HEWLETT PACKARD
Expanding Possibilities

HP 8163A/4A/6A Lightwave Series Mainframes

HP 8163A Lightwave Multimeter, HP 8164A Lightwave Measurement System, & HP 8166A Lightwave Multichannel System Programming Guide

E1299

Notices

Subject Matter

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

Hewlett-Packard makes no warranty of any kind with regard to this printed material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Printing History

New editions are complete revisions of the guide reflecting alterations in the functionality of the instrument. Updates are occasionally made to the guide between editions. The date on the title page changes when an updated guide is published. To find out the current revision of the guide, or to purchase an updated guide, contact your Hewlett-Packard representative.

Control Serial Number: First Edition applies directly to all instruments.

Warranty

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

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

HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, 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. Hewlett-Packard 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. Hewlett-Packard shall not be liable for any direct, indirect, special, incidental, or consequential damages whether based on contract, tort, or any other legal theory.

Assistance

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

Certification

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

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

ISO 9001 Certification

Produced to ISO 9001 international quality system standard as part of our objective of continually increasing customer satisfaction through improved process control.

08164-91016 E1299

First Edition:
E1299: December 1999

Second Edition
E0599: May 1999

In this Manual

This manual contains information about SCPI commands which can be used to program the following instruments:

- HP 8163A Lightwave Multimeter
- HP 8164A Lightwave Measurement System
- HP 8166A Lightwave Multichannel System

The Structure of this Manual

This manual is divided into 5 parts:

- *"Introduction to Programming" on page 13* gives a general introduction to SCPI programming with the HP 8163A Lightwave Multimeter, the HP 8164A Lightwave Measurement System, and the HP 8166A Lightwave Multichannel System.
- *"Specific Commands" on page 33* lists all instrument specific commands.
- *"Instrument Setup and Status" on page 43*, *"Measurement Operations & Settings" on page 63*, and *"Mass Storage, Display, and Print Functions" on page 121* give fuller explanations of all instrument specific commands.
- *"Programming Examples" on page 125* gives some example programs showing how the SCPI commands can be used with the HP 8163A Lightwave Multimeter, the HP 8164A Lightwave Measurement System, and the HP 8166A Lightwave Multichannel System.
- *"The HP 816x VXIplug&play Instrument Driver" on page 147*, *"GPIB Command Compatibility List" on page 163*, and *"Error Codes" on page 171* give information about the HP 816x VXI*plug&play* Instrument Driver, compatibility issues, and error codes.

Conventions used in this Manual

- All commands and typed text is written in Courier font, for example `INIT[:IMM]`.
- SCPI commands are written in mixed case: text that you MUST print is written in capitals; text which is helpful but nor necessary is written in lower case.

  So, the command `INITiate[:IMMediate]` can be entered either as `init[:imm]`, or as `initiate[:immediate]`. It does not matter whether you enter text using capitals or lower-case letters.
- SCPI commands often contain extra arguments in square brackets. These arguments may be helpful, but they need not be entered.

  So, the command `INITiate[:IMMediate]` can be entered as `init` or `initiate:imm`.

- A SCPI command which can be either a command or a query is appended with the text `/?`.

  So, `DISPlay:ENABle/?` refers to both the command `DISPlay:ENABle` and the query `DISPlay:ENABle?`.

Related Manuals

You can find more information about the instruments covered by this manual in the following manuals:

- *HP 8163A Lightwave Multimeter, HP 8164A Lightwave Measurement System, & HP 8166A Lightwave Multichannel System User's Guide* (HP Product Number 08164-91011).

NOTE Please note that User Guides no longer contain programming information, and must now be used in conjunction with this manual.

If you are not familiar with the General Purpose Interface Bus, GPIB, then refer to the following books:

- ANSI/IEEE-488.1-1978, *IEEE Standard Digital Interface for Programmable Instrumentation*, and ANSI/IEEE-488.2-1987, *IEEE Standard Codes, Formats, and Common Commands*, published by the Institute of Electrical and Electronic Engineers.

In addition, the commands not from the IEEE 488.2 standard are defined according to the Standard Commands for Programmable Instruments (SCPI). For an introduction to SCPI and SCPI programming techniques, refer to the following documents:

- Hewlett-Packard Press (Addison-Wesley Publishing Company, Inc.): *A Beginners Guide to SCPI* by Barry Eppler.
- The SCPI Consortium: *Standard Commands for Programmable Instruments*. To obtain a copy of this manual, contact the following address:

  SCPI Consortium Office
  Bode Enterprise
  2515 Camino del Rio South, Suite 340
  San Diego, CA, 92108
  USA

  Web: http://www.scpiconsortium.org

Contents

Figures

Tables

Introduction to Programming

This chapter gives general information on how to control your instrument remotely.

Descriptions for the actual commands for the instruments are given in the following chapters. The information in these chapters is specific to the HP 8163A Lightwave Multimeter, HP 8164A Lightwave Measurement System, and HP 8166A Lightwave Multichannel System and assumes that you are already familiar with programming the GPIB.

GPIB Interface

The interface used by your instrument is the GPIB (General Purpose Interface Bus).

GPIB is the interface used for communication between a controller and an external device, such as the tunable laser source. The GPIB conforms to IEEE standard 488-1978, ANSI standard MC 1.1 and IEC recommendation 625-1.

If you are not familiar with the GPIB, then refer to the following books:

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

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

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

In addition, the commands not from the IEEE-488.2 standard, are defined according to the Standard Commands for Programmable Instruments (SCPI).

For an introduction to SCPI, and SCPI programming techniques, please refer to the following documents:

- Hewlett-Packard Press (Addison-Wesley Publishing Company, Inc.). *A Beginners Guide to SCPI*. Barry Eppler. 1991.
- The SCPI Consortium: *Standard Commands for Programmable Instruments*. To obtain a copy of this manual, contact the following address:

  SCPI Consortium Office
  Bode Enterprise
  2515 Camino del Rio South, Suite 340
  San Diego, CA, 92108
  USA

  Web: http://www.scpiconsortium.org

The interface of the HP 8163A Lightwave Multimeter and of the HP 8164A Lightwave Measurement System to the GPIB is defined by the IEEE Standards 488.1 and 488.2.

Table 1 shows the interface functional subset that the instruments implement.

Mnemonic	Function
SH1	Complete source handshake capability
AH1	Complete acceptor handshake capability
T6	Basic talker; serial poll; unaddressed to talk if addressed to listen
L4	Basic listener; unaddressed to listen if addressed to talk; no listen only
SR1	Complete service request capability
RL1	Complete remote/local capability
PP0	No parallel poll capability
DC1	Device clear capability
DT0	No device trigger capability
C0	No controller capability (Controller capability to be implemented)

Table 1 GPIB Capabilities

Setting the GPIB Address

There are two ways to set the GPIB address:

- You can set the GPIB address by using the command *":SYSTem:COMMunicate:GPIB[:SELF]:ADDRess" on page 62.*
- You can set the GPIB address from the front panel. See your instrument's *User's Guide* for more information.

The default GPIB address is 20.

Returning the Instrument to Local Control

If the instrument is in remote control, a screen resembling Figure 1 will appear. Press [Local] if you wish to return the instrument to local control.

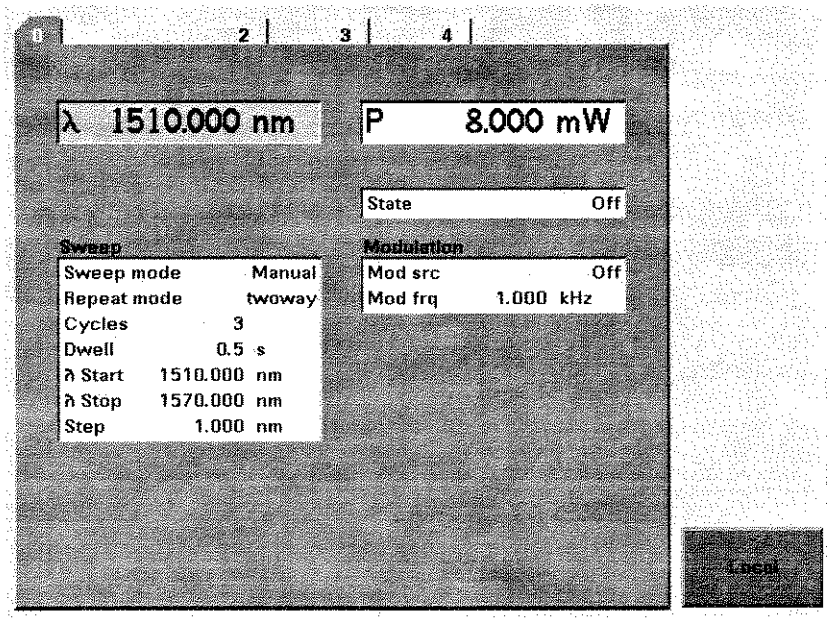


Figure 1 Remote Control

Message Queues

The instrument exchanges messages using an input and an output queue. Error messages are kept in a separate error queue.

How the Input Queue Works

The input queue is a FIFO queue (first-in first-out). Incoming bytes are stored in the input queue as follows:

1 Receiving a byte:

- Clears the output queue.
- Clears Bit 7 (MSB).

2 No modification is made inside strings or binary blocks. Outside strings and binary blocks, the following modifications are made:

- Lower-case characters are converted to upper-case.
- The characters 00_{16} to 09_{16} and $0B_{16}$ to $1F_{16}$ are converted to spaces (20_{16}).
- Two or more blanks are truncated to one.

3 An EOI (End Or Identify) sent with any character is put into the input queue as the character followed by a line feed (LF, $0A_{16}$). If EOI is sent with a LF, only one LF is put into the input queue.

4 The parser starts if the LF character is received or if the input queue is full.

Clearing the Input Queue

Switching the power off, or sending a Device Interface Clear signal, causes commands that are in the input queue, but have not been executed to be lost.

The Output Queue

The output queue contains responses to query messages. The instrument transmits any data from the output queue when a controller addresses the instrument as a talker.

Each response message ends with a carriage return (CR, $0D_{16}$) and a LF ($0A_{16}$), with EOI=TRUE. If no query is received, or if the query has an error, the output queue remains empty.

The Message Available bit (MAV, bit 4) is set in the Status Byte register whenever there is data in the output queue.

The Error Queue

The error queue is 30 errors long. It is a FIFO queue (first-in first-out). That is, the first error read is the oldest error to have occurred. A new error is only put into the queue if it is not already in it.

If more than 29 errors are put into the queue, the message:

```
-350 <Queue Overflow>
```

is placed as the last message in the queue.

Programming and Syntax Diagram Conventions

A program message is a message containing commands or queries that you send to the instruments. The following are a few points about program messages:

- You can use either upper-case or lower-case characters.
- You can send several commands in a single message. Each command must be separated from the next one by a semicolon (;).
- A command message is ended by a line feed character (LF) or <CR><LF>.
- You can use any valid number/unit combination.

  In other words, `1500NM`, `1.5UM` and `1.5E-6M` are all equivalent.

  If you do not specify a unit, then the default unit is assumed. The default unit for the commands are given with command description in the next chapter.

Short Form and Long Form

The instrument accepts messages in short or long forms.

For example, the message

```
:STATUS:OPERATION:ENABLE 768
```

is in long form.

The short form of this message is

```
:STAT:OPER:ENAB 768
```

In this manual, the messages are written in a combination of upper and lower case. Upper case characters are used for the short form of the message.

For example, the above command would be written

```
:STATus:OPERation:ENABle
```

The first colon can be left out for the first command or query in your message. That is, the example given above could also be sent as

```
STAT:OPER:ENAB 768
```

Command and Query Syntax

All characters not between angled brackets must be sent exactly as shown.

The characters between angled brackets (< . . . >) indicate the kind of data that you should send, or that you get in a response. You do not type the angled brackets in the actual message.

Descriptions of these items follow the syntax description. The following types of data are most commonly used:

string	is ascii data. A string is contained between double quotes (`"..."`) or single quotes (`'...'`).
value	is numeric data in integer (`12`), decimal (`34.5`) or exponential format (`67.8E-9`).
wsp	is a white space.

Other kinds of data are described as required.

The characters between square brackets (`[...]`) show optional information that you can include with the message.

The bar (`|`) shows an either-or choice of data, for example, *a|b* means either *a* or *b*, but not both simultaneously.

Extra spaces are ignored, so spaces can be inserted to improve readability.

Units

Where units are given with a command, usually only the base units are specified. The full sets of units are given in the table below.

Unit	Default	Allowed Mnemonics
meters	M	PM, NM, UM, MM, M
decibel	DB	MDB, DB
second	S	NS, US, MS, S
decibel/1mW	DBM	MDBM, DBM
Hertz	HZ	HZ, KHZ, MHZ, GHZ, THZ
Watt	Watt	PW, NW, UW, MW, Watt
meters per second	M/S	NM/S, UM/S, MM/S, M/S

Table 2 Units and allowed Mnemonics

Data Types

With the commands you give parameters to the instrument and receive response values from the instrument. Unless explicitly specified these data are given in ASCII format. The following types of data are used:

- ***Boolean*** data may only have the values 0 or 1.
- ***Integer*** range is given for each individual command.
- ***Float*** variables may be given in decimal or exponential writing (0.123 or 123E-3).
  All ***Float*** values conform to the 32 bit IEEE Standard, that is, all ***Float*** values are returned as 32-bit real values.
- A ***string*** is contained between double quotes (`"..."`) or single quotes (`'...'`). When the instrument returns a string, it is always included in " " and terminated by `<END>`.

- When a ***register*** value is given or returned (for example *ESE), the ***decimal*** values for the single bits are added. For example, a value of nine means that bit 0 and bit 3 are set.
- Larger blocks of data are given as ***Binary Blocks***, preceded by "#<*H*><*Len*><*Block*>", terminated by <END>; <*H*> represents the number of digits, <*Len*> represents the number of bytes, and <*Block*> is the data block. For example, for a ***Binary Block*** with 1 digit and 6 bytes this is: #16TRACES<END>.

Slot and Channel Numbers

Each module is identified by a slot number and a channel number. For commands that require you to specify a channel, the slot number is represented by [*n*] in a command and the channel number is represented by [*m*].

The slot number represents the module's position in the mainframe. These are:

- from one to two for the HP 8163A,
- from zero to four for the HP 8164A, and
- from one to seventeen for the HP 8166A.

These numbers are displayed on the front panel beside each module slot.

NOTE The HP 8164A slot for back-loadable tunable laser modules is numbered zero.

Channel numbers apply to modules that have two inputs/outputs, for example, the HP 81635A Dual Power Sensor.

Modules with two channels, for example, the HP 81635A Dual Power Sensor, use the channel number to distinguish between these channels.

NOTE The channel number of single channel modules is always one.

