiplier	MULTIPLIER LOCK	SHAL								
Coarse CW Control Knob Resolution	COARSE CONTROL*	SHCF*				Sweeper Frequency Limits				
Fine CW Control Knob Resolution	FINE CONTROL*	SHDF*								
Numeric Display Update	BLANK DISPLAY	SH01								
	REGENERATE DISPLAY	SH02								
*This command is accepted by the 83750 series, but no action is performed because the FUNCTION does not exist.										

**Table 1. input Programming Codes (2 of 8)**

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
<b>FREQUENCY MARKERS</b>						
Turn On and Set Marker <b>Frequency*</b>	<b>MARKER1</b>	<b>M1</b>	1, 2, 3, 4	GZ Mz Kz HZ	<b>GHz</b> MHZ <b>kHz</b> HZ	Sweeper Frequency Limits
	<b>MARKER2</b>	<b>M2</b>				
	<b>MARKER3</b>	<b>M3</b>				
	<b>MARKER4</b>	<b>M4</b>				
	<b>MARKER5</b>	<b>M5</b>				
Turn Off A Frequency Marker*	<b>M1 OFF</b>	<b>M1</b>		<b>M0</b>		
	<b>M2 OFF</b>	<b>M2</b>				
	<b>M3 OFF</b>	<b>M3</b>				
	<b>M4 OFF</b>	<b>M4</b>				
	<b>M5 OFF</b>	<b>M5</b>				
Turn Off All Markers	<b>ALL OFF</b>	<b>SHM0</b>				
Turn On and Set Mkr $\Delta^*$	<b>MKR <math>\Delta</math>, Marker "m"</b>	<b>SHM1</b>	<b>Mm</b>			where: <b>m=1-9, A</b>
Turn Off Mkr $\Delta$	<b>MKR <math>\Delta</math> OFF</b>	<b>M0</b>				
Active Marker to Center Frequency	<b>MKR <math>\rightarrow</math> CF</b>	<b>MC</b>				
Marker 1-2 Sweep	<b>MARKER SWEEP ON</b>	<b>MP1</b>				
	<b>MARKER SWEEP OFF</b>	<b>MP0</b>				
Marker 1 to Start	<b>M1 <math>\rightarrow</math> ST</b>	<b>SHMP</b>				
Marker 2 to Stop	<b>M2 <math>\rightarrow</math> SP</b>					
<p><b>*HP 83750 Series Synthesized Sweepers have ten markers. In HP 8350 compability mode, the other markers are accessed by M6, M7, M8, M9, and MA</b></p>						

**Table 1. Input Programming Codes (3 of 8)**



MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
<b>SWEEP TRIGGER TYPE</b>						
Sweep Trigger Mode	INTERNAL	T1				
	LINE*	T2*				
	EXTERNAL	T3				
	SINGLE	T4				
sweep Type	EXTERNAL SWEEP*	SX*				
	MANUAL SWEEP FREQUENCY	SM	1, 2, 3, 4	GZ	GHz	Present Start/Stop Frequency
				MZ	MHz	
				KZ	kHz	
				HZ	Hz	
	SWEEP TIME	ST	1, 2, 3, 4	SC	sec.	Typically .01 to 100 sec.
MS				10 <sup>-3</sup> sec.		
<b>MODULATION/BLANKING</b>						
Amplitude Frequency Markers	AMPTD MKR ON	AK1				
	AMPTD MKR OFF	AK0				
Display Blanking	DISP BLANK ON	DP1				
	DISP BLANK OFF	DP0				
RF Blanking	RF BLANK ON*	RP1				
	RF BLANK OFF*	RP0				
Square Wave Modulation	 MOD ON	MD1				
	 MOD ON	MD0				
This command is accepted by the HP 83750 series but no action is performed because the function does not exist.						

Table 1. Input Programming Codes (4 of 8)

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
<b>STEP FUNCTIONS</b>						
Setting Frequency Step Size	<b>FREQUENCY STEP SIZE</b>	SF	1, 2, 3, 4	GZ	GHz	0 to 100% of sweeper BW
				MZ	MHz	
				KZ	kHz	
				HZ	Hz	
Setting Power Step Size	<b>POWER STEP SIZE</b>	SP	1, 2, 3, 4	DB or DM	dB	
Resetting Step Sizes to Default* Values	<b>DEFAULT STEP SIZES*</b>	SHSS				
Increment Active Parameter	<b>STEP UP ↑</b>	UP				
Decrement Active Parameter	<b>STEP DOWN ↓</b>	DN				
<b>INSTRUMENT STATE</b>						
Instrument State	<b>INSTR PRESET</b>	IP				
Instrument Preset, Unlock/Reset Multiplier To 1	<b>MULTIPLIER UNLOCK, INSTR PRESET</b>	SHIP				
Saving An Instrument State	<b>SAVE n</b>	SV	4		1	Registers 1 through 9
Recalling An Instrument State	<b>RECALL n</b>	RC				
Lock Registers	<b>SAVE LOCK</b>	SHSV				
Unlock Registers	<b>SAVE UNLOCK</b>	SHRC				
Unlock/Presets All Registers	<b>SAVE UNLOCK/ INSTR PRESET</b>	SH03				
Alternate Sweep Mode	<b>ALT n ON</b>	AL1				
	<b>ALT n OFF</b>	AL0				
Undergo Self Test**	<b>SELF TEST #nn</b>	SH	3		1	00-99
<p>This command is accepted by the HP 83750 series but no action is performed because the <b>function</b> does not exist.</p> <p><b>**Note:</b> Because <b>self-tests</b> are so hardware dependent, the HP 83750 does not emulate them. Use the SCPI DIAG commands for this purpose.</p>						

**Table 1. Input Programming Codes (5 of 8)**

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
<b>SPECIAL HP-IB FUNCTIONS</b>						
Status Bytes and Service Requests	OUTPUT STATUS <i>BYTES</i>	OS				
	SERVICE REQUEST MASK	RM	7			1 byte
	REQUEST EXTENDED STATUS <b>BYTE MASK</b>	RE	7			1 byte
	REQUEST SECOND EXTENDED STATUS BYTE MASK*	RS	7			
	CLEAR STATUS <b>BYTES</b>	CS				
<b>Full</b> Learn String	OUTPUT LEARN STRING	OL				
	INPUT LEARN STRING	IL	6			1024 bytes
<b>Micro</b> Learn String	OUTPUT MICRO LEARN STRING*	OX				
	INPUT MICRO LEARN STRING*	IX		6		8 bytes
Active Mode String	OUTPUT MODE STRING**	OM	6			8 bytes
Output Active Parameter Value	OUTPUT ACTIVE VALUE	OA				
output Interrogated Parameter Value	OUTPUT INTERROGATED VALUE	OP		Interrogated Parameter Code		
Current Harmonic <b>Number</b>	OUTPUT HARMONIC NUMBER	OH				
<p><b>*This command</b> is accepted by the HP <b>83750 series</b> but no action is <b>performed</b> because the function does not exist.</p> <p><b>**This command is output only.</b></p>						

Table 1. Input Programming Codes (6 of 8)

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
Software Revision Number	OUTPUT SOFTWARE REVISION NUMBER (OUTPUT IDENTITY)	OI				
Numeric Display Update	DISPLAY UPDATE ON	DU1				
	DISPLAY UPDATE OFF	DU0				
Single Sweep Start/Stop	RESET SWEEP	RS				
	TAKE SWEEP	TS				
Network Analyzer Trigger (8410B)	NETWORK* TRIGGER	NT				
Set Output Power Level	POWER LEVEL	PL	1, 2, 3, 4	DB or DM	dBm	Plug-in Power Limits
Set ALC Power Level	ALC CONTROL	SHPS	1, 2, 3, 4	DB or DM	dBm	Plug-in Power Limits Without Using the Attenuator
Set Attenuator	ATTENUATOR CONTROL	SHSL	1, 2, 3, 4	DB or DM	dB	
Power Sweep Mode	POWER SWEEP ON	PS1	1, 2, 3, 4	DB or DM	dB/Swp	See Plug-in
	POWER SWEEP OFF	PS0				
Power Slope Mode	SLOPE ON	SL1	1, 2, 3, 4	DB or DM	db/GHz	0-5 dB
	SLOPE OFF	SL0				

\*This command is accepted by the HP 83750 series but no action is performed because the function does not exist.