For example, if you want to query slot 1, channel 2 with the command, *":SENSe[n]:[CHANnel[m]]:POWer:WAVelength?" on page 83*, you should send the command:

- :sens1:chan2:pow:wav?

NOTE If you do not specify a slot or channel number, the lowest possible number is used as the default value. This means:

- Slot 1 for the HP 8163A and HP 8166A mainframes.
- Slot 0 for the HP 8164A mainframe.
- Channel 1 for all channels.

Laser Selection Numbers

The laser selection number, [*l*], identifies the upper or lower wavelength laser source for dual wavelength Laser Source modules and Return Loss modules with two internal laser sources. The lower wavelength source is denoted by *1*. The upper wavelength source is denoted by *2*.

NOTE For Return Loss modules, *0* denotes the use of an external laser source as the input to your Return Loss module for the following commands:

- *":SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:FPDelta[l]" on page 85*,
- *":SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:FPDelta[l]?" on page 85*,
- *":SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[l]" on page 85*, and
- *":SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[l]?" on page 86*.

Common Commands

The IEEE 488.2 standard has a list of reserved commands, called common commands. Some of these commands must be implemented by any instrument using the standard, others are optional.

Your instrument implements all the necessary commands, and some optional ones. This section describes the implemented commands.

Common Command Summary

Table 3 gives a summary of the common commands.

Command	Parameter	Function	Page
*CLS		Clear Status Command	page 45
*ESE		Standard Event Status Enable Command	page 45
*ESE?		Standard Event Status Enable Query	page 46
*ESR?		Standard Event Status Register Query	page 46
*IDN?		Identification Query	page 46
*OPC		Operation Complete Command	page 47
*OPC?		Operation Complete Query	page 47
*OPT?		Options Query	page 47
*RST		Reset Command	page 48
*STB?		Read Status Byte Query	page 48
*TST?		Self Test Query	page 49
*WAI		Wait Command	page 49

Table 3 Common Command Summary

NOTE **These commands are described in more detail in *"IEEE-Common Commands" on page 45.***

Common Status Information

There are three registers for the status information. Two of these are status-registers and one is an enable-registers. These registers conform to the IEEE Standard 488.2-1987. You can find further descriptions of these registers under `*ESE`, `*ESR?`, and `*STB?`.

Figure 2 shows how the Standard Event Status Enable Mask (SESEM) and the Standard Event Status Register (SESR) determine the Event Status Bit (ESB) of the Status Byte.

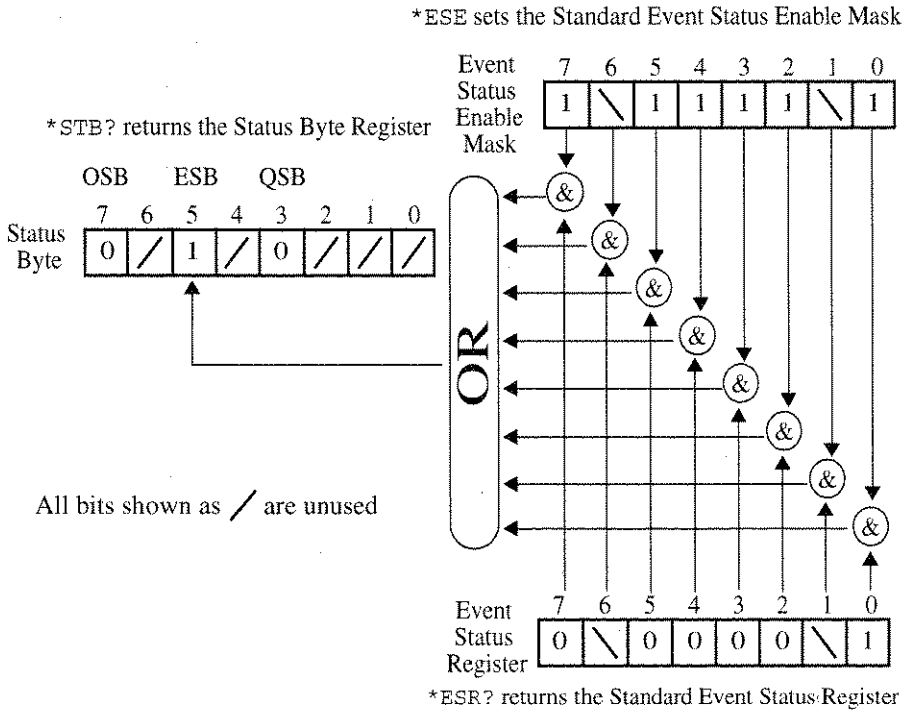


Figure 2 The Event Status Bit

The SESR contains the information about events that are not slot specific. For details of the function of each bit of the SESR, see *"Standard Event Status Register" on page 29.*

The SESEM allows you to choose the event that may affect the ESB of the Status Byte. If you set a bit of the SESEM to zero, the corresponding event cannot affect the ESB. The default is for all the bits of the SESEM to be set to 0.

The questionable and operation status systems set the Operational Status Bit (OSB) and the Questionable Status Bit (QSB). These status systems are described in *"The Status Model" on page 25* and *"Status Reporting – The STATus Subsystem" on page 50.*

NOTE Unused bits in any of the registers change to 0 when you read them.

The Status Model

Status Registers

Each node of the status circuitry has three registers:

- A condition register (CONDition), which contains the current status. This register is updated continuously. It is not changed by having its contents read.
- The event register (EVENt), which contains details of any positive transitions in the corresponding condition register, that is, when a bit changes from 0 → 1. The contents of this register are cleared when it is read. The contents of any higher-level registers are affected with regard to the appropriate bit.
- The enable register (ENABle), which enables changes in the event register to affect the next stage of registers.

NOTE The event register is the only kind of register that can affect the next stage of registers.

The structures of the Operational and Questionable Status Systems are similar. Figure 4 describe how the Questionable Status Bit (QSB) and the Operational Status Bit (OSB) of the Status Byte Register are determined.

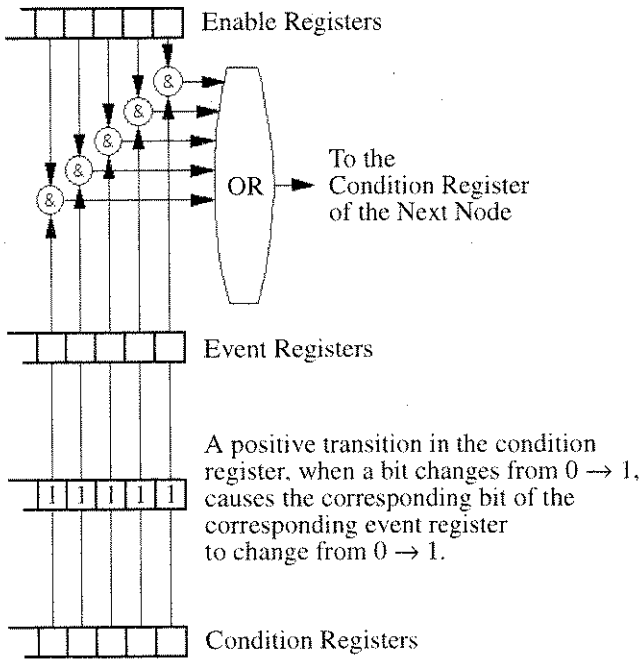


Figure 3 The Registers and Filters for a Node

The Operational/Questionable Slot Status Event Register (OSSER/QSSER) contains the status of a particular module slot. A bit changes from 0 → 1 when an event occurs, for example, when a laser is switched on. For details of the function of each bit of these registers, see *"Operation/Questionable Status Summary Register" on page 29* and *"Operation/Questionable Status Summary Register" on page 29*.

The Operational/Questionable Slot Enable Status Mask (OSESM/QSESM) allows you to choose the events for each module slot that may affect the Operational/Questionable Status Event Register (see below). If you set a bit of the OSESM/QSESM to zero, the occurence of the corresponding event for this particular module slot cannot affect the Operational/Questionable Status Event Register. The default is for all the bits of the OSESM/QSESM to be set to 0.

The Operational/Questionable Status Event Summary Register (OSESR/QSESR) summarizes the status of every module slot of your instrument. If, for any slot, any bit of the QSSER goes from 0 → 1 AND the corresponding bit of the QSSEM is 1at the same time, the QSESR bit representing that slot is set to 1.

The Operational/Questionable Status Enable Summary Mask (OSESM/QSESM) allows you to choose the module slots that may affect the OSB/QSB of the Status Byte. If any bit of the QSESR goes from 0 → 1 AND the corresponding bit of the QSESM is 1at the same time, the QSB of the Status Byte is set to 1. If you set a bit of the OSESM/QSESM to zero, the corresponding module slot cannot affect the OSB/QSB. The default is for all the bits of the OSESM/QSESM to be set to 0.

The Operational/Questionable Status Enable Summary Mask for the HP 8163A Lightwave Multimeter and the HP 8164A Lightwave Measurement System consists of one level. These are described in *"The Operational/Questionable Status System for HP 8163A & HP 8164A" on page 27*.

As the HP 8166A Lightwave Multichannel System has 17 module slots, the Operational/Questionable Status Enable Summary Mask consists of two levels. This is described in *"Status System for HP 8166A" on page 27*.

Status System for HP 8163A & HP 8164A

The status system for the HP 8163A Lightwave Multimeter and the HP 8164A Lightwave Measurement System returns the status of 2 and 5 module slots respectively. The Operational/Questionable Status Summary Registers consist of

one level and are described by Figure 4. Any commands that require LEVel*1* do not apply to these mainframes.

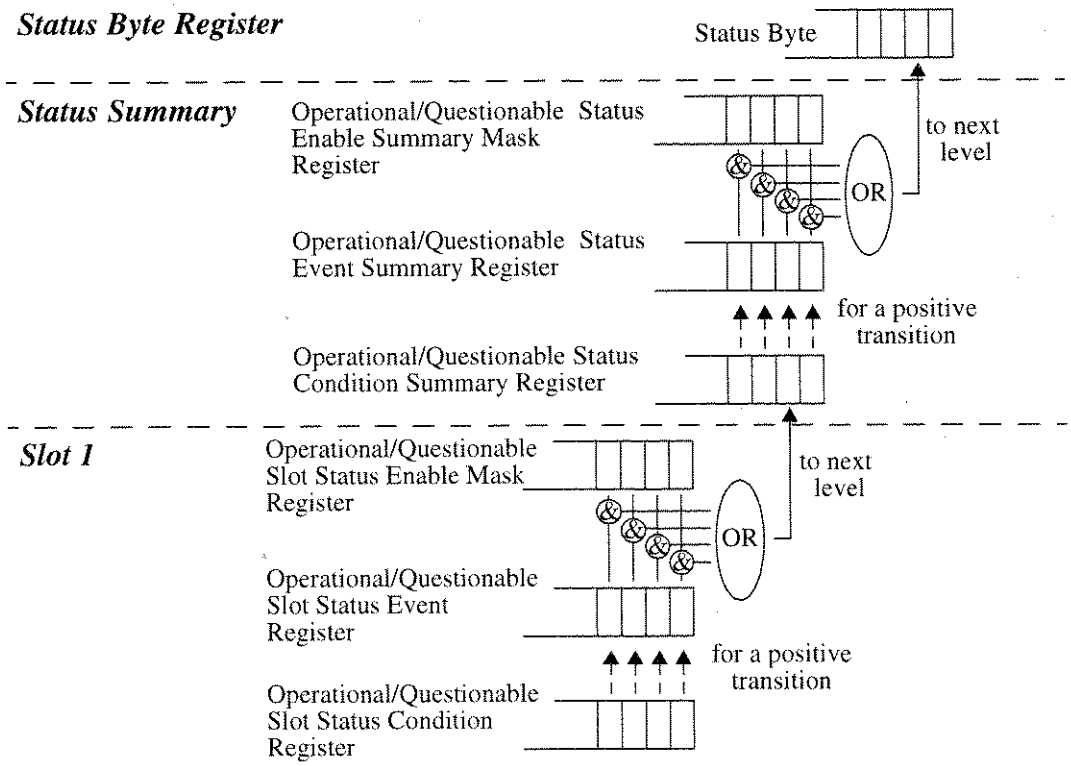


Figure 4 The Operational/Questionable Status System for HP 8163A & HP 8164A

Status System for HP 8166A

The status system for the HP 8166A Lightwave Multichannel System returns the status of 17 module slots. The Operational/Questionable Status Summary Registers consists of two levels, as described by Figure 5.

Module slots 1 to 14 affect the Level 0 summary register as described in Figure 4. Bit 0 of the Level 0 summary registers represents the summary of the status of

module slots 15, 16, and 17. The Level 1 summary registers contain an individual summary for each of these module slots.

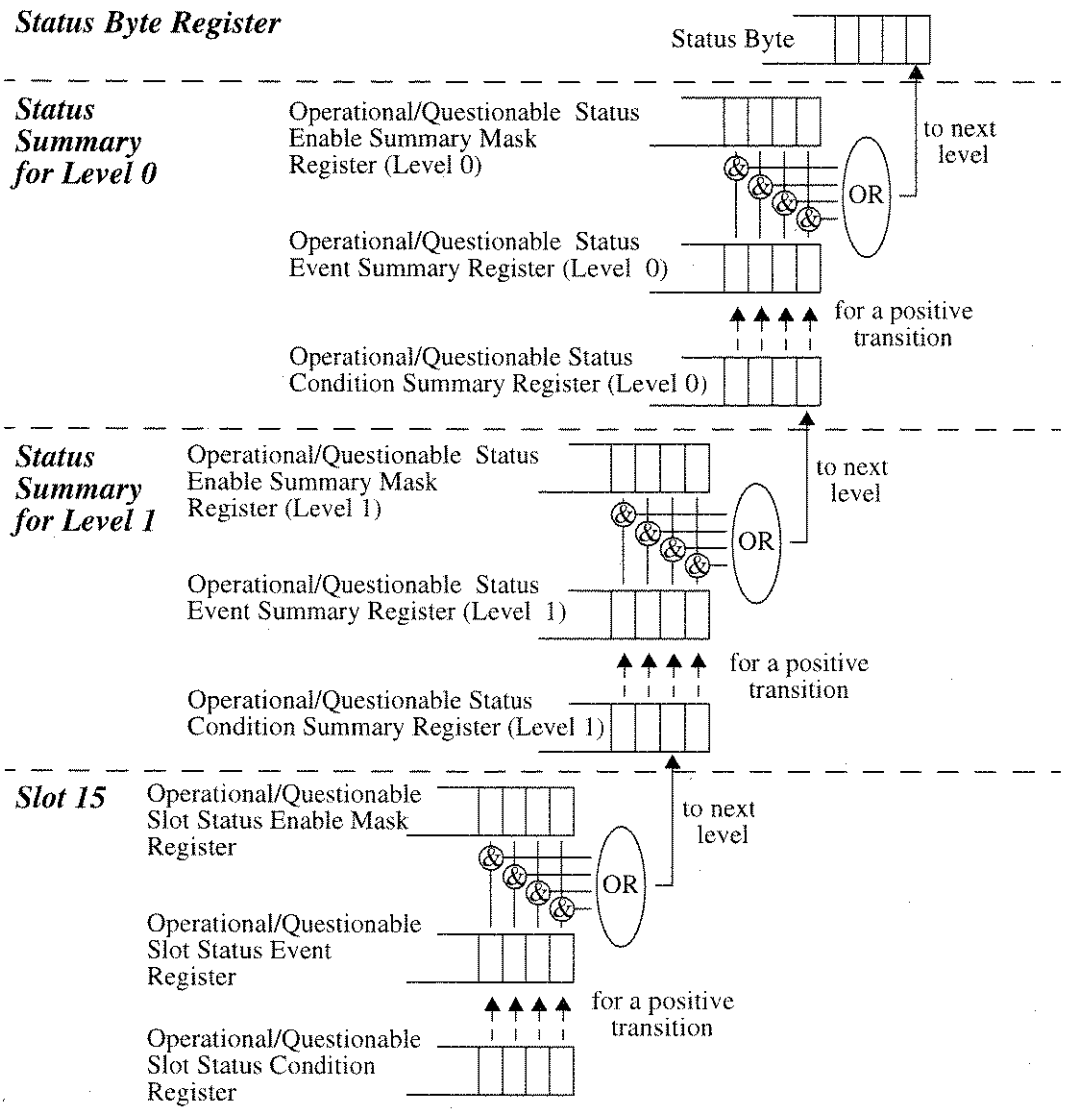


Figure 5 The Operational/Questionable Status System for HP 8166A

Annotations

Status Byte Register

- Bit 3, the QSB, is built from the questionable event status register and its enable mask.
- Bit 5, the ESB, is built from the SESR and its SESEM.
- Bit 7, the OSB, is built from the operation event status register and its enable mask.
- All other bits are unused, and therefore set to 0.