**Table 1. Input Programming Codes (7 of 8)**

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
<b>ALC/SIGNAL CONTROL</b>						
ALC Leveling Modes	INTERNAL	A1				
	EXTERNAL (CRYSTAL)	A2				
	EXTERNAL POWER METER	A3				
RF Power	RF ON	RF1				
	RF OFF	RF0				
CW Filter	FILTER ON*	FI1				
	FILTER OFF*	FI0				
Crystal Marker Frequency	1 MHz*	C1				
	10 MHz*	C2				
	50 MHz*	C3				
	EXTERNAL INPUT*	C4				
Crystal Amplitude Markers	AMPL MKR ON*	CA 1				
	AMPL MKR OFF*	CA 0				
Crystal Amplitude Markers	INTEN MKR ON*	CI 1				
	INTEN MKR OFF*	CI 0				
Amplitude Markers	AMPL MKR ON	AK1				
	AMPL MKR OFF	AK0				
*This command is accepted by the HP 83750 series but no action is performed because the function does not exist.						

**Table 1. Input Programming Codes (8 of 8)**

MODE/ MODIFIERS	FUNCTION	PROGRAM CODE			NUMERIC VALUE	
		PREFIX ACTIVATE	NUMERIC FORMAT	SUFFIX	SCALE	RANGE
<b>PLUG-IN SPECIAL FUNCTIONS</b>						
FM Input Sensitivity	-20 MHz/V	<b>F1</b>				
	-6 MHz/V	<b>F2</b>				
AC Coupled FM	AC Coupled	<b>D1</b>				
Direct Coupled FM	Direct Coupled	<b>D2</b>				

---

## Clear

The sweeper responds to both Device Clear (**DCL**) and Selective Device Clear (SDC) and initializing the interface so that it is ready to receive **HP-IB** programming codes.

---

## Remote/Local Changes

The sweeper goes to the Remote state when the LREN line is true (low) and the sweeper receives its listen address. In Remote, all front panel functions are disabled except the LINE switch and the LOCAL key. The LOCAL function can also be disabled via the Local Lockout (**LLO**) command.

The sweeper goes to the Local state when it receives the Go To Local (GTL) command or when the LREN line is set false (high). If the Local Lockout (**LLO**) command has not been executed, the 83750 Series can also be set to Local by pressing the LOCAL key. In Local, the front panel is active but the instrument will still respond to **HP-IB** programming codes.

---

## Service Request

The sweeper can initiate a Service Request (SRQ) whenever one of the following conditions exists:

- New frequencies or sweep time in effect
- Error in syntax
- End of sweep
- RF settled
- Change in Extended Status Byte bit(s)
- Numeric entry completed
- Front panel key pressed

Further information can be obtained by conducting a Serial Poll or by executing the Output Status command, both of which access Status Byte information. The SRQ is cleared only by executing a Serial Poll. To select an SRQ for a particular set of circumstances, the Request Mask function can be used to determine which of the bits in the first Status Byte can cause an SRQ. The mask value is determined by summing the decimal values of each selected function/condition that is desired. The default Request Mask at power on is '00000000' or decimal 0. SRQ generation due to conditions indicated by the Extended Status Byte can be masked by using the "RE" function, in conjunction with masking bit 2 of the first Status Byte. The "RE" default mask value at power on is "00000000" or decimal 255. All mask values are reset to the default values only at power on. The Second Extended Status Byte is not used and always set to 0.

---

## Status Byte

The sweeper responds to a Serial Poll by sending its status byte as indicated in Table 2. The Extended Status Bytes are available but must be accessed via the Output Status command. When Bit 6 (Request Service) of the Status Byte is true (one), an SRQ has occurred. See Service Request for the conditions causing a Service request. Bit 2 indicates whether a change has occurred in the Extended Status Bytes. If Bit 2 is true, then the extended status bytes should be accessed via the Output Status function to determine the cause of the status change. All other bits (5, 4, 0) indicate the present status of the noted function. The bits are true (one) only if the associated function/condition is true.

---

## Status Bit

The sweeper does not respond to a Parallel Poll.

---

## Pass Control

The sweeper does not have the ability to pass control, and can only take control when loading firmware **from** a disk or controlling a power meter during a flatness adjustment.

---

## Abort

The 83750 Series responds to the Abort message (Interface Clear - IFC true) by stopping all Listener or Talker functions.

---

---

## Interface Function Codes

<b>AH1</b>	Acceptor Handshake-full capability
<b>T6</b>	Basic Talker- Serial Poll capability
<b>L4</b>	Basic Listener- Unaddressed if MLA
<b>SR1</b>	Service Request-full capability
<b>RL1</b>	Remote Local-complete capability
<b>PP0</b>	Parallel Poll-no capability
<b>DC1</b>	Device Clear-full capability
<b>DT1</b>	Device Trigger-full capability
<b>co</b>	Controller- no capability
<b>SH1</b>	Source Handshake-full capability
<b>E1</b>	Driver Electronics-open collector

Table 2. 83750 Series Status Byte Descriptions (1 of 1)

STATUS BYTE (#1)								
BIT#	7	6	5	4	3	2	1	0
DECIMAL VALUE	128	64	32	16	8	4	2	1
FUNCTION	SRQ on New Frequencies or Sweeptime in Effect	Request Service (RQS)	SRQ on HP-IB Syntax Error	SRQ on End of Sweep or Mid-Sweep Update	SRQ on RF Settled	SRQ on Change in Extended Status Bytes	SRQ on Numeric Enter Completed (HP-IB or Front Panel)	SRQ on Any Front Panel Key Pressed
EXTENDED STATUS BYTE (#2)								
BIT#	7	6	5	4	3	2	1	0
DECIMAL VALUE	128	64	32	16	8	4	2	1
FUNCTION	N/A	RF Unleveled	Power Failure/On	RF Unlocked	N/A	N/A	N/A	Self Test Failed
SECOND EXTENDED STATUS BYTE (#3)								
BIT#	7	6	5	4	3	2	1	0
DECIMAL VALUE	128	64	32	16	8	4	2	1
FUNCTION	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

**NOTE** The Second Extended Status Byte is always zero. It is output for compatibility with the HP 8350 Sweep Oscillators.

---

Error Messages

# Error Messages

If an error condition occurs in the sweeper, it will always be reported to both the front panel and GPIB error queues. These two queues are viewed and managed separately. The **MSG** key is used to view the contents of the front panel error queue. The GPIB query “SYSTem:ERRor?” is used to view the contents of the GPIB error queue.

If there are any error messages in the front panel error queue, the front panel **MSG** annunciator will be lit. Pressing the **MSG** key repeatedly until the **MSG** annunciator turns off will empty the front panel error queue. The **MSG** key has no affect on the GPIB error queue. Emptying the GPIB error queue has no affect on the front panel queue, therefore, it will not affect the **MSG** annunciator.

There are some special error types that are called permanent errors. Permanent errors remain in the error queues until the error condition is cleared. Pressing the **MSG** key will empty the front panel error queue, but the permanent errors will be re-reported if the error conditions still exist. In the GPIB error queue, the permanent errors are re-reported after the message, 0, “No error” is read using the “SYSTem:ERRor?” query or after the “\*CLS” command is executed.



# :ERRor? SYSTem:ERRor

The queue query message is a request for the next entry from the instrument's error/event queue. This queue contains an integer in the range [ -32768, 32767 ]. Negative error numbers are reserved by the SCPI standard and defined first in this document. Positive error numbers are instrument-dependent. An error/event value of zero indicates that no error or event has occurred (see next section, "The queue ").

This command is required of all SCPI implementations. STATus:QUEue? is an alias to SYSTem:ERRor?.

The instrument responds to SYSTem:ERRor? query using the following form:  
<error/event number>, <error discription>

The <error/event number> is a unique error/event descriptor. Certain standard error/event numbers are described in this document. The <error description> is a short description of the error/event, (optionally) followed by further information about the error/event. Short descriptions of the standard error/event numbers are given in this document; information following the error message contains corrective actions that should be followed in order to correct the error condition.

The <device-dependent info> part of the response may contain information which will allow the user to determine the exact error/event and context. For example,

```
-131,"Invalid suffix;FREQUENCY:CENT 2.0E+5 dBuV"
```

The maximum string length of <error\_description> plus <device- dependent information> is 255 characters. The <error description> shall be sent exactly as indicated in this document including case.

If there has been no error/event, that is, if the queue is empty, the instrument should respond with

```
0, "No error"
```

If there has been more than one error, the instrument should respond with the first one in its queue. Subsequent responses to SYSTem:ERRor? should continue with the que until it is empty. Note that the string should be sent exactly as indicated in this document, especially with reference to case.

## The Error/Event Queue

As errors and events are detected, they are placed in a queue. This queue is first in, first out. If the queue overflows, the last error/event in the queue is replaced with error

–350 "Queue overflow"

Any time the queue overflows, the least recent errors remain in the queue, and the most recent error/event is discarded. The minimum length of the error/event queue is 2, one position for the first error, and one for the "Queue overflow" message. Reading an error/event from the head of the queue removes that error/event from the queue, and opens a position at the tail of the queue for a new error/event, if one is subsequently detected.