Standard Event Status Register

- Bit 0 is set if an operation complete event has been received since the last call to `*ESR?`.
- Bit 1 is always 0 (no service request).
- Bit 2 is set if a query error has been detected.
- Bit 3 is set if a device dependent error has been detected.
- Bit 4 is set if an execution error has been detected.
- Bit 5 is set if a command error has been detected.
- Bit 6 is always 0 (no service request).
- Bit 7 is set for the first call of `*ESR?` after Power On.

Operation/Questionable Status Summary

- The Operation/Questionable Status Summary consist of a condition and an event register.
- A "rising" bit in the condition register is copied to the event register.
- A "falling" bit in the condition register has no effect on the event register.
- Reading the condition register is non-destructive.
- Reading the event register is destructive.
- A summary of the event register and its enable mask is set in the status byte.

Operation/Questionable Status Summary Register

- Bits 0 to 4 are built from the OSSER/QSSER and the OSSEM/QSSEM.
- A summary of the event register, the condition register and the enable mask is set in the status byte.

Operation/Questionable Slot Status

- The Operation/Questionable Slot Status consist of a condition and an event register.
- A "rising" bit in the condition register is copied to the event register.
- A "falling" bit in the condition register has no effect on the event register.
- Reading the condition register is non-destructive.
- Reading the event register is destructive.
- A summary of the event register, the condition register and the enable mask is set in the status byte.

Operation Slot Status Register

- Bit 0 is set if the laser is switched on.
- Bit 1 is set if the Coherence Control is switched on.
- Bit 3 is set if Power Meter zeroing or Tunable Laser module lambda zeroing is ongoing.
- All other bits are unused, and therefore set to 0.

Questionable Slot Status Register

- Bit 0 is set if excessive power is set by the user for any source module or if excessive averaging time is set for any Power Meter.
- Bit 1 is set if the last Power Meter zeroing or Tunable Laser module lambda zeroing failed.
- Bit 2 is set if temperature is out of range.
- Bit 3 is set if laser protection is switched on.
- Bit 4 is set if the module has not settled.
- Bit 5 is set if the module is out of specifications.
- Bit 6 is set if ARA is recommended.
- Bit 7 is set if the duty cycle is out of range.
- All other bits are unused, and therefore set to 0.

Status Command Summary

*STB?	returns status byte, value 0 .. +255
*ESE	sets the standard event status enable mask, parameter 0 .. +255
*ESE?	returns SESE, value 0 .. +255
*ESR?	returns the standard event status register, value 0 .. +255
*OPC	parses all program message units in the message queue.
*OPC?	returns 1 if all operations (scan trace printout, measurement) are completed. Otherwise it returns 0.
*CLS	clears the status byte and SESR, and removes any entries from the error queue.
*RST	clears the error queue, loads the default setting, and restarts communication. **NOTE:** *RST does NOT touch the STB or SESR. A running measurement is stopped.
*TST?	initiates an instrument selftest and returns the results as a 32 bit LONG.

Other Commands

*OPT?	returns the installed modules and the slots these modules are installed in: For example, `*OPT?` → 81682A, 81533B, 81532A, , Modules 81682A, 81533B, and 81532A are installed in slots 0 to 2 respectively. Slots 3 and 4 are empty.
*WAI	prevents the instrument from executing any further commands until the current command has finished executing. All pending operations are completed during the wait period.
*IDN?	identifies the instrument; returns the manufacturer, instrument model number, serial number, and firmware revision level.

Specific Commands

This chapter lists all the instrument specific commands relating to the HP 8163A Lightwave Multimeter and the HP 8164A Lightwave Measurement System, with a single-line description.

Each of these summaries contains a page reference for more detailed information about the particular command later in this manual.

Specific Command Summary

The commands are ordered in a command tree. Every command belongs to a node in this tree.

The root nodes are also called the subsystems. A subsystem contains all commands belonging to a specific topic. In a subsystem there may be further subnodes.

All the nodes have to be given with a command. For example in the command `disp:brig`

- `DISPlay` is the subsystem containing all commands for controlling the display.
- `BRIGhtness` is the command selecting brightness.

NOTE If a command and a query are both available, the command ends `/?`. So, `disp:brig/?` means that `disp:brig` and `disp:brig?` are both available.

Table 4 gives an overview of the command tree. You see the nodes, the subnodes, and the included commands.

Command	Description	Page
:DISPlay		
:BRIGhtness/?	Changes or queries the current display brightness.	123
:CONTrast/?	Changes or queries the current display contrast.	123
:ENABle/?	Switches the display on or off or queries whether the display is on or off.	124
:FETCh[*n*][:CHANnel[*m*]][:SCALar]		
:POWer[:DC]?	Returns a power value from a sensor.	70
:RETurnloss?	Returns the current return loss value.	70
:RETurnloss?	Returns a return loss value.	70
:INITiate[*n*]:[CHANnel[*m*]]		
[:IMMediate]	Starts a measurement.	70
:CONTinuous/?	Starts or Queries a single/continuous measurement.	71
:LOCK/?	Switches the lock on/off or returns the current state of the lock.	65
:OUTPut[*n*][:CHANnel[*m*]]		

Table 4 Specific Command Summary

Command	Description	Page
:CONNection/?	Selects or returns Analog Output parameter.	86
:PATH/?	Sets or returns the regulated path.	87
[:STATe]/?	Sets a source's output terminals to open or closed or returns the current status of a source's output terminals.	87
:READ[*n*][:CHANnel[*m*]][:SCALar]		
:POWer[:DC]?	Reads the current power value from a sensor.	71
:RETurnloss?	Reads the current return loss value.	72
:RETurnloss?	Returns the current return loss value.	72
:SENSe[*n*][:CHANnel[*m*]]:CORRection		
[:LOSS][:INPut][:MAGNitude]/?	Sets or returns the value of correction data for a sensor.	72
:COLLECT:ZERO	Executes a zero calibration of a sensor module.	73
:COLLECT:ZERO?	Returns the current zero state of a sensor module.	73
:COLLECT:ZERO:ALL	Executes a zero calibration of all sensor modules.	73
:SENSe[*n*][:CHANnel[m]]:FUNCtion		
:PARameter:LOGGing/?	Sets or returns the number of samples and the averaging time, t_{avg}, for logging.	74
:PARameter:MINMax/?	Sets or returns the minmax mode and the window size.	75
:PARameter:STABility/?	Sets or returns the total time, delay time and the averaging time, t_{avg}, for stability.	76
:RESult?	Returns the data array of the last function.	77
:STATe/?	Enables/disables the function mode or returns whether the function mode is enabled.	77
:THReshold/?	Sets or returns the threshold value and the start mode.	78
:SENSe[*n*][:CHANnel[*m*]]:POWer		
:ATIMe/?	Sets or returns the average time of a sensor.	78
:RANGe[:UPPer]/?	Sets or returns the most positive signal entry expected for a sensor.	79
:RANGe:AUTO/?	Sets or returns the range of a sensor to produce the most dynamic range without overloading.	79
:REFerence/?	Sets or returns the reference level of a sensor.	80
UNIT/?	Sets or returns the units used for absolute readings on a sensor.	82
:WAVelength/?	Sets or returns the wavelength for a sensor.	83
:SENSe[*n*][:CHANnel[*m*]]:POWer:REFerence		

Table 4 Specific Command Summary (continued)

Command	Description	Page
:DISPlay	Sets the reference level for a sensor from the input power level.	81
:STATe/?	Sets or returns whether sensor results are in relative or absolute units.	81
:STATe:RATio/?	Sets or returns whether sensor results are displayed relative to a channel or to an absolute reference.	82
:SENSe[*n*][:CHANnel[*m*]]:RETurnloss:CALibration		
:FACTory		84
:REFLectance		84
:TERMination		84
:SENSe[*n*][:CHANnel[*m*]]:RETurnloss:CORRection		
:FPDelta[*l*]/?	Sets or returns	85
:REFLectance[*l*]/?	Sets or returns	85
:SENSe[*n*][:CHANnel[*m*]]:RETurnloss:CALibration		
:FACTory	Selects the factory-set calibration values	84
:COLLect:REFLectance	Sets the Reflection Reference calibration values to the values currently measured by the chosen return loss module	84
:COLLect:TERMination	Sets the Reflection Reference calibration values to the values currently measured by the chosen return loss module	84
:SENSe[*n*][:CHANnel[*m*]]:RETurnloss:CORRection		
:FPDelta[n]/?	Sets or returns the front panel delta, that is, the loss variation value due to the front panel connector	85
:REFLectance[n]/?	Sets or returns Return Loss Reference, the return loss value of your reference reflector.	85
:SLOT[*n*]		
:EMPTy?	Returns whether the module slot is empty.	65
:IDN?	Returns information about the module.	66
:OPTions?	Returns the module's options.	66
:TST?	Returns the latest selftest results for a module.	66
:SLOT[*n*][:HEAD[*m*]]		
:EMPTy?	Returns whether an optical head is connected.	67
:IDN?	Returns information about the optical head.	67
:OPTions?	Returns the optical head's options.	67
:TST?	Returns the latest selftest results for an optical head.	68

Table 4 Specific Command Summary (continued)

Command	Description	Page
[:SOURce[*n*]][:CHANnel[*m*]]		
:MODout/?	Returns the mode of the modulation output.	91
[:SOURce[*n*]][:CHANnel[*m*]:]AM		
[:INTernal]:FREQuency[*l*]/?	Sets or returns the frequency of an internal signal source.	88
:SOURce[*l*]/?	Sets or returns a source for the modulating system.	89
:STATe[*l*]/?	Turns Amplitude Modulation of a source on or off or returns whether Amplitude Modulation is on or off.	90
[:SOURce[*n*]][:CHANnel[*m*]:]POWer		
[:LEVel][:IMMediate][:AMPLitude]	Sets the laser output power of a source.	94
[:LEVel][:IMMediate][:AMPLitude[*l*]]?	Returns the laser output power of a source.	95
[:LEVel]:RISetime/?	Sets or returns the laser rise time of a source.	95
:ATTenuation[*l*]/?	Sets or returns the attenuation level for a source.	92
:STATe/?	Sets or returns the state of the source output signal.	96
:UNIT/?	Sets or returns the power units.	96
:WAVelength/?	Sets or returns the wavelength source of a dual-wavelength source.	99
[:SOURce[*n*]][:CHANnel[*m*]:]POWer:ATTenuation		
:AUTO/?	Selects Automatic or Manual Attenuation Mode for a source or returns the selected mode.	93
:DARK/?	Enables/disables 'dark' position on a source or returns whether 'dark' position is active for a source.	93
[:SOURce[*n*]][:CHANnel[*m*]:]READout		
:DATA?	Returns number of datapoints returned by the [:SOURce[n]][:CHANnel[m]]:READout:POINts? command.	98
:POINts?	Returns the data as a binary stream from either a lambda logging operation or the maximum power the laser can produce at each wavelength.	98
[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength		
[:CWl:FIXED]	Sets the absolute wavelength of a source.	99
[:CWl:FIXED[*l*]]?	Returns the absolute wavelength of a source.	99
:FREQuency/?	Sets the frequency difference used to calculate a relative wavelength for a source.	100
:REFerence?	Returns the reference wavelength of a source.	101
[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:CORRection		
:ARA	Realigns the laser cavity.	100

Table 4 Specific Command Summary (continued)

Command	Description	Page
:ZERO	Exexutes a wavelength zero.	100
[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:REFerence		
:DISPlay	Sets the reference wavelength of a source to the value of the output wavelength.	101
[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:SWEep		
:CYCLes/?	Sets or returns the number of cycles.	101
:DWELl/?	Sets or returns the dwell time.	102
:LLOGging/?	Switches lambda logging on or off or queries the state of lambda logging.	103
:MODE/?	Sets or returns the sweep mode.	103
:PMAX?	Returns the highest permissible power for a wavelength sweep.	104
:REPeat/?	Sets or returns the repeat mode.	104
:SPEed/?	Sets or returns the speed for continuous sweeping.	105
:STARt/?	Sets or returns the start point of the sweep.	105
:STOP/?	Sets or returns the end point of the sweep.	106
[:STATe]/?	Stops, starts, pauses or continues a wavelength sweep or returns the the state of a sweep.	106
[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:SWEep:STEP		
:NEXT	Performs the next sweep step.	107
:PREVious	Performs the previous sweep step again.	107
[:WIDTh]/?	Sets or returns the width of the sweep step.	107
:SPECial		
:REBoot	Reboots the mainframe and all modules.	68
:STATus[*n*]		
:PRESet	Presets all Enable Registers.	54
:STATus:OPERation		
[:EVENt]?	Returns the Operational Status Event Summary Register.	50
[:EVENt]:LEVel*l*?	Returns the Operational Status Event Summary Register for slots 15 - 17 of the HP 8166A Lightwave Multichannel System.	52
:CONDition?	Returns the Operational Status Condition Summary Register.	51
:CONDition:LEVel*l*?	Returns the Operational Status Condition Summary Register for slots 15 - 17 of the HP 8166A Lightwave Multichannel System.	52