When all errors/events have been read from the queue, further error/event queries shall return

0, "No error"

Individual errors and events may be enabled into the queue. The STATus:QUEue:ENABle command accomplishes this. At STATus:PRESet, only errors are enabled. This means that both SYSTem:ERRor? and STATus:QUEue[:NEXt] report only errors unless the user changes the enable mask.

The error/event queue shall be cleared when any of the following occur (IEEE 488.2, section 11.4.3.4):

- Upon power up.
- Upon receipt of a \*CLS command.
- Upon reading the last item from the queue.

---

## Error numbers

The system-defined error/event numbers are chosen on an enumerated (“1 of N”) basis. The SCPI-defined error/event numbers and the <error description> portions of the ERRor query response are listed here. The first error/event described in each class (for example, –100, –200, –300, –400) is a “generic” error. In selecting the proper Error/event number to report, more specific error/event codes are preferred, and the generic error/event is used only if the others are inappropriate.

---

## No Error

This message indicates that the device has no errors.

Error Number	Error Description [description/explanation/examples]
--------------	--

0	No error
---	----------

The queue is completely empty. Every error/event in the queue has been read or the queue was purposely cleared by power-on, \*CLS, and so forth.

---

## Error Message Description

The list of error messages in this chapter lists all of the error messages associated with sweeper operation. An example of the error format found in the list of error messages is as follows:

Example Error

---

403    **-222, "Data out of range;Test Patch Value Out of Range (403)"**

Indicates that user has entered a Self-Test Patch with upper or lower limit values greater than allowed. All upper and lower limits of these Self-Test Patches must be within the range of +32767 to -32768.

---

The following explains each element of an error message listing.

- **Manual Error Number** – The number 403 to the left and in the parenthesis is called the Manual Error Number. The error message list is organized in ascending order off the manual error number. The manual error number will always be found in the parenthesis contained in the message.
- **Error Message** – The bold text **-222, "Data out of range;Test Patch Value Out of Range"** is the error message. When the **MSG** key is pressed, the error message is displayed in the leftmost display. The entire message is returned by the GPIB query "SYSTEM:ERRor?". The error message contains the following parts:
  - **SCPI Error Number** – The standard SCPI error number (-222 in the example) usually differs from the manual error number because the manual error number is unique for every possible message. Standard SCPI error numbers are always negative (except for 0, "No error"). If there is no standard SCPI error number for a message, the manual error number replaces it in the error message.

- **SCPI Error Message** – The SCPI error message is **Data out of range** in the example.
- **Detailed Description** – All information after the semicolon (;) is a detailed description of what exactly caused the error. In the example, **Test Patch Value Out of Range** tells you that the user has entered a Self-Test Patch with upper or lower limit values greater than allowed. If no detailed description exists, it will be omitted from the message.
- **Action Required** – The text that appears below each error message listing contains corrective actions that should be followed in order to correct the error condition. Note that the action required is never shown in the sweeper display.

## Command Error

An <error/event number> in the range [ -199, -100 ] indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class shall cause the command error bit (bit 5) in the event status register (IEEE 488.2, section 11.5.1) to be set. one of the following events has occurred:

- An IEEE 488.2 syntax error has been detected by the parser. That is, a controller-to-device message was received which is in violation of the IEEE 488.2 standard. Possible violations include a data element which violates the device listening formats or whose type is unacceptable to the device.
- An unrecognized header was received. Unrecognized headers include incorrect device-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 <PROGRAM MESSAGE>.

Events that generate command errors shall not generate execution errors, device-specific errors, or query errors; see the other error definitions in this chapter.

Error Number	Error Description [description/explanation/examples]
-100	<p><b>Command error</b></p> <p>This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that a Command Error as defined in IEEE 488.2, 11.5.1.1.4 has occurred.</p>
-101	<p><b>Invalid character</b></p> <p>A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&amp;. This error might be used in place of errors -114, -121, -141, and perhaps some others.</p>
-102	<p><b>Syntax error</b></p> <p>An unrecognized command or data type was encountered; for example, a string was received when the device does not accept strings.</p>
-103	<p><b>Invalid separator</b></p> <p>The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit, *EMC 1 :CHI:VoLTS 5.</p>
-104	<p><b>Data type error</b></p> <p>The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.</p>
-105	<p><b>GET not allowed</b></p> <p>A Group Execute Trigger was received within a program message (see IEEE 488.2, 7.7). Correct the GPIB controller program so that the group execute trigger does not occur within a line of GPIB program code.</p>
-108	<p><b>Parameter not allowed</b></p> <p>More parameters were received than expected for the header; for example, the *EMC common command only accepts one parameter, so receiving *EMC 0,1 is not allowed.</p>

**SCPI Error Messages**

- 109     **Missing parameter**  
Fewer parameters were received than required for the header; for example, the \*EMC common command requires one parameter, so receiving \*EMC is not allowed.
- 110     **Command header error)**  
An error was detected in the header. This error message should be used when the device cannot detect the more specific errors described for errors –111 through –119.
- 111     **Header separator error**  
A character which is not a legal header separator was encountered while parsing the header; for example, no white space followed the header, thus \*GMC“MACRO” is an error.
- 112     **Program mnemonic too long**  
The header contains more than twelve characters (see IEEE 488.2, 7.6.1.4.1).
- 113     **Undefined header**  
The header is syntactically correct, but it is undefined for this specific device; for example, \*XYZ is not defined for any device.
- 114     **Header suffix out of range**  
The value of a numeric suffix attached to a program mnemonic makes the header invalid.
- 120     **Numeric data error**  
This error, as well as errors –121 through –129, are generated when parsing a data element which appears to be numeric, including the nondecimal numeric types. This particular error message should be used if the device cannot detect a more specific error.
- 121     **Invalid character in number**  
An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric or a “9” in octal data.



- 123     **Exponent too large**

The magnitude of the exponent was larger than 32000 (see IEEE 488.2, 7.7.2.4.1).
- 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).
- 128     **Numeric data not allowed**

A legal numeric data element was received, but the device does not accept one in this position for the header.
- 130     **Suffix error**

This error, as well as errors –131 through –139, are generated when parsing a suffix. This particular error message should be used if the device cannot detect a more specific error.
- 131     **Invalid suffix**

The suffix does not follow the syntax described in IEEE 488.2, 7.7.3.2, or the suffix is inappropriate for this device.
- 134     **Suffix too long**

The suffix contained more than 12 characters (see IEEE 488.2, 7.7.3.4).
- 138     **Suffix not allowed**

A suffix was encountered after a numeric element which does not allow suffixes.
- 140     **Character data error**

This error, as well as errors –141 through –149, are generated when parsing a character data element. This particular error message should be used if the device cannot detect a more specific error.
- 141     **Invalid character data**

Either the character data element contains an invalid character or the particular element received is not valid for the header.

**SCPI Error Messages**

- 144     **Character data too long**  
The character data element contains more than twelve characters (see IEEE 488.2, 7.7.1.4).
- 148     **Character data not allowed**  
A legal character data element was encountered where prohibited by the device.
- 150     **String data error**  
This error, as well as errors –151 through –159, are generated when parsing a string data element. This particular error message should be used if the device cannot detect a more specific error.
- 151     **Invalid string data**  
A string data element was expected, but was invalid for some reason (see IEEE 488.2, 7.7.5.2); for example, an END message was received before the terminal quote character.
- 158     **String data not allowed**  
A string data element was encountered but was not allowed by the *device* at this point in parsing.
- 160     **Block data error**  
This error, as well as errors –161 through –169, are generated when parsing a block data element. This particular error message should be used if the device cannot detect a more specific error.
- 161     **Invalid block data**  
A block data element was expected, but was invalid for some reason (see IEEE 488.2, 7.7.6.2); for example, an END message was received before the length was satisfied.
- 168     **Block data not allowed**  
A legal block data element was encountered but was not allowed by the device at this point in parsing.

- 170     **Expression error**  
This error, as well as errors –171 through –179, are generated when parsing an expression data element. This particular error message should be used if the device cannot detect a more specific error.
- 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.
- 178     **Expression data not allowed**  
A legal expression data was encountered but was not allowed by the device at this point in parsing.
- 180     **Macro error**  
This error, as well as errors –181 through –189, are generated when defining a macro or executing a macro. This particular error message should be used if the device cannot detect a more specific error.
- 181     **Invalid outside macro definition**  
Indicates that a macro parameter placeholder ( $\$<number>$ ) was encountered outside of a macro definition.
- 183     **Invalid inside macro definition**  
Indicates that the program message unit sequence, sent with a \*DDT or \*DMC command, is syntactically invalid (see IEEE 488.2, 10.7.6.3).
- 184     **Macro parameter error**  
Indicates that a command inside the macro definition had the wrong number or type of parameters.

---

## Execution Error

An <error/event number> in the range [ -299, -200 ] indicates that an error has been detected by the instrument's execution control block. The occurrence of any error in this class shall cause the execution error bit (bit 4) in the event status register (IEEE 488.2, section 11.S.1) to be set. one of the following events has occurred:

- A <PROGRAM DATA> element following a header was evaluated by the device as outside of its legal input range or is otherwise inconsistent with the device's capabilities.
- A valid program message could not be properly executed due to some device condition.

Execution errors shall be reported by the device after rounding and expression evaluation operations have taken place. Rounding a numeric data element, for example, shall not be reported as an execution error. Events that generate execution errors shall not generate Command Errors, device-specific errors, or Query Errors; see the other error definitions in this section.

Error Number	Error Description [description/explanation/examples]
-200	<p><b>Execution error</b></p> <p>This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that an Execution Error as defined in IEEE 488.2, 11.5.1.1.5 has occurred.</p>
-201	<p><b>Invalid while in local</b></p> <p>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 cannot be executed.</p>
-202	<p><b>Settings lost due to rtl</b></p> <p>Indicates that a setting associated with a hard local control (see IEEE 488.2, 5.6.15) was lost when the device changed to LOCS from REMS or to LWLS from RWLS.</p>
-210	<p><b>Trigger error</b></p> <p>A trigger error occurred in the signal generator.</p>
-211	<p><b>Trigger ignored</b></p> <p>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. Note: a DT0 device always ignores GET and treats *TRG as a Command Error.</p>
-212	<p><b>Arm ignored</b></p> <p>Indicates that an arming signal was received and recognized by the device but was ignored.</p>
-213	<p><b>Init ignored</b></p> <p>Indicates that a request for a measurement initiation was ignored as another measurement was already in progress.</p>

**SCPI Error Messages**

- 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.
- 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.
- 220     **Parameter error**  
Indicates that a program data element related error occurred. This error message 0 should be used when the device cannot detect the more specific errors described for errors –221 through –229.
- 221     **Settings conflict**  
Indicates that a legal program data element was parsed but could not be executed due to the current device state (see IEEE 488.2, 6.4.5.3 and 11.5.1.1.5.).
- 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.).
- 223     **Too much data**  
Indicates that a legal program data element of block, expression, or string type was received that contained more data than the device could handle due to memory or related device-specific requirements.
- 224     **Illegal parameter value**  
Used where exact value, from a list of possibilities, was expected.
- 225     **Out of memory.**  
The device has insufficient memory to perform the requested operation.

- 226     **Lists not same length.**  
 Attempted to use LIST structure having individual LIST's of unequal lengths.
- 230     **Data corrupt or stale**  
 Possibly invalid data; new reading started but not completed since last access.
- 231     **Data questionable**  
 Indicates that measurement accuracy is suspect.
- 240     **Hardware error**  
 Indicates that a legal program command or query could not be executed because of a hardware problem in the *device*. Definition of what constitutes a hardware problem is completely device-specific. This error message should be used when the *device* cannot detect the more specific errors described for errors –241 through –249.
- 241     **Hardware missing**  
 Indicates that a legal program command or query could not be executed because of missing *device* hardware; for example, an option was not installed. Definition of what constitutes missing hardware is completely device-specific.
- 260     **Expression error**  
 Indicates that a expression program data element related error occurred. This error message should be used when the device cannot detect the more specific errors described for errors –261 through –269.
- 261     **Math error in expression**  
 Indicates that a syntactically legal expression program data element could not be executed due to a math error; for example, a divide-by-zero was attempted. The definition of math error is device-specific.

**SCPI Error Messages**

- 270     **Macro error**  
Indicates that a macro-related execution error occurred. This error message should be used when the device cannot detect the more specific errors described for errors –271 through –279.
- 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.)
- 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.).
- 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.
- 274     **Macro parameter error**  
Indicates that the macro definition improperly used a macro parameter placeholder (see IEEE 488.2, 10.7.3).
- 275     **Macro definition too long**  
Indicates that a syntactically legal macro program data sequence could not be executed because the string or block contents were too long for the device to handle (see IEEE 488.2, 10.7.6.1).



- 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).
- 277      **Macro redefinition not allowed**  
Indicates that a syntactically legal macro label in the \*DMC command could not be executed because the macro label was already defined (see IEEE 488.2, 10.7.6.4).
- 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.

## Device-Specific Error

An <error/event number> in the range [ -399, -300 ] or [ 1, 32767 ] indicates that the instrument has detected an error which is not a command error, a query error, or an execution error; some device operations did not properly complete, possibly due to an abnormal hardware or firmware condition. These codes are also used for self-test response errors. The occurrence of any error in this class should cause the device-specific error bit (bit 3) in the event status register (IEEE 488.2, section 11.5.1) to be set. The meaning of positive error codes is device-dependent and may be enumerated or bit mapped; the <error message> string for positive error codes is not defined by SCPI and available to the device designer. Note that the string is not optional; if the designer does not wish to implement a string for a particular error, the null string should be sent (for example, 42, ""). The occurrence of any error in this class should cause the device-specific error bit (bit 3) in the event status register (IEEE 488.2, section 11.5.1) to be set. Events that generate device-specific errors shall not generate command errors, execution errors, or query errors; see the other error definitions in this section.

<b>Error Number</b>	<b>Error Description [description/explanation/examples]</b>
-300	<b>Device-specific error</b>  This is the generic device dependent error for devices that cannot detect more specific errors. This code indicates only that a Device-Dependent Error as defined in IEEE 488.2, 11.5.1.1.6 has occurred.
-310	<b>System error</b>  Indicates that some error, termed “system error” by the device, has occurred. This code is device-dependent.
-311	<b>Memory error</b>  Indicates that an error was detected in the device’s memory. The scope of this error is device-dependent.
-314	<b>Save/recall memory lost</b>  Indicates that the nonvolatile data saved by the *SAV? command has been lost.
-315	<b>Configuration memory lost</b>  Indicates that nonvolatile configuration data saved by the device has been lost. The meaning of this error is device-specific.
-330	<b>Self-test failed</b>
-350	<b>Queue overflow</b>  A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.

## Query Error

An <error/event number> in the range [ -499, -400] indicates that the output queue control of the instrument has detected a problem with the message exchange protocol described in IEEE 488.2, chapter 6. The occurrence of any error in this class shall cause the query error bit (bit 2) in the event status register (IEEE 488.2, section 11.5.1) to be set. These errors correspond to message exchange protocol errors described in IEEE 488.2, section 6.5. One of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending;
- Data in the output queue has been lost.

Events that generate query errors shall not generate command errors, execution errors, or device-specific errors; see the other error definitions in this section.

Error Number	Error Description [description/explanation/examples]
-400	<b>Query error</b>  This is the generic query error for <i>devices</i> that cannot detect more specific errors. This code indicates only that a Query Error as defined in IEEE 488.2, 11.5.1.1.7 and 6.3 has occurred.
-410	<b>Query INTERRUPTED</b>  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.
-420	<b>Query UNTERMINATED</b>  Indicates that a condition causing an UNTERMINATED Query error occurred (see IEEE 488.2, 6.3.2.2); for example, the <i>device</i> was addressed to talk and an incomplete program message was received.
-430	<b>Query DEADLOCKED</b>  Indicates that a condition causing an 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.
-440	<b>Query UNTERMINATED after indefinite response</b>  Indicates that a query was received in the same program message after an query requesting an indefinite response was executed (see IEEE 488.2, 6.3.7.5).

---

## Instrument Specific Error Messages

---

### Block Transfer Errors

- 101 – 161, “Invalid block data;Too Many Calibration Array Elements Sent (101)”
- For a specific calibration array, the GPIB controller has sent more array elements than needed by the array definition.
- 102 – 161, “Invalid block data;Incorrect Number Of Calibration Array Elements (102)”
- For a specific calibration array, the GPIB controller has sent an incorrect number of array elements than needed by the array definition.
- 103 – 161, “Invalid block data;Bad Learn String Checksum (103)”
- Indicates that an incoming learn string was rejected because the newly calculated checksum did not match the original checksum stored with the learn string.

---

## Bus Control Errors

- 201      –310, “System error;Another Controller Is On The GPIB Bus (201)”  
Indicates that during a Flatness Calibration, the instrument was trying to establish the control of the Power Meter, but figured out another controller is on the GPIB bus. Flatness Calibration is aborted.
- 204      –310, “System error;Command Send Error—No GPIB Devices Found (204)”  
Indicates that during a Flatness Calibration, the instrument was sending a command to an GPIB device, but could not find it. Flatness Calibration is aborted.
- 205      –310, “System error;Cannot Find Power Meter On GPIB Bus (205)”  
Indicates that during a Flatness Calibration, the instrument was trying to establish the control of a supported Power Meter, but could not find it. Flatness Calibration is aborted.
- 206      –310, “System error;Meter Returns Error Message (206)”  
Indicates that during a Flatness Calibration, the GPIB Power Meter error checking returns an error message of some type.
- 207      –310, “System error;Meter Data Measured Is Invalid or Out Of Range (207)”  
Indicates that during a Flatness Calibration, a reading return value which GPIB Power Meter measured is invalid or out of range. Flatness Calibration is aborted.
- 208      –310, “System error;Unable To Receive Message From Meter (208)”  
Indicates that during a Flatness Calibration, a time out is happened while the instrument was waiting to receive a message from the Power Meter. Flatness Calibration is aborted.