Table 4 Specific Command Summary (continued)

Command	Description	Page
:ENABle/?	Sets or queries the Operational Status Enable Summary Mask.	51
:ENABle:LEVel*1*/?	Sets or queries the Operational Status Enable Summary Mask for slots 15 - 17 of the HP 8166A Lightwave Multichannel System.	52
:STATus*n*:OPERation		
[:EVENt]?	Returns the Operational Slot Status Event Register for slot *n*.	53
:CONDition?	Returns the Operational Slot Status Condition Register for slot *n*.	53
:ENABle/?	Sets or queries the Operation Slot Status Enable Mask for slot *n*.	54
:STATus:QUEStionable		
[:EVENt]?	Returns the Questionable Status Event Summary Register.	55
[:EVENt]:LEVel*1*?	Returns the Questionable Status Event Summary Register for slots 15 - 17 of the HP 8166A Lightwave Multichannel System.	57
:CONDition?	Returns the Questionable Status Condition Summary Register.	56
:CONDition:LEVel*1*?	Returns the Questionable Status Condition Summary Register for slots 15 - 17 of the HP 8166A Lightwave Multichannel System.	57
:ENABle/?	Sets or queries the Questionable Status Enable Summary Mask.	56
:ENABle:LEVel*1*/?	Sets or queries the Questionable Status Enable Summary Mask for slots 15 - 17 of the HP 8166A Lightwave Multichannel System.	57
:STATus*n*:QUEStionable		
[:EVENt]?	Returns the Questionable Slot Status Event Register for slot *n*.	58
:CONDition?	Returns the Questionable Slot Status Condition Register for slot *n*.	58
:ENABle/?	Sets or queries the Questionable Slot Status Enable Mask for slot *n*.	59
:SYSTem		
:DATE/?	Sets or returns the instrument's internal date.	59
:ERRor?	Returns the contents of the instrument's error queue.	60
:HELP:HEADers?	Returns a list of GPIB commands.	60
:PRESet	Sets all parameters to their default values.	61
:TIME/?	Sets or returns the instrument's internal time.	61

Table 4 Specific Command Summary (continued)

Command	Description	Page
:VERSion?	Returns the instrument's SCPI version.	61
:SYSTem:COMMunicate:GPIB		
[:SELF]:ADDress/?	Sets or returns the GPIB address.	62
:TRIGger	Generates a hardware trigger.	109, 115
:CONFiguration/?	Sets or returns trigger configuration.	113
:TRIGger:CONFiguration		
:EXTended/?	Sets or returns extended trigger configuration.	115
:FPEDal/?	Enables/disables the Input Trigger connector to be triggered using a Foot Pedal or returns whether the Input Trigger connector can be triggered using a Foot Pedal.	115
:TRIGger[*n*][CHANnel[*m*]]		
:INPut/?	Sets or returns the incoming trigger response .	110
:OUTPut/?	Sets or returns the outgoing trigger response.	112

Table 4 Specific Command Summary (continued)

Instrument Setup and Status

This chapter gives descriptions of commands that you can use when setting up your instrument. The commands are split into the following separate subsytems:

- IEEE specific commands that were introduced in *"Common Commands" on page 22*.
- `STATus` subsystem commands that relate to the status model.
- `SYSTem` subsystem commands that control the serial interface and internal data.

IEEE-Common Commands

"Common Commands" on page 22 gave a brief introduction to the IEEE-common commands which can be used with the instruments. This section gives fuller descriptions of each of these commands.

command: ***CLS**

syntax: *CLS

description: The CLear Status command *CLS clears the following:

- Error queue
- Standard event status register (SESR)
- Status byte register (STB)

After the `*CLS` command the instrument is left waiting for the next command. The instrument setting is unaltered by the command, although `*OPC`/`*OPC?` actions are cancelled.

parameters: none

response: none

example: `*CLS`

command: ***ESE**

syntax: *ESE<wsp><value>

$0 \leq value \leq 255$

description: The standard Event Status Enable command (`*ESE`) sets bits in the Standard Event Status Enable Mask (SESEM) that enable the corresponding bits in the standard event status register (SESR).

The register is cleared:

- at power-on,
- by sending a value of zero.

The register is not changed by the `*RST` and `*CLS` commands.

parameters: The bit value for the register (a ***16-bit signed integer*** value):

Bit	Mnemonic	Decimal Value
7 (MSB)	Power On	128
6	Not Used	0
5	Command Error	32
4	Execution Error	16
3	Device Dependent Error	8
2	Query Error	4
1	Not Used	0
0 (LSB)	Operation Complete	1

response: none

example: `*ESE 21`

command: ***ESE?**

syntax: *ESE?

description: The standard Event Status Enable query *ESE? returns the contents of the Standard Event Status Enable Mask (see *ESE for information on this register).

parameters: none

response: The bit value for the register (a ***16-bit signed integer*** value).

example: `*ESE? → 21<END>`

command: ***ESR?**

syntax: *ESR?

description: The standard Event Status Register query *ESR? returns the contents of the Standard Event Status Register. The register is cleared after being read.

parameters none

response The bit value for the register (a ***16-bit signed integer*** value):

Bit	Mnemonic	Decimal Value
7 (MSB)	Power On	128
6	Not used	0
5	Command Error	32
4	Execution Error	16
3	Device Dependent Error	8
2	Query Error	4
1	Not used	0
0 (LSB)	Operation Complete	1

example: `*ESR? → 21<END>`

command: ***IDN?**

syntax: *IDN?

description: The IDeNtification query *IDN? gets the instrument identification over the interface.

parameters: none

response: The identification terminated by <END>:

For example.

`HEWLETT-PACKARD`	manufacturer
mmmm	instrument model number (for example 8164A)
sssssss	serial number
rrrrrrrrr	firmware revision level

NOTE The HP 8163A, HP8164A, and HP8166A will always return HEWLETT-PACKARD as the manufacturer. This will not be affected by the transition of these instruments to Agilent Technologies. This will allow programs that use this string to continue functioning.

See *":SLOT[n]:HEAD[n]:IDN?" on page 67* for information on module identity strings.

example: `*IDN? → HEWLETT-PACKARD,mmmm,sssssss,rrrrrrrrr<END>`

command: ***OPC**

syntax: *OPC

description: The instrument parses and executes all program message units in the input queue and sets the operation complete bit in the standard event status register (SESR). This command can be used to avoid filling the input queue before the previous commands have finished executing.

The following actions cancel the *OPC command (and put the instrument into Operation Complete, Command Idle State):

- Power-on
- the Device Clear Active State is asserted on the interface.
- *CLS
- *RST

parameters: none

response: none

example: `*OPC`

command: ***OPC?**

syntax: *OPC?

description: The OPeration Complete query *OPC? parses all program message units in the input queue, sets the operation complete bit in the Standard Event Status register, and places an ASCII '1' in the output queue, when the contents of the input queue have been processed.

The following actions cancel the *OPC? query (and put the instrument into Operation Complete, Command Idle State):

- Power-on
- the Device Clear Active State is asserted on the interface.
- *CLS
- *RST

parameters: none

response: `1<END>` is always returned.

example: `*OPC? → 1<END>`

command: ***OPT?**

syntax: *OPT?

description: The OPTions query *OPT? returns the modules installed in your instrument.

parameters: none

response: Returns the part number of all installed modules, separated by commas.

Slots are listed starting with the lowest slot number, that is, slot 0 for the HP 8164A and Slot 1 for the HP 8163A and HP 8166A.

If any slot is empty or not recognised, two spaces are inserted instead of the module's part number. See the example below, where slots 1 and 4 are empty.

example: `*OPT? → 81682A , , 81533B, 81532A, <END>`

command: ***RST**

syntax: *RST

description: The ReSeT command *RST sets the mainframe and all modules to the reset setting (standard setting) stored internally.

Pending *OPC? actions are cancelled.

The instrument is placed in the idle state awaiting a command. The *RST command clears the error queue.

The *RST command is equivalent to the `*CLS` command AND the `syst:preset` command.

The following are not changed:

- GPIB (interface) state
- Instrument interface address
- Output queue
- Service request enable register (SRE)
- Standard Event Status Enable Mask (SESEM)

parameters: none

response: none

example: `*RST`

command: ***STB?**

syntax: *STB?

description: The STatus Byte query *STB? returns the contents of the Status Byte register.

parameters: none

response: The bit value for the register (a ***16-bit signed integer*** value):

Bit	Mnemonic	Decimal Value
7 (MSB)	Operation Status	128
6	Not used	0
5	Event Status Bit	32
4	Not used	0
3	Questionable Status	8
2	Not used	0
1	Not used	0
0	Not used	0

example: `*STB? → 128<END>`

command: ***TST?**

syntax: *TST?

description: The self-TeST query *TST? makes the instrument perform a self-test and place the results of the test in the output queue. If the self-test fails, the results are also put in the error queue. We recommend that you read self-test results from the error queue. No further commands are allowed while the test is running. After the self-test the instrument is returned to the setting that was active at the time the self-test query was processed. The self-test does not require operator interaction beyond sending the `*TST?` query.

parameters: none

response: The sum of the results for the individual tests (a ***32-bit signed integer*** value, where $0 \leq$ *value* ≤ 4294967296):

Bits	Mnemonic	Decimal Value
31	Selftest failed on Mainframe	A negative value
18 - 30	Not used	0
17	Selftest failed on Slot 17	131072
16	Selftest failed on Slot 16	65536
15	Selftest failed on Slot 15	32768
14	Selftest failed on Slot 14	16384
13	Selftest failed on Slot 13	8192
12	Selftest failed on Slot 12	4096
11	Selftest failed on Slot 11	2048
10	Selftest failed on Slot 10	1024
9	Selftest failed on Slot 9	512
8	Selftest failed on Slot 8	256
7	Selftest failed on Slot 7	128
6	Selftest failed on Slot 6	64
5	Selftest failed on Slot 5	32
4	Selftest failed on Slot 4	16
3	Selftest failed on Slot 3	8
2	Selftest failed on Slot 2	4
1	Selftest failed on Slot 1	2
0	Selftest failed on Slot 0	1

If 16 is returned, the module in slot 4 has failed.
If 18 is returned, the modules in slots 1 and 4 have failed.
A value of zero indicates no errors.

example: `*TST? → 0<END>`

command: ***WAI**

syntax: *WAI

description: The WAIt command prevents the instrument from executing any further commands until the current command has finished executing. All pending operations are completed during the wait period.

parameters: none

response: none

example: `*WAI`

Status Reporting – The STATus Subsystem

The Status subsystem allows you to return and set details from the Status Model. For more details, see *"The Status Model" on page 25*.

command: **:STATus:OPERation[:EVENt][:LEVel*0*]?**

syntax: :STATus:OPERation[:EVENt][:LEVel*0*]?

description: Returns the Operational Status Event Summary Register (OSESR).

parameters: none

response: The sum of the results for the slots (a ***16-bit signed integer*** value, where $0 \leq value \leq 32767$):

Bits	Mnemonics			Decimal Value
	HP 8163A	**HP 8164A**	**HP 8166A**	
15	Not used	Not used	Not used	0
14	Not used	Not used	Slot 14 Summary	16384
13	Not used	Not used	Slot 13 Summary	8192
12	Not used	Not used	Slot 12 Summary	4096
11	Not used	Not used	Slot 11 Summary	2048
10	Not used	Not used	Slot 10 Summary	1024
9	Not used	Not used	Slot 9 Summary	512
8	Not used	Not used	Slot 8 Summary	256
7	Not used	Not used	Slot 7 Summary	128
6	Not used	Not used	Slot 6 Summary	64
5	Not used	Not used	Slot 5 Summary	32
4	Not used	Slot 4 Summary	Slot 4 Summary	16
3	Not used	Slot 3 Summary	Slot 3 Summary	8
2	Slot 2 Summary	Slot 2 Summary	Slot 2 Summary	4
1	Slot 1 Summary	Slot 1 Summary	Slot 1 Summary	2
0	Not used	Slot 0 Summary	Level 1 Summary	1

example: `stat:oper? → +0<END>`

command: **:STATus:OPERation:CONDition[:LEVel*0*]?**

syntax: :STATus:OPERation:CONDition[:LEVel*0*]?

description: Reads the Operational Status Condition Summary Register.

parameters: none

response: The sum of the results for the individual slots (a ***16-bit signed integer*** value, where $0 \leq value \leq 32767$):

Bits	Mnemonics			Decimal Value
	HP 8163A	HP 8164A	HP 8166A	
15	Not used	Not used	Not used	0
14	Not used	Not used	Slot 14 Summary	16384
13	Not used	Not used	Slot 13 Summary	8192
12	Not used	Not used	Slot 12 Summary	4096
11	Not used	Not used	Slot 11 Summary	2048
10	Not used	Not used	Slot 10 Summary	1024
9	Not used	Not used	Slot 9 Summary	512
8	Not used	Not used	Slot 8 Summary	256
7	Not used	Not used	Slot 7 Summary	128
6	Not used	Not used	Slot 6 Summary	64
5	Not used	Not used	Slot 5 Summary	32
4	Not used	Slot 4 Summary	Slot 4 Summary	16
3	Not used	Slot 3 Summary	Slot 3 Summary	8
2	Slot 2 Summary	Slot 2 Summary	Slot 2 Summary	4
1	Slot 1 Summary	Slot 1 Summary	Slot 1 Summary	2
0	Not used	Slot 0 Summary	Level 1 Summary	1

example: `stat:oper:cond? → +0<END>`

command: **:STATus:OPERation:ENABle[:LEVel*0*]**

syntax: :STATus:OPERation:ENABle[:LEVel*0*]<wsp><value>

description: Sets the bits in the Operational Status Enable Summary Mask (OSESM) that enable the contents of the OSESR to affect the Status Byte (STB).