---

## Parsing and Compatibility Errors

- 301 –178, “Expression data not allowed;C[1-4]: No External Crystal Marker Allowed (301)”
- Indicates that one of the commands “C1”, “C2”, “C3”, or “C4” were detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 302 –178, “Expression data not allowed;CA: No Amplitude Crystal Marker Allowed (302)”
- Indicates that the command “CA” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 303 –178, “Expression data not allowed;CI: No Intensity Crystal Markers Allowed (303)”
- Indicates that the command “CI” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 304 –178, “Expression data not allowed;DP: Display Blanking is always ON (304)”
- Indicates that the command “DP” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 305 –178, “Expression data not allowed;IX, OX: No Micro Learn Strings Allowed (305)”
- Indicates that the commands “IX” or “OX” were detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.



- 306 –178, “Expression data not allowed;NT: Network Analyzer Trigger Ignored (306)”
- Indicates that the command “NT” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 307 –178, “Expression data not allowed;RP: RF Blanking Is Always ON (307)”
- Indicates that the command “RP” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 308 –178, “Expression data not allowed;SHCF: No Coarse CW Resolution Allowed (308)”
- Indicates that the command “SHCF” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 309 –178, “Expression data not allowed;SHDF: No Fine CW Resolution Allowed (309)”
- Indicates that the command “SHDF” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.
- 310 –178, “Expression data not allowed;SHM2, SHM3: No Counter Interface (310)”
- Indicates that the commands “SHM2” or “SHM3” were detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.

**SCPI Error Messages**

311 –178, “Expression data not allowed;SHSS: No Default Step Sizes Allowed (311)”

Indicates that the command “SHSS” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.

312 –178, “Expression data not allowed;SX: No External Sweep Allowed (312)”

Indicates that the command “SX” was detected while the instrument was using the 8350 compatible language. These commands are accepted but no action is taken because the instrument does not have this feature.

---

## Diagnostics and Self-Test Errors

- 401        –300, “Device specific error;Test Patch Table Overflow (401)”  
 Indicates that a Self-Test Patch was requested for storage in EEPROM Patch Table, but the table already has the maximum allowed (50).
- 402        –300, “Device specific error;Illegal Test Patch Name (402)”  
 Indicates that an illegal Self-Test Patch <name> has been acquired to set a Self-Test Patch in EEPROM. Node <name> must be a test node and it cannot be a menu node. Any self test whose name is preceded by \* (on the front panel display) is a self test menu. [ By convention, any name which starts with an assembly number (e.g. A4CPU, A12RFintf . . . ) is a menu. And any name that contains the word 'Menu' is a menu. However, not all entries were able to follow this convention due to display width limitations. ]
- 403        –222, “Data out of range;Test Patch Value Out Of Range (403)”  
 Indicates that user has entered a Self-Test Patch with upper or lower limit values greater than allowed. All upper and lower limits of these Self-Test Patches must be with in the range of +32767 to –32768.
- 404        –220, “Parameter error;Incorrect Number of Parameters (404)”  
 Indicates that user has entered too many or not enough parameters to complete the entry for a Self-Test Patch. Parameters required to enter a Self-Test Patch are <name>, <upperLim>, <lowerLim>, and <patchType>. Refer to the Service Manual for more information.
- 405        –330, “Self-test failed;Self Test Patches Lost (405)”  
 The conditions indicated by this error are: (1) firmware has been upgraded and the test patch table has been initialized. Refer to service documentation for the appropriate patch table entries associated with the new firmware revision. (2) SRAM and EEPROM test patch tables have been corrupted and are incorrecable. Refer to service documentation for troubleshooting information.

**SCPI Error Messages**

- 406 –330, “Self-test failed;Self Test Patch Table Locked (406)”  
Indicates that segment 7 of the CPU board DIP switch is closed, prohibiting modification of the test patch table. Switch 7 must be in the open position to allow modification.
- 407 –330, “Self-test failed;Instrument Bus Error Occurred (407)”  
As part of the power on process, the cpu attempts to write and read a special latch on the A5 timer board to verify the integrity of the instruments data and address bus. This test has failed. Refer to service documentation for troubleshooting information.
- 408 –330, “Self-test failed;Static Ram Overflow by Firmware (408)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and Static Ram was found to be overflowed by the program running in firmware.
- 409 –330, “Self-test failed;Static Ram Not Recovered Error (409)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has found that contents of Static Ram (SRAM) has been corrupted during power up. SRAM Calibration data and SRAM Instrument State have been cleared and are lost. The rear panel dip switch 7 can/may be set to deliberately cause this condition.
- 410 –330, “Self-test failed;Power Supply Voltage Error (410)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and Power Supply Voltage errors were found.
- 411 –330, “Self-test failed;CPU Self Test Error On Power Up (411)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and the CPU board tests failed.
- 412 –330, “Self-test failed;ROM CheckSum Error (LOW BYTE) (412)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has found that the FLASH ROM has a low byte error.

- 413      –330, “Self-test failed;ROM CheckSum Error (HIGH BYTE) (413)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has found that FLASH ROM has a high byte error.
- 414      –330, “Self-test failed;Boot-ROM CheckSum Error (LOW BYTE) (414)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has found that Boot-ROM has a low byte error.
- 415      –330, “Self-test failed;Boot-ROM CheckSum Error (HIGH BYTE) (415)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction haming code checking has found that Boot-ROM has a high byte error.
- 416      –330, “Self-test failed;RAM-backup battery is LOW (416)”  
Indicates SRAM-backup battery is LOW.
- 417      –330, “Self-test failed;Power Up RAM Addressing Error (417)”  
Indicates RAM Addressing Error during Power Up.
- 418      –330, “Self-test failed;Power Up RAM Test Error (LOW BYTE) (418)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and RAM Test is found to have low byte error.
- 419      –330, “Self-test failed;Power Up RAM Test Error (HIGH BYTE) (419)”  
Indicates that after the instrument is up and running, a series of power on self-tests have been run and RAM Test is found to have high byte error.

**SCPI Error Messages**

420 –330, “Self-test failed;Power Up Calibration Corrupted: Default Used (420)”

Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has found that contents of one of the calibration arrays were found corrupted. A default calibration has been used.

421 –330, “Self-test failed;Power Up Calibration Defaulted (421)”

Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has found that contents of one of the calibration arrays were found corrupted. A default calibration has been used.

422 –330, “Self-test failed;Power Up Calibration Improved (422)”

Indicates that after the instrument is up and running, a series of power on self-tests have been run and error correction code checking has corrected a one bit error when recovering data from the EEPROM. Proper operation of the instrument is guaranteed. It is suggested that a calibration save operation be performed to permanently correct this problem.

423 –330, “Self-test failed;Power Up DSP Handshake Failed (423)”

Indicates that during the series of power on self-tests, the handshake control with the Digital Signal Processor has failed.

424 –330, “Self-test failed;DSP Handshake Fail During Byte Transfer (424)”

Indicates that after the instrument is up and running, byte transfers with the Digital Signal Processor have failed.

430 –120, “Numeric data error;Entered Value is not a Valid Patch Number (430)”

Indicates that a Self-Test patch had been requested to be deleted from the eeprom Self-Test Patch Table, however, the entered value is not a valid patch number. A valid patch number is the index of the patch item in the patch table (starting at 1.) It is not the test <name>.