Setting a bit in this register to 1 enables the corresponding bit in the OSESR to affect bit 7 of the Status Byte.

parameters: The bit value for the OSESM as a ***16-bit signed integer*** value (0 .. +32767)

The default value is 0.

response: none

example: `stat:oper:enab 128`

command: **:STATus:OPERation:ENABle[:LEVel*0*]?**

syntax: :STATus:OPERation:ENABle[:LEVel*0*]?

description: Returns the OSESM for the OSESR

parameters: none

response: The bit value for the operation enable mask as a ***16-bit signed integer*** value (0 .. +32767)

example: `stat:oper:enab? → +128<END>`

command: **:STATus:OPERation[:EVENt]:LEVel*l*?**

syntax: :STATus:OPERation[:EVENt]:LEVel*l*?

description: Returns the Operational Status Event Summary Register (OSESR) for slots 15 to 17 of the HP 8166A Lightwave Multichannel System.

parameters: none

response: The sum of the results for the slots (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bits	Mnemonics HP 8166A	Decimal Value
15-4	Not used	0
3	Slot 17 Summary	8
2	Slot 16 Summary	4
1	Slot 15 Summary	2
0	Not used	0

example: `stat:oper:level1? → +0<END>`

command: **:STATus:OPERation:CONDition:LEVel*l*?**

syntax: :STATus:OPERation:CONDition:LEVel*l*?

description: Returns the Operational Status Condition Summary Register for slots 15 to 17 of the HP 8166A Lightwave Multichannel System.

parameters: none

response: The sum of the results for slots 15 to 17 (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bits	Mnemonics HP 8166A	Decimal Value
15-4	Not used	0
3	Slot 17 Summary	8
2	Slot 16 Summary	4
1	Slot 15 Summary	2
0	Not used	0

example: `stat:oper:cond:level1? → +0<END>`

command: **:STATus:OPERation:ENABle:LEVel*l***

syntax: :STATus:OPERation:ENABle:LEVel*l*<wsp><value>

description: Sets the bits in the Operational Status Enable Summary Mask (OSESM) that enable the contents of the OSESR for slots 15 - 17 of the HP 8166A Lightwave Measurement System to affect the Status Byte (STB).

Setting a bit in this register to 1 enables the corresponding bit in the OSESR for slots 15 - 17 of the HP 8166A Lightwave Measurement System to affect bit 7 of the Status Byte.

parameters: The bit value for the OSESM as a ***16-bit signed integer*** value (0 .. +32767)

The default value is 0.

response: none

example: `stat:oper:enab:level1 128`

command: **:STATus:OPERation:ENABle:LEVel*l*?**

syntax: :STATus:OPERation:ENABle:LEVel*l*?

description: Returns the OSESM for the OSESR for slots 15 - 17 of the HP 8166A Lightwave Measurement System

parameters: none

response: The bit value for the operation enable mask as a ***16-bit signed integer*** value (0 .. +32767)

example: `stat:oper:enab:level1? → +128<END>`

command: **:STATus*n*:OPERation[:EVENt]?**

syntax: :STATus*n*:OPERation[:EVENt]?

description: Returns the Operational Slot Status Event Register (OSSER) of slot *n*.

parameters: none

response: The results for the individual slot events (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bit	Mnemonic	Decimal Value
4-15	Not used	0
3	Slot *n*: Zeroing started	8
2	Not used	0
1	Slot *n*: Coherence Control has been switched on	2
0	Slot *n*: Laser has been switched on	1

example: `stat1:oper? → +0<END>`

command: **:STATus*n*:OPERation:CONDition?**

syntax: :STATus*n*:OPERation:CONDition?

description: Returns the Operational Slot Status Condition Register of slot *n*.

parameters: none

response: The results for the individual slot events (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bit	Mnemonic	Decimal Value
4-15	Not used	0
3	Slot *n*: Zeroing ongoing	8
2	Not used	0
1	Slot *n*: Coherence Control is switched on	2
0	Slot *n*: Laser is switched on	1

example: `stat1:oper:cond? → +0<END>`

command: **:STATus*n*:OPERation:ENABle**

syntax: :STATus*n*:OPERation:ENABle<wsp><value>

description: Sets the bits in the Operation Slot Status Enable Mask (OSSEM) for slot *n* that enable the contents of the Operation Slot Status Event Register (OSSER) for slot *n* to affect the OS-ESR.

Setting a bit in this register to 1 enables the corresponding bit in the OSSER for slot *n* to affect bit *n* of the OSESR.

parameters: The bit value for the OSSEM as a ***16-bit signed integer*** value (0 .. +32767)

response: none

example:
```
stat:oper:enab 128
```

command: **:STATus*n*:OPERation:ENABle?**

syntax: :STATus*n*:OPERation:ENABle?

description: Returns the OSSEM of slot *n*

parameters: none

response: The bit value for the OSSEM as a ***16-bit signed integer*** value (0 .. +32767)

example:
```
stat:oper:enab? → +128<END>
```

command: **:STATus:PRESet**

syntax: :STATus:PRESet

description: Presets all bits in all the enable masks for both the OPERation and QUEStionable status systems to 0, that is, OSSEM, QSSEM, OSESM, and QSESM.

parameters: none

response: none

example:
```
stat:pres
```

command: **:STATus:QUEStionable[:EVENt][:LEVel*0*]?**

syntax: :STATus:QUEStionable[:EVENt][:LEVel*0*]?

description: Returns the Questionable Status Event Summary Register (QSESR).

parameters: none

response: The sum of the results for the QSESR as a ***16-bit signed integer*** value (0 .. +32767)

Bits	Mnemonics			Decimal Value
	HP 8163A	HP 8164A	HP 8166A	
15	Not used	Not used	Not used	0
14	Not used	Not used	Slot 14 Summary	16384
13	Not used	Not used	Slot 13 Summary	8192
12	Not used	Not used	Slot 12 Summary	4096
11	Not used	Not used	Slot 11 Summary	2048
10	Not used	Not used	Slot 10 Summary	1024
9	Not used	Not used	Slot 9 Summary	512
8	Not used	Not used	Slot 8 Summary	256
7	Not used	Not used	Slot 7 Summary	128
6	Not used	Not used	Slot 6 Summary	64
5	Not used	Not used	Slot 5 Summary	32
4	Not used	Slot 4 Summary	Slot 4 Summary	16
3	Not used	Slot 3 Summary	Slot 3 Summary	8
2	Slot 2 Summary	Slot 2 Summary	Slot 2 Summary	4
1	Slot 1 Summary	Slot 1 Summary	Slot 1 Summary	2
0	Not used	Slot 0 Summary	Level 1 Summary	1

example: `stat:ques? → +0<END>`

command: **:STATus:QUEStionable:CONDition[:LEVel*0*]?**

syntax: :STATus:QUEStionable:CONDition[:LEVel*0*]?

description: Returns the Questionable Status Condition Summary Register.

parameters: none

response: The sum of the results for the Questionable Status Condition Summary Register as a ***16-bit signed integer*** value (0 .. +32767)

Bits	Mnemonics			Decimal Value
	HP 8163A	HP 8164A	HP 8166A	
15	Not used	Not used	Not used	0
14	Not used	Not used	Slot 14 Summary	16384
13	Not used	Not used	Slot 13 Summary	8192
12	Not used	Not used	Slot 12 Summary	4096
11	Not used	Not used	Slot 11 Summary	2048
10	Not used	Not used	Slot 10 Summary	1024
9	Not used	Not used	Slot 9 Summary	512
8	Not used	Not used	Slot 8 Summary	256
7	Not used	Not used	Slot 7 Summary	128
6	Not used	Not used	Slot 6 Summary	64
5	Not used	Not used	Slot 5 Summary	32
4	Not used	Slot 4 Summary	Slot 4 Summary	16
3	Not used	Slot 3 Summary	Slot 3 Summary	8
2	Slot 2 Summary	Slot 2 Summary	Slot 2 Summary	4
1	Slot 1 Summary	Slot 1 Summary	Slot 1 Summary	2
0	Not used	Slot 0 Summary	Level 1 Summary	1

example: `stat:ques:cond? → +0<END>`

command: **:STATus:QUEStionable:ENABle[:LEVel*0*]**

syntax: :STATus:QUEStionable:ENABle[:LEVel*0*]<wsp><value>

description: Sets the bits in the Questionable Status Enable Summary Mask (QSESM) that enable the contents of the QSESR to affect the Status Byte (STB).

Setting a bit in this register to 1 enables the corresponding bit in the QSESR to affect bit 3 of the Status Byte.

parameters: The bit value for the questionable enable mask as a ***16-bit signed integer*** value (0 .. +32767)

The default value is 0.

response: none

example: `stat:ques:enab 128`

command: **:STATus:QUEStionable:ENABle[:LEVel*0*]?**

syntax: :STATus:QUEStionable:ENABle[:LEVel*0*]?

description: Returns the QSESM for the event register

parameters: none

response: The bit value for the QSEM as a ***16-bit signed integer*** value (0 .. +32767)

example: `stat:ques:enab? → +128<END>`

command: **:STATus:QUEStionable[:EVENt]:LEVel*l*?**

syntax: :STATus:QUEStionable[:EVENt]:LEVel*l*?

description: Returns the Questionable Status Event Summary Register (QSESR) for slots 15 to 17 of the HP 8166A Lightwave Multichannel System.

parameters: none

response: The sum of the results for the slots (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bits	Mnemonics HP 8166A	Decimal Value
15-4	Not used	0
3	Slot 17 Summary	8
2	Slot 16 Summary	4
1	Slot 15 Summary	2
0	Not used	0

example: `stat:ques:level1? → +0<END>`

command: **:STATus:QUEStionable:CONDition:LEVel*l*?**

syntax: :STATus:QUEStionable:CONDition:LEVel*l*?

description: Returns the Questionable Status Condition Summary Register for slots 15 to 17 of the HP 8166A Lightwave Multichannel System.

parameters: none

response: The sum of the results for the slots (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bits	Mnemonics HP 8166A	Decimal Value
15-4	Not used	0
3	Slot 17 Summary	8
2	Slot 16 Summary	4
1	Slot 15 Summary	2
0	Not used	0

example: `stat:ques:cond:level1? → +0<END>`

command: **:STATus:QUEStionable:ENABle:LEVel*l***

syntax: :STATus:QUEStionable:ENABle:LEVel*l*<wsp><value>

description: Sets the bits in the Questionable Status Enable Summary Mask (QSESM) that enable the contents of the QSESR for slots 15 - 17 of the HP 8166A Lightwave Measurement System to affect the Status Byte (STB).

Setting a bit in this register to 1 enables the corresponding bit in the OSESR for slots 15 - 17 of the HP 8166A Lightwave Measurement System to affect bit 7 of the Status Byte.

parameters: The bit value for the QSESM as a ***16-bit signed integer*** value (0 .. +32767)

The default value is 0.

response: none

example: `stat:oper:enab:level1 128`

command: **:STATus:QUEStionable:ENABle:LEVel*l*?**

syntax: :STATus:QUEStionable:ENABle:LEVel*l*?

description: Returns the QSESM for the QSESR for slots 15 - 17 of the HP 8166A Lightwave Measurement System

parameters: none

response: The bit value for the QSESM as a ***16-bit signed integer*** value (0 .. +32767)

example: `stat:oper:enab:level1? → +128<END>`

command: **:STATus*n*:QUEStionable[:EVENt]?**

syntax: :STATus*n*:QUEStionable[:EVENt]?

description: Returns the questionable status of slot *n* - the Questionable Slot Status Event Register (QSSER).

parameters: none

response: The results for the individual slot events (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bit	Mnemonic	Decimal Value
8-15	Not used	0
7	Slot *n*: Duty cycle has been out of range	128
6	Slot *n*: ARA has been recommended	64
5	Slot *n*: Module has been out of specification	32
4	Slot *n*: Module has settled unsuccessfully	16
3	Slot *n*: Laser protection has been on	8
2	Slot *n*: Temperature has been out of range	4
1	Slot *n*: A Zeroing operation has failed	2
0	Slot *n*: Excessive Value has occurred	1

Every *n*th bit is the summary of slot *n*.

example: `stat1:oper? → +0<END>`

command: **:STATus*n*:QUEStionable:CONDition?**

syntax: :STATus*n*:QUEStionable:CONDition?

description: Returns the Questionable Slot Status Condition Register for slot *n*.

parameters: none

response: The results for the individual slot events (a ***16-bit signed integer*** value, where 0 ≤ *value* ≤ 32767):

Bit	Mnemonic	Decimal Value
8-15	Not used	0
7	Slot *n*: Duty cycle is out of range	128
6	Slot *n*: ARA recommended	64
5	Slot *n*: Module is out of specification	32
4	Slot *n*: Module has not settled	16
3	Slot *n*: Laser protection on	8
2	Slot *n*: Temperature out of range	4
1	Slot *n*: Zeroing failed	2
0	Slot *n*: Excessive Value	1

Every *n*th bit is the summary of slot *n*.

example: `stat1:ques:cond? → +0<END>`

command: **:STATus*n*:QUEStionable:ENABle**

syntax: :STATus*n*:QUEStionable:ENABle<wsp><value>

description: Sets the bits in the Questionable Slot Status Enable Mask (QSSEM) for slot *n* that enable the contents of the Questionable Slot Status Register (QSSR) for slot *n* to affect the QS-ESR.

Setting a bit in this register to 1 enables the corresponding bit in the QSSER for slot *n* to affect bit *n* of the QSESR.

parameters: The bit value for the QSSEM as a ***16-bit signed integer*** value (0 .. +32767)

response: none

example:
```
stat:ques:enab 128
```

command: **:STATus*n*:QUEStionable:ENABle?**

syntax: :STATus*n*:QUEStionable:ENABle?

description: Returns the QSSEM for slot *n*

parameters: none

response: The bit value for the QSSEM as a ***16-bit signed integer*** value (0 .. +32767)

example:
```
stat:ques:enab? → +128<END>
```

Interface/Instrument Behaviour Settings – The SYSTem Subsystem

The SYSTem subsystem lets you control the instrument's serial interface. You can also control some internal data (like date, time, and so on).

command: **:SYSTem:DATE**

syntax: :SYSTem:DATE<wsp><year>,<month>,<day>

description: Sets the instrument's internal date.

parameters:
- the first value is the year (four digits),
- the second value is the month, and
- the third value is the day.

response: none

example:
```
syst:date 1999, 1, 12
```

command: **:SYSTem:DATE?**

syntax: :SYSTem:DATE?

description: Returns the instrument's internal date.

parameters: none

response: The date in the format year, month, day (***16-bit signed integer*** values)

example: `syst:date? → +1999,+1,+12<END>`

command: **:SYSTem:ERRor?**

syntax: :SYSTem:ERRor?

description: Returns the next error from the error queue (see *"The Error Queue" on page 18*).

Each error has the error code and a *short* description of the error, separated by a comma, for example `0, "No error"`.

Error codes are numbers in the range -32768 and +32767.

Negative error numbers are defined by the SCPI standard. Positive error numbers are device dependent.

parameters: none

response: The number of the latest error, and its meaning.

example: `syst:err? → -113,"Undefined header"<END>`

command: **:SYSTem:HELP:HEADers?**

syntax: :SYSTem:HELP:HEADers?

description: Returns a list of GPIB commands.

parameters: none

response: Returns a list of GPIB commands

example: syst:help:head? → *Returns a list of all GPIB commands*

command: **:SYSTem:PRESet**

syntax: :SYSTem:PRESet

description: Sets the mainframe and all installed modules to their standard settings. This command has the same function as the `Preset` hardkey.