---

## Internal Hardware Errors

- 501 –300, “Device specific error;V/GHz DAC Out Of Range (501)”
- 502 –211, “Trigger ignored;Trigger Immediate Ignored (502)”
- 503 –211, “Trigger ignored;Sweep Trigger Immediate Ignored (503)”
- 504 –213, “Init ignored;Init Immediate Ignored (504)”
- 505 –211, “Trigger ignored;Group Execute Trigger or \*TRG Ignored (505)”

---

## Hardware Configuration Errors

- 601 –311, “Memory Error;Not Able to Recall From EEPROM: Default Used (601)”  

More than a single bit error has been detected when recovering calibration data from EEPROM. Thus, it could not be used. Default calibration data is used instead.
- 602 –311, “Memory Error;EEPROM Failure. Calibration data could not be saved (602)”  

Calibration could not be stored in EEPROM. EEPROM have been detected to have failed.
- 603 –311, “Memory Error;RECALL Was Aborted. Presetting to Fix Instrument State Used (603)”
- 604 –311, “Memory Error;SAVE/RECALL Registers Corrupted. Registers Erased (604)”
- 605 –311, “Memory Error;No Data In SAVE/RECALL Registers. RECALL Ignored (605)”

**SCPI Error Messages**

- 606        “Warning! Learn String FW Revision Not Matched (606)”  
The learn string that was received does not match the current firmware revision. It was rejected and not used.
- 607        –200, “Execution error;Execution Not Allowed. Currently In Restricted Mode (607)”  
The instrument is in a restricted mode due to either the operation of self test, or current operation of a calibration. Running of most commands is not allowed. For best results, a device clear followed by a \*rst command should be sent.
- 608        –200, “Execution error;Execution Not Allowed. Currently In Network Analyzer mode(608)”  
Frequency zero can only be executed when the instrument is in stand alone mode. When the instrument is connected to an 8757 or when the instrument is speaking 8350 compatibility language, frequency zero cannot be implemented. If the user attempts to implement frequency zero in these modes, an error message will be generated.

---

## Calibration Routine Errors

- 701        –300, “Device specific error;Peaking Failed (701)”  
For unspecified reasons, the CW peaking algorithms failed.
- 702        –300, “Device specific error;Peaking Never Leveled (702)”  
ALC could not achieved level power under the current conditions.
- 703        –300, “Device specific error;Instrument Not in CW Mode (703)”  
It is required that the instrument first be in CW mode before a YTF peak is executed.
- 704        –300, “Device specific error;No Sufficiently Wide Pass Band Was Found (704)”



- No sufficiently wide YTF pass band was found in the initial phase of the peaking algorithm.
- \705 \-300, “Device specific error;The fine\_peak phase of the peaking algorithm failed (705)”
- For unspecified reasons the later “fine peak” phase of the peaking algorithm failed.
- 706      -300, “Device specific error;SAF Tracking Failure (706)”
- The SAF tracking algorithm failed for unspecified reasons.
- 709      -300, “Device specific error;Calibration Security is LOCKED. Unable to Access Cal Data (709)”
- Current Calibration Security system is in LOCKED position. Calibration data is unable for write access. To UNLOCK the Calibration Security system, refer to the Service Guide for CPU board dip switch configuration.
- 710      -300, “Device specific error;Flatness Calibration Failed (710)”
- The Flatness calibration algorithm failed for unspecified reasons.
- 711      -300, “Device specific error;Flatness Calibration Failed (711)”
- The Flatness calibration algorithm failed due to an error in reading power from the external power meter.
- 712      -300, “Device specific error;Flatness Calibration Failed Relinquish Failure (712)”
- The Flatness calibration algorithm failed experienced difficulties in relinquishing control of the external power meter.
- 713      -300, “Device specific error;No Tracking With MMH(713)”
- The SAF tracking algorithm is not allowed to run under millimeter head personality. The millimeter head should be disconnected first.
- 714      -300, “Calibration Array Elements Sent In Descending Order(714)”
- A calibration correction flatness array was sent in descending order. The new array is rejected at the point of the descending element, X, which causes the array to be out of order. The previous elements, up to element X, are being written over by the new elements and cannot be restored.

**SCPI Error Messages**

- 715       –300, “Entered Password does not match the Security Password(715)”  
The user is trying to change the calibration security password and the verified password is incorrect as it does not match the system security password.
- 716       –300, “User-Defined Password must be a 5-numerical-digit(715)”  
The user is trying to change the calibration security password and the new password is not a 5-digit numerical entry.

---

## Loops Unlocked Errors

- 801       –300, “Device specific error;YIG Oscillator Unlock (801)”  
Phase lock with the YIG oscillator was lost or could not be achieved.
- 802       –300, “Device specific error;Reference Oscillator Unlock (802)”  
Phase lock with the Reference oscillator was lost or could not be achieved.
- 803       –300, “Device specific error;Fractional-N VCO Unlock (803)”  
Phase lock with the Fractional-N VCO was lost or could not be achieved.
- 804       –300, “Device specific error;Heterodyne Oscillator Unlock (804)”  
Phase lock with the Heterodyne oscillator was lost or could not be achieved.

---

## Miscellaneous Hardware Dependent Errors

901 –221, “Setting conflict;FNCW: Instrument Not In CW Mode (901)”

902 –300, “Device specific error;Need Same Attenuator Settings In Alt Sweep Mode (902)”

When using the Alternate Sweep feature, the attenuator settings must be the same. This prevents the attenuator from being continuously switched between two different attenuation values.

903 –300, “Device specific error;Bad Sweep Mode, Alternate Sweep Rejected (903)”

The instrument cannot sweep alternately with a stepped sweep as one of the sweep types.

904 –300, “Device specific error;Bad Magic Numbers in MM Head (904)”

The instrument will attempt to read known constants from predefined memory locations in the mm-wave source module NOVRAM (non-volatile RAM and ROM). An error condition occurs if the constants are not read back correctly, and the instrument reverts back to its stand-alone mode.

**SCPI Error Messages**

- 905       –300, “Device specific error;Bad Checksum in MM Head (905)”
- The error condition occurs when the checksum test fails on the mm-wave source module NOVRAM. If the error occurs at power up or instrument preset, instrument will revert back to stand-alone mode.
- 906       –300, “Device specific error;MM Head ALC Test Failed (906)”
- The test checks the overall integrity of the mm-wave source source module ALC circuitry at minimum settable power for the specific module.

---

**SCPI Conformance  
Information**

# SCPI Conformance Information

This chapter contains information pertaining to SCPI conformance.

# SCPI Conformance

The sweeper uses the SCPI language for GPIB communication. The SCPI commands and queries that the sweeper understands are listed and described individually in Chapter 2, “Programming Commands.” Table 5-1 lists all of the commands and queries that the sweeper understands and their status;

- SCPI approved
- SCPI confirmed
- Not part of the present SCPI 1992.0 definition
- IEEE 488.2 Required (non-SCPI command)
- IEEE 488.2 Optional (non-SCPI command)

## **NOTE**

In the table, if a command is terminated with a question mark enclosed in parentheses [(?)], that particular syntax is both a command and a query.

The SCPI version number that the sweeper supports at the writing of this manual is 1992.0

If you need more information about SCPI, refer to Chapter 1 “Getting Started Programming” or the *Beginner’s Guide to SCPI* (part number H2325-90001).

**Table 5-1. SCPI Conformance**

<b>Programming Command</b>	<b>Status</b>
*CLS	IEEE 488.2 Required
*DMC	IEEE 488.2 Optional
*EMC?	IEEE 488.2 Optional
*ESE ?	IEEE 488.2 Required
*ESR?	IEEE 488.2 Required
*GMC?	IEEE 488.2 Optional
*IDN?	IEEE 488.2 Required
*LMC?	IEEE 488.2 Optional
*LRN?	IEEE 488.2 Optional
*OPC ?	IEEE 488.2 Required
*OPT?	IEEE 488.2 Optional
*PMC	IEEE 488.2 Optional
*PSC ?	IEEE 488.2 Optional
*RCL	IEEE 488.2 Optional
*RMC	IEEE 488.2 Optional
*RST	IEEE 488.2 Required
*SAV	IEEE 488.2 Optional
*SRE ?	IEEE 488.2 Required
*STB?	IEEE 488.2 Required
*TRG?	IEEE 488.2 Optional
*TST?	IEEE 488.2 Required
*WAI	IEEE 488.2 Required



**Table 5-1. SCPI Conformance (continued)**

<b>Programming Command</b>	<b>Status</b>
ABORT	SCPI Confirmed
AM:STATe ?	SCPI Confirmed
AM:SOURce ?	SCPI Confirmed
CALibration:PEAKing[:EXECute] ?	Not part of the present SCPI 1992.0 definition
CALibration:TRACK	Not part of the present SCPI 1992.0 definition
CALibration:EXTerナル:FREQUency	Not part of the present SCPI 1992.0 definition
CALibration:EXTerナル:AMPLitude	Not part of the present SCPI 1992.0 definition
CALibration:EXTerナル:POINts?	Not part of the present SCPI 1992.0 definition
CALibration:EXTerナル:ZERO	Not part of the present SCPI 1992.0 definition
CALibration:EXTerナル:GAIN	Not part of the present SCPI 1992.0 definition
CALibration:EXTerナル:OFFset	Not part of the present SCPI 1992.0 definition
CALibration:PMEter:FLATness:INITiate?	Not part of the present SCPI 1992.0 definition
CALibration:PMEter:FLATness:NEXT?	Not part of the present SCPI 1992.0 definition
CORRection:FLATness:FREQ ?	Not part of the present SCPI 1992.0 definition
CORRection:FLATness:AMPL ?	Not part of the present SCPI 1992.0 definition
CORRection:FLATness:POINts ?	Not part of the present SCPI 1992.0 definition
CORRection:STATe ?	SCPI Confirmed
CORRection:VOLts:SCALE ?	Not part of the present SCPI 1992.0 definition
CORRection:VOLts:OFFSet ?	Not part of the present SCPI 1992.0 definition
DIAG:LRNS?	Not part of the present SCPI 1992.0 definition
DIAGnostic:TEST:FULLtest?	Not part of the present SCPI 1992.0 definition
DIAGnostic:TEST:FULLtest:REPort?	Not part of the present SCPI 1992.0 definition