The following are not affected by this command:

- the GPIB (interface) state,
- the backlight and contrast of the display,
- the interface address,
- the output and error queues,
- the Service Request Enable register (SRE),
- the Status Byte (STB),
- the Standard Event Status Enable Mask (SESEM), and
- the Standard Event Status Register (SESR).

parameters: none

response: none

example: SYST:PRES

command: **:SYSTem:TIME**

syntax: :SYSTem:TIME<wsp><hour>,<minute>,<second>

description: Sets the instrument's internal time.

parameters:
- the first value is the hour (0 .. 23),
- the second value is the minute, and
- the third value is the seconds.

response: none

example:
```
syst:time 20,15,30
```

command: **:SYSTem:TIME?**

syntax: :SYSTem:TIME?

description: Returns the instrument's internal time.

parameters: none

response: The time in the format hour, minute, second. Hours are counted 0...23 (***16-bit signed integer*** values).

example:
```
syst:time? → +20,+15,+30<END>
```

command: **:SYSTem:VERSion?**

syntax: :SYSTem:VERSion?

description: Returns the SCPI revision to which the instrument complies.

parameters: none

response: The revision year and number.

example:
```
syst:vers? → 1995.0<END>
```

command: **:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess**

syntax: :SYSTem:COMMunicate:GPIB[:SELF]:ADDRess<wsp><GPIB Address>

description: Sets the GPIB address.

parameters: The GPIB Address

response: none

example: SYST:COMM:GPIB:ADDR 20

command: **:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?**

syntax: :SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?

description: Returns the GPIB address.

parameters: none

response: The GPIB Address

example: SYST:COMM:GPIB:ADDR? → +20<END>

Measurement Operations & Settings

This chapter gives descriptions of commands that you can use when you are setting up or performing measurements. The commands are split up into the following subsystems:

- Root layer commands that take power measurements, configures triggering, and return information about the mainframe and it's slots
- `SENSe` subsystem commands that control Power Sensors, Optical Head Interface Modules, and Return Loss Modules.
- `SOURce` subsystem commands that control Laser Source modules, DFB source modules, Tunable Laser modules, and Return Loss Modules with internal laser sources.
- `TRIGger` subsystem commands that control triggering.

Root Layer Command

command: **:LOCK**

syntax: :LOCK<wsp><boolean>, <value>

description: Switches the lock off and on.
High power lasers cannot be switched on, if you switch the lock on. High power lasers are switched off immediately when you switch the lock on.

parameters: A **boolean** value: 0 or OFF: switch lock off
1 or ON: switch lock on
<value> is the four-figure lock password.

response: none

example: `lock 1,1234` - *1234 is the default password*

command: **:LOCK?**

syntax: :LOCK?

description: Returns the current state of the lock.

parameters: none

response: A ***boolean*** value: 0: lock is switched off
1: lock is switched on

example: `lock?` → `1<END>`

The commands in the Slot subsystem allow you to query the following:

- a particular slot, for example, using `slot1:empt?`,
- or, an Optical Head attached to an Optical Head Interface Module, for example, an Optical Head Interface Module in slot1 with an Optical Head attached to channel 2, using `slot1:head2:empt?`.

command: **:SLOT[*n*]:EMPTy?**

syntax: :SLOT[*n*]:EMPTy?

description: Returns whether the module slot is empty.

parameters: none

response: A ***boolean*** value: 0: there is a module in the slot
1: the module slot is empty

examples: `slot1:empt?` → `0<END>` There is a module in slot1

affects: Independent of module type

command: **:SLOT[*n*]:IDN?**

syntax: :SLOT[*n*]:IDN?

description: Returns information about the module.

parameters: none

response:

HEWLETT-PACKARD:	manufacturer
mmmm:	instrument model number (for example 81533B)
ssssssss:	serial number
rrrrrrrrr:	date of firmware revision

example:
```
slot1:idn? →
HEWLETT-PACKARD, 81533B,3411G06054,07-Aug-98<END>
```

NOTE
- The HP 81640A/80A/82A/89A Tunable Laser modules will always return `HEWLETT-PACKARD` as the manufacturer.
- All other HP 8163A Series modules return `Agilent Technologies` as the manufacturer.
- The HP 8153A Series modules will always return `HEWLETT-PACKARD` as the manufacturer.

See *"*IDN?" on page 46* for information on mainframe identity strings.

affects: Independent of module type

command: **:SLOT[*n*]:OPTions?**

syntax: :SLOT[*n*]:OPTions?

description: Returns information about a module's options.

parameters: none

response: A string.

example:
```
slot1:opt? → NO CONNECTOR OPTION, NO INSTRUMENT OPTIONS<END>
```

affects: Independent of module type

command: **:SLOT[*n*]:TST?**

syntax: :SLOT[*n*]:TST?

description: Returns the latest selftest results for a module.

NOTE This command does not perform a selftest. Use selfTeST command, *TST? on page 59, to perform a selftest.

parameters: none

response: Returns an error code and a short description of the error.

example:
```
slot:tst? → +0,"self test OK"<END>
```

affects: Independent of module type

command: **:SLOT[*n*]:HEAD[*n*]:EMPTy?**

syntax: :SLOT[*n*]:HEAD[*n*]:EMPTy?

description: Returns whether an optical head is connected.

parameters: none

response: A ***boolean*** value:
0: there is a module in the slot
1: the module slot is empty

examples: `slot1:head:empt?` → `0<END>` An optical head is connected to the optical head interface module in slot 1

NOTE
- All other HP 8163A Series Optical Heads return `Agilent Technologies` as the manufacturer.
- The HP 8153A Series Optical Heads will always return `HEWLETT-PACKARD` as the manufacturer.

See *"*IDN?" on page 46* for information on mainframe identity strings.

affects: Optical heads

command: **:SLOT[*n*]:HEAD[*n*]:IDN?**

syntax: :SLOT[*n*]:HEAD[*n*]:IDN?

description: Returns information about the optical head.

parameters: none

response:

`HEWLETT-PACKARD`:	manufacturer
mmmm:	instrument model number (for example 81520A)
sssssss:	serial number
rrrrrrrrr:	date of firmware revision

example: `slot1:head:idn?` →
`HEWLETT-PACKARD, 81520A,3411G06054,07-Aug-98<END>`

affects: Optical heads

command: **:SLOT[*n*]:HEAD[*m*]:OPTions?**

syntax: :SLOT[*n*]:HEAD[*m*]:OPTions?

description: Returns information about an optical head's options.

parameters: none

response: A string.

example: `slot1:head:opt?` → `NO CONNECTOR OPTION, NO INSTRUMENT OPTIONS<END>`

affects: Optical heads

command: **:SLOT[*n*]:HEAD[*m*]:TST?**

syntax: :SLOT[*n*]:HEAD[*m*]:TST?

description: Returns the latest selftest results for an optical head.

NOTE This command does not perform a selftest. Use selfTeST command, *"*TST?" on page 49*, to perform a selftest.

parameters: none

response: Returns an error code and a short description of the error.

example: `slot:head:tst? → +0,"self test OK"<END>`

affects: Optical heads

command: **:SPECial:REBoot**

syntax: :SPECial:REBoot

description: Reboots the mainframe and all modules.

parameters: none

response: none

example: `spec:reb`

Measurement Functions – The SENSe Subsystem

The SENSe subsystem lets you control measurement parameters for a Power Sensor, an Optical Head Interface module, or a return loss module.

HP 81635A and HP 81619A- Master and Slave Channels

For the HP 81635A Dual Power Sensor and HP 81619A Dual Optical Head Interface module, channel 1 is the master channel and channel 2 is the slave channel. The master and slave channels share the same software and hardware triggering system. For some commands, setting parameters for the master channel sets the parameters for the slave channel. In these cases, you may only set parameters for the slave channel by setting master channel parameters.

The commands listed in Table 5 can only be configured using the master channel.

The commands listed in Table 6 are independent for both master and slave channels.

Command	Page
:INITiate[n]:[CHANnel[m]][:IMMediate]	70
:INITiate[n]:[CHANnel[m]]:CONTinuous/?	71
:READ[n][:CHANnel[m]][:SCALar]:POWer[:DC]?	71
:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO	73
:SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:LOGGing/?	74
:SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:MINMax/?	75
:SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:STABility/?	76
:SENSe[n][:CHANnel[m]]:FUNCtion:STATe/?	77
:SENSe[n]:[CHANnel[m]]:POWer:ATIME/?	78
:SENSe[n]:[CHANnel[m]]:POWer:RANGe:AUTO/?	79
:TRIGger[n][:CHANnel[m]]:INPut/?	110
:TRIGger[n][:CHANnel[m]]:INPut:REARm/?	111
:TRIGger[n][:CHANnel[m]]:OUTPut/?	112
:TRIGger[n][:CHANnel[m]]:OUTPut:REARm/?	112

Table 5 Commands that can only be configured using the master channel

Command	Page
:FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]?	70
:SENSe[n]:[CHANnel[m]]:CORRection[:LOSS][:INPut] [:MAGNitude]/?	72
:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO?	73
:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO:ALL	73
:SENSe[n][:CHANnel[m]]:FUNCtion:RESult?	77
:SENSe[n]:[CHANnel[m]]:POWer:RANGe[:UPPer]/?	79
:SENSe[n]:[CHANnel[m]]:POWer:REFerence/?	80
:SENSe[n]:[CHANnel[m]]:POWer:REFerence:DISPlay	81
:SENSe[n]:[CHANnel[m]]:POWer:REFerence:STATe/?	81
:SENSe[n]:[CHANnel[m]]:POWer:REFerence:STATe:RATio/?	82
:SENSe[n]:[CHANnel[m]]:POWer:UNIT/?	82
:SENSe[n]:[CHANnel[m]]:POWer:WAVelength/?	83

Table 6 Commands that are independent for both master and slave channels

command: **:FETCh[*n*][:CHANnel[*m*]][:SCAlar]:POWer[:DC]?**

syntax: :FETCh[*n*]:[CHANnel[*m*]][:SCAlar]:POWer[:DC]?

description: Reads the current power meter value. It does not provide its own triggering and so must be used with either continuous software triggering (see *":INITiate[n]:[CHANnel[m]]:CONTinuous?" on page 71*) or a directly preceding immediate software trigger (see *":INITiate[n]:[CHANnel[m]][:IMMediate]" on page 70*).

It returns the power meter value the previous software trigger measured. Any subsequent FETCh command will return the same value, if there is no subsequent software trigger.

parameters: none

response: The current power meter value as a **float** value in dBm,W or dB.

NOTE If the reference state is absolute, units are dBm or W.

If the reference state is relative, units are dB.

example: `fetc1:pow? → +6.73370400E-04<END>`

affects: All power meters and return loss modules

dual sensors: Master and slave channels are independent.

command: **:FETCh[*n*][:CHANnel[*m*]][:SCAlar]:RETurnloss?**

syntax: :FETCh[*n*]:[CHANnel[*m*]][:SCAlar]:RETurnloss?

description: Reads the current return loss value. It does not provide its own triggering and so must be used with either continuous software triggering (see *":INITiate[n]:[CHANnel[m]]:CONTinuous?" on page 71*) or a directly preceding immediate software trigger (see *":INITiate[n]:[CHANnel[m]][:IMMediate]" on page 70*).

It returns the return loss value the previous software trigger measured. Any subsequent FETCh command will return the same value, if there is no subsequent software trigger.

parameters: none

response: The current power meter value as a **float** value in dB.

example: `fetc1:ret? → +6.73370400E-00<END>`

affects: All return loss modules

command: **:INITiate[*n*]:[CHANnel[*m*]][:IMMediate]**

syntax: :INITiate[*n*]:[CHANnel[*m*]][:IMMediate]

description: Initiates the software trigger system and completes one full trigger cycle, that is, one measurement is made.

parameters: none

response: none

example: `init`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:INITiate[*n*]:[CHANnel[*m*]]:CONTinuous**

syntax: :INITiate[*n*]:[CHANnel[*m*]]:CONTinuous<wsp><boolean>

description: Sets the software trigger system to continuous measurement mode.

parameters: A ***boolean*** value: 0 or OFF: do not measure continuously
1 or ON: measure continuously

response: none

example: `init2:cont 1`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:INITiate[*n*]:[CHANnel[*m*]]:CONTinuous?**

syntax: :INITiate[*n*]:[CHANnel[*m*]]:CONTinuous?

description: Queries whether the software trigger system operates continuously or not

parameters: none

response: A ***boolean*** value: 0 or OFF: measurement is not continuous
1 or ON: measurement is continuous

example: `init2:cont? → 1<END>`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:READ[*n*][:CHANnel[*m*]][:SCALar]:POWer[:DC]?**

syntax: :READ[*n*]:[CHANnel[*m*]][:SCALar]:POWer[:DC]?

description: Reads the current power meter value. It provides its own software triggering and does not need a triggering command.

If the software trigger system operates continuously (see *":INITiate[n]:[CHANnel[m]]:CONTinuous?" on page 71*), this command is identical to *":FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]?" on page 70*.

If the software trigger system does not operate continuously, this command is identical to generating a software trigger (*":INITiate[n]:[CHANnel[m]][:IMMediate]" on page 70*) and then reading the power meter value.

NOTE The power meter must be running for this command to be effective.

parameters: none

response: The current power meter reading as a ***float*** value in dBm, W or dB.

NOTE If the reference state is absolute, units are dBm or W.
If the reference state is relative, units are dB.

example: `read1:pow? → +1.33555600E-006<END>`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also triggered.

To read a simultaneous result from the slave channel, send *":FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]?" on page 70* directly after this command.

command: **:READ[*n*][:CHANnel[*m*]][:SCALar]:RETurnloss?**

syntax: :READ[*n*]:[CHANnel[*m*]][:SCALar]:RETurnloss?

description: Reads the current return loss value. It provides its own software triggering and does not need a triggering command.

If the software trigger system operates continuously (see *":INITiate[n]:[CHANnel[m]]:CONTinuous?" on page 71*), this command is identical to *":FETCh[n][:CHANnel[m]][:SCAlar]:RETurnloss?" on page 70*.

If the software trigger system does not operate continuously, this command is identical to generating a software trigger (*":INITiate[n]:[CHANnel[m]][:IMMediate]" on page 70*) and then reading the power meter value.

NOTE The return loss module must be running for this command to be effective.

parameters: none

response: The current power meter reading as a ***float*** value in dB.

example: `read1:ret? → +1.33555600E-000<END>`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:CORRection[:LOSS][:INPut][:MAGNitude]**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:CORRection[:LOSS][:INPUT][:MAGNitude]<wsp><value>[DB|MDB]

description: Enters a calibration value for a module.

parameters: The calibration factor as a ***float*** value

If no unit type is specified, decibels (dB) is implied.

response: none

example: `sens1:corr 10DB`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:CORRection[:LOSS][:INPut][:MAGNitude]?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:CORRection[:LOSS][:INPUT][:MAGNitude]?

description: Returns the calibration factor for a module.

parameters: none

response: The calibration factor as a ***float*** value. Units are in dB, although no units are returned in the response message.

example: `sens1:corr? → +1.00000000E+000<END>`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:CORRection:COLLect:ZERO**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:CORRection:COLLect:ZERO

description: Zeros the electrical offsets for a power meter or return loss module.

parameters: none

response: none

example: `sens1:corr:coll:zero`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also zeroed.

command: **:SENSe[*n*]:[CHANnel[*m*]]:CORRection:COLLect:ZERO?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:CORREction:COLLect:ZERO?

description: Returns the status of the most recent zero command.

parameters: none

response:

0:	zero succeeded without errors.
any other number:	remote zeroing failed (the number is the error code returned from the operation).

example: `sens1:corr:coll:zero?` → `0<END>`

affects: All power meters and return loss modules

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:CORRection:COLLect:ZERO:ALL**

syntax: SENSe[*n*]:[CHANnel[*m*]]:CORRection:COLLect:ZERO:ALL

description: Zeros the electrical offsets for all installed power meter and return loss modules.

parameters: none

response: none

example: `sens:chan:corr:coll:zero:all`

affects: All power meters and return loss modules

dual sensors: Command is independent of channel.

NOTE Setting parameters for the logging function sets some parameters, including hidden parameters, for the stability and MinMax functions and vice versa. You must use the :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:LOGGing

command to set parameters before you start a logging function using the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe command.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:LOGGing**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:LOGGing<wsp><data points>, <averaging time>[NS|US|MS|S]

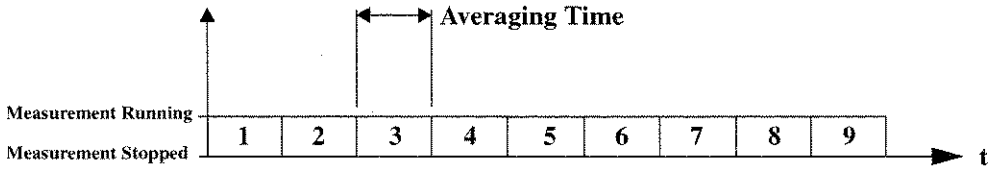
description: Sets the number of data points and the averaging time for the logging data acquisition function.

parameters:

Data Points: Data Points is the number of samples that are recorded before the logging mode is completed. Data Points is an **integer** value.

Averaging time: Averaging time is a time value in seconds.

There is no time delay between averaging time periods. Use *":SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:STABility?" on page 76* if you want to use delayed measurement.



If you specify no units for the averaging time value in your command, seconds are used as the default.

NOTE See *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77* for information on starting/stopping a data acquisition function.

NOTE See *":SENSe[n][:CHANnel[m]]:FUNCtion:RESult?" on page 77* for information on accessing the results of a data acquisition function.

NOTE See *"Triggering and Power Measurements" on page 108* for information on how triggering affects data acquisition functions.

response: none

example: `sens1:func:par:logg 64,1ms`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:LOGGing?**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:LOGGing?

description: Returns the number of data points and the averaging time for the logging data acquisition function.

parameters: none

response: Returns the number of data points as an **integer** value and the averaging time, t_{avg}, as a **float** value in seconds.

example: `sens1:func:par:logg? → +64,+1.00000000E-001<END>`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

NOTE Setting parameters for the MinMax function sets some parameters, including hidden parameters, for the stability and logging functions and vice versa. You must use the :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:MINMax

command to set parameters before you start a MinMax function using the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe command.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:MINMax**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:MINMax<wsp>CONTinous|WINDow|REFResh,<data points>

description: Sets the MinMax mode and the number of data points for the MinMax data acquisition function.

parameters:

CONTinous:	continuous MinMax mode
WINDow:	window MinMax mode
REFResh:	refresh MinMax mode

Data Points is the number of samples that are recorded in the memory buffer used by the WINDow and REFResh modes.

Data Points is an **integer** value.

See Chapter 3 of the HP 8163A Lightwave Multimeter, HP 8164A Lightwave Measurement System, & HP 8166A Lightwave Multichannel System User's Guide, for more information on MinMax mode.

NOTE See *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77* for information on starting/stopping a data acquisition function.

NOTE See *":SENSe[n][:CHANnel[m]]:FUNCtion:RESult?" on page 77* for information on accessing the results of a data acquisition function.

NOTE See *"Triggering and Power Measurements" on page 108* for information on how triggering affects data acquisition functions.

response: none

example:
```
sens1:func:par:minm WIND,10
```

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:MINMax?**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:MINMax?

description: Returns the MinMax mode and the number of data points for the MinMax data acquisition function.

parameters: none

response:

CONT:	continuous MinMax mode
WIND:	window MinMax mode
REFR:	refresh MinMax mode

The number of data points is returned as an **integer** value.

example:
```
sens1:func:par:minm? → WIND,+10<END>
```

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

NOTE Setting parameters for the stability function sets some parameters, including hidden parameters, for the logging and MinMax functions and vice versa. You must use the :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:STABility

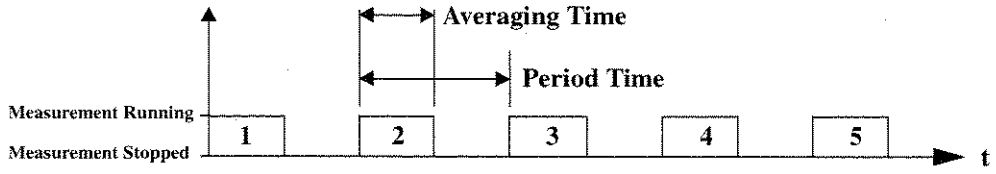
command to set parameters before you start a stability function using the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe command.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:STABility**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:STABility<wsp>
<total time>[NS|US|MS|S],<period time>[NS|US|MS|S],<averaging time>[NS|US|MS|S]

description: Sets the total time, period time, and averaging time for the stability data acquisition function.

parameters: Total time: The total time from the start of stability mode until it is completed.
Period time: A new measurement is started after the completion of every period time.
Averaging time: A measurement is averaged over the averaging time.



NOTE The total time should be longer than the period time.

The period time should be longer than the averaging time.

The number of data points is equal to the total time divided by the period time.

Total time, period time, and averaging time are time values in seconds.

If you specify no units in your command, seconds are used as the default.

NOTE See *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77* for information on starting/stopping a data acquisition function.

NOTE See *":SENSe[n][:CHANnel[m]]:FUNCtion:RESult?" on page 77* for information on accessing the results of a data acquisition function.

NOTE See *"Triggering and Power Measurements" on page 108* for information on how triggering affects data acquisition functions.

response: none

example:
```
sens1:func:par:stab 1s,0.1s,0.1s
```

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:STABility?**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:STABility?

description: Returns the total time, period time, and averaging time for the stability data acquisition function.

parameters: none

response: Total time, delay time, and averaging time are **float** values in seconds.

example:
```
sens1:func:par:stab? → +1.00000000E+000,
+1.00000000E-001,+1.00000000E-001<END>
```

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult?**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult?

description: Returns the data array of the last data acquisition function.

parameters: none

response: The last data acquisition function's data array as a binary block, one measurement value is a 4-byte-long **float** in Intel byte order.

example: `sens1:func:res?` → *returns a data array*

affects: All power meters and return loss modules

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:STATe**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:STATe<wsp>
LOGGing|STABility|MINMax,STOP|STARt

description: Enables/Disables the logging, MinMax, or stability data acquisition function mode.

parameters:

LOGGing:	Logging data acquisition function
STABility:	Stability data acquisition function
MINMax:	MinMax data acquisition function
STOP:	Stop data acquisition function
STARt:	Start data acquisition function

NOTE When you enable a logging data acquisition function for a HP 8163A Series Power Meter with averaging time of less than 100 ms with input hardware triggering disabled, all GPIB commands will be ignored for the duration of the function.

See *":SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:LOGGing" on page 74* for more information on the logging data acquisition function.

NOTE Stop any function before you try to set up a new function. Some parameters cannot be set until you stop the function.

response: none

example: `sens1:func:stat logg,star`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:STATe?**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:STATe?

description: Returns the function mode and the status of the data acquisition function.

parameters: none

response:

NONE	No function mode selected
LOGGING_STABILITY	Logging or stability data acquisition function
MINMAX	MinMax data acquisition function
PROGRESS	Data acquisition function is in progress
COMPLETE	Data acquisition function is complete

example: `sens1:func:stat?` → LOGGING_STABILITY,COMPLETE<END>

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:THReshold**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:THReshold<wsp><mode>, <threshold value>[PW|NW|UW|MW|Watt|DBM]

description: Sets the start mode and the threshold value.

parameters:

ABOVe:	Function starts when power is above the threshold value.
BELow:	Function starts when power is below the threshold value.
IMMediately:	Function starts immediately.
Threshold Value:	A **float** value in Watts or dBm.

response: none

example: `sens1:func:thr IMM,20nw<END>`

affects: All HP 8153A Lightwave Multimeter series power meters and the HP 81534A Return Loss module

command: **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:THReshold?**

syntax: :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:THReshold?

description: Returns the start mode and the threshold value.

parameters: none

response:

ABOV:	Function starts when power is above the threshold value.
BEL:	Function starts when power is below the threshold value.
IMM:	Function starts immediately.
Threshold Value:	A **float** value in Watts or dBm.

example: `sens1:func:thr?` → `IMM,+2.00000000E-008<END>`

affects: All HP 8153A Lightwave Multimeter series power meters and the HP 81534A Return Loss module

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:ATIMe**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:ATIMe<wsp><averaging time>[NS|US|MS|S]

description: Sets the averaging time for the module.

parameters: The averaging time as a float value in seconds.

If you specify no units in your command, seconds are used as the default.

response: none

example: `sens1:pow:atim 1s`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:ATIMe?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:ATIMe?

description: Returns the averaging time for the module.

parameters: none

response: The averaging time as a ***float*** value in seconds.

example: `sens1:pow:atim?` → `+1.00000000E+000<END>`

affects: All power meters and return loss modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe[:UPPer]**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe[:UPPer]<wsp><value>[DBM]

description: Sets the power range for the module.

The range changes at 10 dBm intervals. The corresponding ranges for linear measurements (measurements in Watts) is given below:

Range	Upper Linear Power Limit	Range	Upper Linear Power Limit
+30 dBm	1999.9 mW	–50 dBm	19.999 nW
+20 dBm	199.99 mW	–60 dBm	1999.9 pW
+10 dBm	19.999 mW	–70 dBm	199.99 pW
0 dBm	1999.9 µW	–80 dBm	19.999 pW
–10 dBm	199.99 µW	–90 dBm	1.999 pW
–20 dBm	19.999 µW	–100 dBm	0.199 pW
–30 dBm	1999.9 nW	–110 dBm	0.019 pW
–40 dBm	199.99 nW		

parameters: The range as a **float** value in dBm. The number is rounded to the closest multiple of 10, because the range changes at 10 dBm intervals. Units are in dBm.

response: none

example: `sens1:pow:rang -20DBM`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe[:UPPer]?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe[:UPPer]?

description: Returns the range setting for the module

parameters: none

response: The range setting as a ***float*** value in dBm

(–110 Š≤ *value* ≤ +30).

example: `sens1:pow:rang?` → `-2.00000000E+001<END>`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:AUTO**

syntax: SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:AUTO <wsp><boolean>

description: Enables or disables automatic power ranging for the module.

If automatic power ranging is enabled, ranging is automatically determined by the instrument. Otherwise, it must be set by the `sensn:pow:rang` command.

parameters: A ***boolean*** value: 0 or OFF: automatic ranging disabled
1 or ON: automatic ranging enabled

response: none

example: `sens1:pow:rang:auto 1`

affects: All power meters

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:AUTO?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:AUTO?

description: Returns whether automatic power ranging is being used by the module.

parameters: none

response: A ***boolean*** value: 0: automatic ranging is not being used.
1: automatic ranging is being used.

example: `sens1:pow:rang:auto? → 1<END>`

affects: All power meters

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence<wsp>
TOMODule|TOREF,<value>PW|NW|UW|MW|Watt|DBM|DB|MDB

description: Sets the sensor reference value.

parameters: TOMODule: Sets the reference value in dB used if you choose measurement relative to another channel

TOREF: Sets the reference value in Watts or dBm if you choose measurement relative to a constant reference value

The reference as a **float** value.

NOTE You must append a unit type
- dB if you use TOMODule or
- Watts or dBm if you use TOREF.

NOTE The two reference values are completely independent. When you change the reference mode using the command *":SENSe[n]:[CHANnel[m]]:POWer:REFerence:STATe:RATio?" on page 82*, the instrument uses the last reference value entered for the selected reference mode.

response: none

example: `sens1:pow:ref tomod,-40DB`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence?<wsp>TOMODule|TOREF

description: Returns the sensor reference value.

parameters: TOMODule: Returns the reference value in dB used if you choose measurement relative to another channel

TOREF: Returns the reference value in Watts or dBm if you choose measurement relative to a constant reference value

response: The reference as a **float** value.

example: `sens1:pow:ref? toref → +1.00000000E-006<END>`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence:DISPlay**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence:DISPlay

description: Takes the input power level value as the reference value.

parameters: none

response: none

example: `sens1:pow:ref:disp`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence:STATe**

syntax: :SENSe[*n*]:[CHANnel[*m*]]POWer:REFerence:STATe<wsp><boolean>

description: Sets the measurement units to relative or absolute units.

parameters: A ***boolean*** value: 0 or OFF: absolute
1 or ON: relative

response: none

example: `sens1:pow:ref:stat 1`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence:STATe?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]POWer:REFerence:STATe?

description: Inquires whether the current measurement units are relative (dB) or absolute (Watts or dBm).

parameters: none

response: A ***boolean*** value: 0: absolute
1: relative

example: `sens1:pow:ref:stat? → 1<END>`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence:STATe:RATio**

syntax: :SENSe[*n*]:[CHANnel[*m*]]POWer:REFerence:STATe:RATio<wsp><slot number>|255|TOREF,<channel number>

description: Selects the reference for the module.

parameters:

slot number:	an **integer** value representing the slot number you want to reference
255 or TOREF:	results are displayed relative to an absolute reference
channel number:	an **integer** value representing the channel number you want to reference

NOTE If you want to reference another power sensor channel, use an integer value corresponding to the slot for the first parameter and an integer value corresponding to the channel for the second value.