**Table 5-1. SCPI Conformance (continued)**

<b>Programming Command</b>	<b>Status</b>
DISPlay[:STATE]?	SCPI Confirmed
FM:COUPling?	SCPI Confirmed
FM:STATe ?	SCPI Confirmed
FM:SENSitivity?	SCPI Confirmed
FM:SOURce ?	SCPI Confirmed
FREQuency:CENTer ?	SCPI Confirmed
FREQuency[:CW]:FIXed]	SCPI Confirmed
FREQuency[:CW] ?	SCPI Confirmed
FREQuency:FIXed)?	SCPI Confirmed
FREQuency[:CW] :AUTO  ?	SCPI Confirmed
FREQuency:FIXed] :AUTO  ?	SCPI Confirmed
FREQuency:MANual ?	SCPI Confirmed
FREQuency:MODE ?	SCPI Confirmed
FREQuency:MULTiplier ?	SCPI Confirmed
FREQuency:MODE SWCW?	Not part of the present SCPI 1992.0 definition
FREQuency:MULTiplier:STATe ?	Not part of the present SCPI 1992.0 definition
FREQuency:OFFSet ?	SCPI Confirmed
FREQuency:OFFSet:STATe ?	Not part of the present SCPI 1992.0 definition
FREQuency:SPAN ?	SCPI Confirmed
FREQuency:START ?	SCPI Confirmed
FREQuency:STEP [INCRement]  ?	Not part of the present SCPI 1992.0 definition
FREQuency:STOP ?	SCPI Confirmed

**Table 5-1. SCPI Conformance (continued)**

<b>Programming Command</b>	<b>Status</b>
INITiate:CONtinuous[?]	SCPI Confirmed
INITiate [:IMMediate]	SCPI Confirmed
MARKer [n]:AMPliTude[?]	SCPI Confirmed
MARKer [n] :AOFF	Not part of the present SCPI 1992.0 definition
MARKer [n] :FREQUENCY[?]	Not part of the present SCPI 1992.0 definition
MARKer [n] :MODE[?]	SCPI Confirmed
MARKer [n] :REFERENCE[?]	SCPI Confirmed
MARKer [n] :[:STATE][?]	SCPI Confirmed
OUTPut :STATE[?]	Not part of the present SCPI 1992.0 definition
OUTPut:IMPedance?	Not part of the present SCPI 1992.0 definition
POWer:ALC:CFACtor[?]	Not part of the present SCPI 1992.0 definition
POWer:ALC:SOURce[?]	SCPI Confirmed
POWer:ALC [STATE][?]	SCPI Confirmed
POWer:ATTeNuation[?]	SCPI Confirmed
POWer:ATTeNuation:AUTO[?]	SCPI Confirmed
POWer:CENter[?]	SCPI Confirmed
POWer:[LEVel][?]	SCPI Confirmed
POWer:MODE[?]	SCPI Confirmed
POWer:OFFSet[?][?]	Not part of the present SCPI 1992.0 definition
POWer:OFFSet:STATE[?]	Not part of the present SCPI 1992.0 definition
POWer:SLOPe[?]	Not part of the present SCPI 1992.0 definition
POWer:SLOPe:STATE[?]	Not part of the present SCPI 1992.0 definition

**Table 5-1. SCPI Conformance (continued)**

<b>Programming Command</b>	<b>Status</b>
POWer:SPAN ?	SCPI Confirmed
POWer:STARt ?	SCPI Confirmed
POWer:STATe ?	Not part of the present SCPI 1992.0 definition
POWer:STEP ?	Not part of the present SCPI 1992.0 definition
POWer:STOP ?	SCPI Confirmed
PULSE:PERiod ?	SCPI Confirmed
PULSE:FREQuency ?	Not part of the present SCPI 1992.0 definition
PULSE:WIDTh ?	SCPI Confirmed
PULM:SOURce ?	SCPI Confirmed
PULM:STATe ?	SCPI Confirmed
ROSCillator:SOURce ?	SCPI Confirmed
ROSCillator:SOURce:AUTO ?	SCPI Confirmed
STAtus:OPERation:CONDition?	SCPI Confirmed
STAtus:OPERation:ENABle  ?	SCPI Confirmed
STAtus:OPERation [:EVENT]?	SCPI Confirmed
STAtus:OPERation:NTRansition ?	SCPI Confirmed
STAtus:OPERation:PTRansition ?	SCPI Confirmed
STAtus:PRESet?	SCPI Confirmed
STAtus:QUESTionable:Condition?	SCPI Confirmed
STAtus:QUESTionable:ENABle ?	SCPI Confirmed
STAtus:QUESTionable:EVENt?	SCPI Confirmed
STAtus:QUESTionable:NTRansition ?	SCPI Confirmed

**Table 5-1. SCPI Conformance (continued)**

<b>Programming Command</b>	<b>Status</b>
STATus:QUES tionable:PTRransition ?	SCPI Confirmed
SWEEP:CONTRol:TYPE ?	Not part of the present SCPI 1992.0 definition
SWEEP:DWEL1 ?	SCPI Confirmed
SWEEP:DWEL1:AUTO ?	SCPI Confirmed
SWEEP:POINts ?	SCPI Confirmed
SWEEP:POWer:STEP ?	Not part of the present SCPI 1992.0 definition
SWEEP[:FREQuency]:STEP ?	Not part of the present SCPI 1992.0 definition
SWEEP:TIME ?	SCPI Confirmed
SWEEP:TIME:AUTO ?	SCPI Confirmed
SWEEP:TIME:LIMit ?	SCPI Confirmed
SWEEP:GENeration ?	SCPI Confirmed
SWEEP:MODE ?	SCPI Confirmed
SWEEP:MANual [RELative] ?	Not part of the present SCPI 1992.0 definition
SWEEP:MANual:POINt ?	Not part of the present SCPI 1992.0 definition
SWEEP:MARKer:STATe ?	Not part of the present SCPI 1992.0 definition
SWEEP:MARKer:XFER	Not part of the present SCPI 1992.0 definition
SWEEP[:POINts]:TRIGger:SOURce ? ?	Not part of the present SCPI 1992.0 definition
SWEEP[:POINts]:TRIGger:[IMMediate]	Not part of the present SCPI 1992.0 definition
SYSTem:ALTerNate ?	SCPI Confirmed
SYSTem:ALTerNate:STATe ?	SCPI Confirmed
SYSTem:COMMunicate:GPIB:ADDRes	SCPI Confirmed

Table 5-1. SCPI Conformance (continued)

Programming Command	Status
SYSTem:COMMunicate:PMEter:ADDRess ?	Not part of the present SCPI 1992.0 definition
SYSTem:COMMunicate:PMEter:TYPE ?	Not part of the present SCPI 1992.0 definition
SYSTem:ERRor?	SCPI Confirmed
SYSTem:KEY ?	SCPI Confirmed
SYSTem:KEY:DISable ?	Not part of the present SCPI 1992.0 definition
SYSTem:KEY:ENABle ?	Not part of the present SCPI 1992.0 definition
SYSTem:LANGuage ?	SCPI Confirmed
SYSTem:PRESet	SCPI Confirmed
SYSTem:PRESet [:EXECute]	Not part of the present SCPI 1992.0 definition
SYSTem:PRESet:SAVE	Not part of the present SCPI 1992.0 definition
SYSTem:PRESet:TYPE ?	Not part of the present SCPI 1992.0 definition
SYSTem:VERSion?	SCPI Confirmed
TRIGger[:IMMEdiate]	SCPI Confirmed
TRIGger:SOURce ?	SCPI Confirmed
TSWEEP	Not part of the present SCPI 1992.0 definition

**————** Index

# Index

- A** ABORt
  - command defined, 1-84
  - abort statement, 1-6
  - angle brackets, 1-16
  
- B** bits
  - in general status register model, 1-69
  - summary bit in general status register model, 1-70Boolean parameters
  - discussed in detail, 1-44
  - explained briefly, 1-31brackets, angle, 1-16BUS
  - trigger source defined, 1-84
  
- C** calibration commands, 2-14clear statement, 1-9colon
  - examples using, 1-23
  - proper use of, 1-22, 1-23, 1-44
  - types of command where used, 1-20command errors, 4-8command examples, 1-16commands, 1-36
  - calibration, 2-14
  - common, 1-19
  - correction, 2-16
  - defined, 1-15
  - diagnostic, 2-20
  - display, 2-22
  - event, 1-26
  - FM, 2-23
  - frequency, 2-25
  - IEEE 488.2 common commands, 2-4
  - implied, 1-26
  - marker, 2-37
  - memory, 2-43
  - output, 2-44
  - power, 2-45
  - pulse, 2-55
  - query, 1-26



- status, 2-59
- subsystem, 1-19, 1-20, 2-12
- sweep, 2-64
- syntax, 2-3
- syntax overview, 1-38, 1-39
- system, 2-76
- trigger, 2-34, 2-86
- command statements, fundamentals, 1-5
- command tables
  - how to read, 1-25
  - how to use, 1-24
- command trees
  - defined, 1-21
  - how to change paths, 1-21
  - how to read, 1-21
  - simplified example, 1-24
  - using efficiently, 1-23
- commas
  - proper use of, 1-22, 1-40
- common commands, 1-19, 1-22
  - defined, 1-19
- compatible language
  - 8350B, 3-1
- condition register, 1-69
- controller
  - defined, 1-15
- controller, definition of, 1-3
- correcton commands, 2-16
- current path
  - defined, 1-21
  - rules for setting, 1-21

**D** data types
 

- explained briefly, 1-29
- definitions of terms, 1-15
- device enter statement, 1-12
- device output statement, 1-10
- device-specific errors, 4-20
- diagnostic commands, 2-20
- discrete parameters
  - discussed in detail, 1-44
  - explained briefly, 1-31
- discrete response data
  - discussed in detail, 1-46
- display commands, 2-22

- E enable register, 1-70
  - in general status register model, 1-69
- END, 1-16
- END[end], 1-37
- enter statement, 1-12
- EOI, 1-16, 1-37
- EOI, suppression of, 1-12
- error/event queue, 4-3
- error message
  - action required, 4-6
  - detailed description, 4-6
  - manual error number, 4-6
  - SCPI error message, 4-6
  - SCPI error number, 4-6
- error message format, 4-6
- error numbers, 4-4
- errors
  - permanent, 4-2
- event commands, 1-26
- event register, 1-70
  - in general status register model, 1-69, 1-70
- events
  - event commands, 1-26
- example program
  - flatness correction, 1-64
  - GPIB check, 1-51
  - local lockout, 1-52
  - looping and synchronization, 1-60
  - setting up a sweep, 1-54
  - synchronous sweep, 1-62
  - use of queries, 1-55
  - use of save/recall, 1-58
- example programs, 1-47-67
- examples, simple program messages, 1-27
- example, stimulus response program, 1-33
- execution errors, 4-14
- extended numeric parameters
  - discussed in detail, 1-43
  - explained briefly, 1-30
- EXternal
  - trigger source defined, 1-84

- F** filter
  - transition, 1-70
  - flatness correction, example program, 1-64
  - FM commands, 2-23
  - forgiving listening, 1-19, 1-41
  - frequency commands, 2-25
  
- G** GP-IB check, example program, 1-51
  - Group Execute Trigger, 1-84
  
- H** 8350B
  - compatible language, 3-1
  - current active function code, 3-9
  - front panel key codes, 3-6
  - learn string, 3-5
  - mode string description, 3-6
  - replacement, 3-1
  - to 83750 Series syntax, 3-15
  - 83750 Series
    - to 8350B syntax, 3-15
  - GPIB
    - technical standard, 1-85
  - GPIB check, example program, 1-51
  - GPIB connecting cables, 1-3
  - GPIB, definition of, 1-2
  
- I** IEEE
  - mailing address, 1-85
  - IEEE 488.1
    - how to get a copy, 1-85
  - IEEE 488.2
    - how to get a copy, 1-85
  - IEEE 488.2 common commands, 2-4
  - IMMediate
    - trigger command defined, 1-84
    - trigger source defined, 1-84
  - implied commands, 1-26
  - instruments
    - defined, 1-15
  - integer response data
    - discussed in detail, 1-45
  - integers
    - rounding, 1-42

**L** listener, definition of, 1-3  
local lockout, example program, 1-52  
local lockout statement, 1-8  
local statement, 1-8  
looping and synchronization, example program, 1-60

**M** marker commands, 2-37  
memory commands, 2-43  
messages  
  details of program and response, 1-19  
  simple examples, 1-27  
message terminators  
  response message terminator defined, 1-40  
mnemonics, 1-15, 1-16  
  conventions for query commands, 1-15  
  long form, 1-16  
  short form, 1-16

**N** new line  
  affect on current path, 1-22  
  in response message terminator, 1-40  
  symbol used for, 1-16  
  use as a program message terminator, 1-17  
  use as a response message terminator, 1-17  
  with HTBasic **OUTPUT** statements, 1-37  
new line[new line]  
  use as a program message terminator, 1-37  
no errors, 4-5  
numeric parameters  
  discussed in detail, 1-42  
  explained briefly, 1-29

**O** \*OPC?  
  in example program, 1-35  
optional parameters, 1-26  
output commands, 2-44  
output statement, 1-10

**P** parameters  
  Boolean, 1-31, 1-44  
  discrete, 1-31, 1-44  
  extended numeric, 1-30, 1-43  
  numeric, 1-29, 1-42  
  optional, 1-26  
  types explained briefly, 1-29  
parser

- explained briefly, 1-21
- permanent errors, 4-2
- power commands, 2-45
- precise talking, 1-19, 1-41
- program and response messages, 1-19
- program example
  - flatness correction, 1-64
  - GPIB check, 1-51
  - local lockout, 1-52
  - looping and synchronization, 1-60
  - queries and response data, 1-55
  - save/recall, 1-58
  - setting up a sweep, 1-54
  - synchronous sweep, 1-62
- program examples, 1-47-67
- program message examples, 1-27
- program messages
  - defined, 1-15
- program message terminators
  - affect on current path, 1-22
  - defined, 1-37
  - syntax diagram, 1-37
  - use in examples, 1-17
- pulse commands, 2-55

**Q** queries

- defined, 1-15
- discussed, 1-19
- queries, example program, 1-55
- query commands, 1-26
  - query only, 1-26
- query errors, 4-22
- query only, 1-26
- questionable data status group, 1-76

**R** recall/save, example program, 1-58

- related documents, 1-14
- remote statement, 1-7
- response data
  - discrete, 1-46
  - integer, 1-45
- response data format, example program, 1-55
- response examples, 1-17
- response messages
  - defined, 1-15
  - discussed in detail, 1-36
  - syntax, 1-40
- response message terminators, 1-17

- defined, 1-40
- root
  - defined, 1-21
- root commands
  - defined, 1-21
- rounding, 1-42

**S** save/recall, example program, 1-58

- SCPI conformance information, 5-3
- SCPI conformance table, 5-3-10
- semicolon
  - examples using, 1-23
  - problems with input statements, 1-17
  - proper use of, 1-22, 1-23
- SOURce
  - trigger command defined, 1-84
- space
  - proper use of, 1-22
- standard event status group, 1-74
- standard notation, 1-16
- standard operation status group, 1-75
- status byte group, 1-72
- status commands, 2-59
- status registers
  - condition register, 1-69
  - enable register, 1-70
  - event register, 1-70
  - example sequence, 1-71
  - general model, 1-69
  - 83750 model, 1-72
  - transition filter, 1-70
- status register structure, SCPI, 1-78
- status register system programming example, 1-77
- status system
  - overview, 1-68
- stimulus response measurements
  - programming example, 1-33
- string response data
  - discussed in detail, 1-46
- subsystem commands, 1-19, 2-12
  - defined, 1-20
  - graphical tree format, 1-21
  - tabular format, 1-24
- summary bit, 1-70
- suppression of EOI, 1-12
- SWEep
  - simplified subsystem command tree, 1-24
- sweep commands, 2-64

- sweep, example program, 1-54
- synchronization, example program, 1-60
- synchronous sweep, example program, 1-62
- syntax diagrams
  - commands, 1-38, 1-39
  - message terminators, 1-37
  - program message, 1-37
  - response message, 1-40
- syntax drawings, 1-5
- synthesized sweeper status groups, 1-72
- system commands, 2-76

**T** tab

- proper use of, 1-22
- talker, definition of, 1-3
- terminators
  - program message, 1-17, 1-37
  - program message:use in examples, 1-17
  - response message, 1-17
- transition filter, 1-70
  - in general status register model, 1-69
- \*TRG- trigger command flowchart, 2-34
- trigger commands, 2-34, 2-86
  - defined, 1-84
- TRIGGER (HTBasic), 1-84
- trigger system
  - general programming model, 1-80

**U** user flatness correction commands, example program, 1-64

**W** \*WAI, use of example program, 1-62

- whitespace
  - proper use of, 1-22