If you want to use an absolute reference, use TOREF as the first parameter and any integer value as the second parameter.

response: none

examples:

`sens1:pow:ref:stat:rat 2,1`	References channel 2.1
`sens1:pow:ref:stat:rat TOREF,1`	References an absolute reference

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:REFerence:STATe:RATio?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]POWer:REFerence:STATe:RATio?

description: Returns the reference setting for the module.

parameters: none

response: results are displayed relative to an absolute reference or to the current power reading from another channel.

examples:

`sens1:pow:ref:stat:rat? → +255,+0<END>`	results are displayed relative to an absolute reference
`sens1:pow:ref:stat:rat? → +2,+1<END>`	results are displayed relative to channel 2.1

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:UNIT**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:UNIT<wsp>DBM|0|Watt|1

description: Sets the sensor power unit

parameters:

An ***integer*** value:	0: dBm
	1: Watt

or DBM or Watt

response: none

example: `sens1:pow:unit 1`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:UNIT?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:UNIT?

description: Inquires the current sensor power unit

parameters: none

response: An ***integer*** value: 0: Current power units are dBm.
1: Current power units are Watts.

example: `sens1:pow:unit?` → `+1<END>`

affects: All power meters

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:WAVelength**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:WAVelength<wsp><value>|MIN|MAX|DEF [PM|NM|UM|MM|M]

description: Sets the sensor wavelength.

parameters: The wavelength as a ***float*** value in meters.

Also allowed are:

`MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

response: none

example: `sens1:pow:wav 1550nm`

affects: All power meters and return loss modules

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:POWer:WAVelength?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:POWer:WAVelength?[<wsp>MIN|MAX|DEF]

description: Inquires the current sensor wavelength.

parameters: none

Also allowed are:

`MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

response: The wavelength as a ***float*** value in meters.

example `sens1:pow:wav?` → `+1.55000000E-006<END>`

affects: All power meters and return loss modules

dual sensors: Master and slave channels are independent.

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:FACTory**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:FACTory

description: Overwrites the current calibration values with the factory-set calibration settings. See *":SENSe[n]:[CHANnel[m]]:RETurnloss:CALibration:COLLect:REFLectance" on page 84* and *":SENSe[n]:[CHANnel[m]]:RETurnloss:CALibration:COLLect:TERMination" on page 84* for information on calibrating your return loss module.

parameters: none

response: none

example `sens1:ret:cal:fact`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:COLLect:REFLectance**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:COLLect:REFLectance

description: Start the calibration and save the calibration values for a defined reflectance reference measurement. See *":SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[1]" on page 85* for information on setting the return loss value of your reference reflector.

parameters: none

response: none

example `sens1:ret:cal:coll:refl`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:COLLect:TERMination**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:COLLect:TERMination

description: Start the calibration and save the calibration values for a defined termination reference measurement. See *":SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[1]" on page 85* for information on setting the return loss value of your reference reflector.

parameters: none

response: none

example `sens1:ret:cal:coll:term`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:FPDelta[*l*]**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:FPDelta[*l*]<wsp><value>[dB]

description: Sets the front panel delta, that is, the loss correction value, for example, due to the front panel connector. Twice this value is added to the measured Return Loss.

NOTE Use [*l*] to set the front panel delta for an external source or the upper or lower wavelength laser source of a dual return loss module.
An external laser source is denoted by *0*. *0* is the default value of [*l*].
A lower wavelength source is denoted by *1*.
An upper wavelength source is denoted by *2*.

parameters: Sets the front panel delta as a **float** value in dB

response: none

example `sens1:ret:cal:corr:fpd 0.08DB`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:FPDelta[*l*]?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:FPDelta[*l*]?

description: Returns the front panel delta, that is, the loss correction value, for example, due to the front panel connector. Twice this value is added to the measured Return Loss.

NOTE Use [*l*] to query the front panel delta for an external source or the upper or lower wavelength laser source of a dual return loss module.
An external laser source is denoted by *0*. *0* is the default value.
A lower wavelength source is denoted by *1*.
An upper wavelength source is denoted by *2*.

parameters: Returns the front panel delta as a **float** value in dB

response: none

example `sens1:ret:cal:corr:fpd? → +8.00000000E-002<END>`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:REFLectance[*l*]**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:REFLectance[*l*]<wsp><value>[dB]

description: Sets the Return Loss Reference, the return loss value of your reference reflector.

For example, the HP 81000BR reference reflector provides an accurate and stable 0.18 dB reference.

NOTE Use [*l*] to set the return loss value of your reference reflector for an external source or the upper or lower wavelength laser source of a dual return loss module.
An external laser source is denoted by *0*. *0* is the default value of [*l*].
A lower wavelength source is denoted by *1*.
An upper wavelength source is denoted by *2*.

parameters: Sets the Return Loss Reference as a **float** value in dB

response: none

example `sens1:ret:cal:corr:refl 0.18DB`

affects: All return loss modules

command: **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:REFLectance[*l*]?**

syntax: :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CORRection:REFLectance[*l*]?

description: Returns the Return Loss Reference, the return loss value of your reference reflector.

For example, the HP 81000BR reference reflector provides an accurate and stable 0.18 dB reference.

NOTE Use [*l*] to query the return loss value of your reference reflector for an external source or the upper or lower wavelength laser source of a dual return loss module.
An external laser source is denoted by *0*. *0* is the default value of [*l*].
A lower wavelength source is denoted by *1*.
An upper wavelength source is denoted by *2*.

parameters: none

response: Returns the Return Loss Reference as a float value in dB

example `sens1:ret:cal:corr:refl? → +1.80000000E-001<END>`

affects: All return loss modules

Signal Generation – The SOURce Subsystem

The SOURce subsystem allows you to control a laser source module, DFB source module, tunable laser module, or a return loss module that has an internal source.

command: **:OUTPut[*n*][:CHANnel[*m*]]:CONNection**

syntax: OUTPut[*n*][:CHANnel[*m*]]:CONNection<wsp>MOD|VPP

description: Sets the analog output parameter.

parameters: MOD: The modulation frequency modulates the analog output.

VPP: Output Voltage is proportional to optical power.

response: none

example: `outp1:conn mod`

affects: All tunable laser modules

command: **:OUTPut[*n*][:CHANnel[*m*]]:CONNection?**

syntax: OUTPut[*n*][:CHANnel[*m*]]:CONNection?

description: Returns the analog output parameter.

parameters: none

response: MOD: The modulation frequency modulates the analog output.

VPP: Output Voltage is proportional to optical power.

example: `outp1:conn? → MOD<END>`

affects: All tunable laser modules

command: **:OUTPut[*n*][:CHANnel[*m*]]:PATH**

syntax: :OUTPut[*n*][:CHANnel[*m*]]:PATH<wsp><path>

description: Sets the regulated path.

parameters:
- HIGHpower: The High Power output is regulated.
- LOWSse: The Low SSE output is regulated.
- BHRegulated: Both outputs are active but only the High Power output is Regulated.
- BLRegulated: Both outputs are active but only the Low SSE output is Regulated.

response: none

example: `output1:path high`

affects: All tunable laser modules

command: **:OUTPut[*n*][:CHANnel[*m*]]:PATH?**

syntax: :OUTPut[*n*][:CHANnel[*m*]]:PATH?

description: Returns the regulated path.

parameters: none

response:
- HIGH: The High Power output is regulated.
- LOWS: The Low SSE output is regulated.
- BHR: Both outputs are active but only the High Power output is Regulated.
- BLR: Both outputs are active but only the Low SSE output is Regulated.

example: `output1:path?` → `HIGH<END>`

affects: All tunable laser modules

command: **:OUTPut[*n*][:CHANnel[*m*]][:STATe]**

syntax: :OUTPut[*n*][:CHANnel[*m*]][:STATe]<wsp>OFF|ON|0|1

description: Switches the laser current off and on.

The laser emits light only when the current is on. Set the state to `OFF` or `0` to switch the laser current off. Set the state to `ON` or `1` to switch the laser current on. The default is for the laser current to be off.

parameters:
- `0` or `OFF`: switch laser current off
- `1` or `ON`: switch laser current on

response: none

example: `outp 1`

affects: All laser source, DFB source, tunable laser modules and return loss modules with an internal source

command: **:OUTPut[*n*][:CHANnel[*m*]][:STATe]?**

syntax: :OUTPut[*n*][:CHANnel[*m*]][:STATe]?

description: Returns the current state of the laser current.

parameters: none

response: A ***boolean*** value:
- 0 – laser current off
- 1 – laser current on

example: `outp?` → `1<END>`

affects: All laser source, DFB source, tunable laser modules and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:AM[:INTernal]:FREQuency[*l*]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:AM[:INTernal]:FREQuency[*l*]<wsp><frequency> [THZ|GHZ|MHZ|KHZ|HZ]

description: Sets the frequency of the amplitude modulation of the laser output.

parameters: The frequency as a ***float*** value in Hz.

Also allowed are: `MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

The default units are `HZ`, although `KHZ`, `MHZ`, `GHZ`, and `THZ` can also be specified.

The resolution of the frequency is always 1 Hz.

NOTE Use [*l*] to set the modulation frequency of the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

response: none

example:
```
sour2:am:freq 270hz
```

affects: All laser source, DFB source, and tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:AM[:INTernal]:FREQuency[*l*]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:AM[:INTernal]:FREQuency[*l*]? `[MIN | DEF | MAX]`

description: Returns the frequency of the amplitude modulation as a ***float*** value in Hertz.

parameters: `MIN`: minimum modulation frequency

`MAX`: maximum modulation frequency

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum modulation frequency and the maximum modulation frequency

NOTE Use [*l*] to query the modulation frequency of the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

response: modulation frequency relevant to the current value or specified parameter (if MIN, MAX, or DEF are chosen as a parameter).

example:
```
sour2:am:freq? min → +2.00000000E+002<END>
```

All laser source, DFB source, and tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:AM:SOURce[*l*]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:AM:SOURce[*l*]<wsp> INT|INT1|INT2|EXT|0|1|2

description: Selects the type or source of the modulation of the laser output.

parameters:

0, INT1, or INTernal:	internal digital modulation
1, COHCtrl, or INT2:	coherence control
2, AEXTernal, or EXT:	external analog modulation
3 or DEXTernal:	external digital modulation
4 or LFCohctrl:	low frequency coherence control
5 or WVLLocking:	wavelength locking
6 or BACKplane:	external digital modulation using Input Trigger Connector

NOTE Use [*l*] to set the modulation source of the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

response: none

example: `sour2:am:sour int`

affects: All laser source, DFB source, and tunable laser modules can use internal digital modulation

All other modulation modes are only available with tunable laser modules.

command: **[:SOURce[*n*]][:CHANnel[*m*]]:AM:SOURce[*l*]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:AM:SOURce[*l*]?

description: Returns the type or source of the modulation of the laser output.

parameters:

0:	internal digital modulation
1:	coherence control
2:	external analog modulation
3:	external digital modulation
4:	low frequency coherence control
5:	wavelength locking
6:	external digital modulation using Input Trigger Connector

NOTE Use [*l*] to query the modulation source of the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

response: none

example: `sour2:am:sour?` → `+0`<END>

affects: All laser source, DFB source, and tunable laser modules can use internal digital modulation

All other modulation modes are only available with tunable laser modules.

command: **[:SOURce[*n*]][:CHANnel[*m*]]:AM:STATe[*l*]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:AM:STATe[*l*]<wsp> OFF|ON|0|1

description: Enables and disables amplitude modulation of the laser output.

parameters: A ***boolean*** value: `OFF` or `0`: modulation disabled (default)
`ON` or `1`: modulation enabled.

NOTE Use [*l*] to enable/disable amplitude modulation for the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

NOTE When the internal modulation is selected, the Modulation Output on the front panel outputs a version of the modulating signal that has the same frequency and phase as the modulating signal, but has a fixed, TTL-level amplitude. You can use this to synchronize your external measuring equipment to your instrument.

To allow for your possible synchronization requirements, there are two ways in which the signal can be output. Either the signal is combined with the laser-ready signal, so that the output is kept low when there is no optical signal being output (for example, while the laser is settling after a change of wavelength). Or the modulation signal is output all the time. This is set by the **`:SOURCE:MODOUT`** command (see *"[:SOURce[n]][:CHANnel[m]]:MODout" on page 91*).

NOTE When you enable lambda logging, see *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:LLOGging" on page 103*, and modulation simultaneously, a sweep can not be started, see *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:[STATe]" on page 106*.

response: none

example:
```
sour2:am:stat 0
```

affects: All laser source, DFB source, and tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:AM:STATe[*l*]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:AM:STATe[*l*]?

description: Returns the current state of modulation.

NOTE Use [*l*] to query the current state of modulation of the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

parameters: none

response: A ***boolean*** value: 0: modulation is disabled
1: modulation is enabled

example:
```
sour2:am:stat? → 0<END>
```

affects: All laser source, DFB source, and tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:MODout**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:MODout<wsp>FRQ|FRQRDY|0|1

description: Sets the modulation output

parameters:
`FRQ` or `0`: modulation signal is output all the time

`FRQRDY` or `1`: modulation is combined with the laser-ready signal. In this case, the output is kept low when no optical signal is output (for example, while the laser is settling after a change of wavelength).

response: none

example:
```
sour2:mod 0
```

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:MODout?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:MODout?

description: Returns the mode of the modulation output.

parameters: none

response:
`0`: modulation signal is output all the time

`1`: modulation is combined with the laser-ready signal. In this case, the output is kept low when no optical signal is output (for example, while the laser is settling after a change of wavelength).

example:
```
sour2:mod? → 0<END>
```

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]<wsp><value>[DB|MDB]

description: Sets the level of attenuation.

parameters: Any value in the specified range (see the specifications in the appropriate *User's Guide*).

Also allowed (for tunable laser modules only) are:
- `MIN`: minimum programmable value
- `MAX`: maximum programmable value
- `DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

NOTE Use [*l*] to set the attenuation level of the upper or lower wavelength laser source of a dual-wavelength laser source or of a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

NOTE Tunable laser modules with in-built optical attenuators need to be in Manual Attenuation Mode (see *"[:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation:AUTO" on page 93*) for this value to have an affect. The output power is a combination of this value and the laser output power (see *"[:SOURce[n]][:CHANnel[m]]:POWer[:LEVel][:IMMediate][:AMPLitude]" on page 94*).

NOTE In this respect, this command does not conform to the SCPI standard. The SCPI standard requires that entering an explicit value for the attenuation switches the attenuation mode OFF.

The default units are **dB**.

response: none

example:
```
sour2:pow:att 22.32db
```

affects: All tunable laser modules with an in-built optical attenuator, all laser source modules, and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]?`[MIN|DEF|MAX]`

description: Returns the attenuation level.

When using a tunable laser module with an in-built optical attenuator, the value returned applies only to the attenuation mode (see *"[:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation:AUTO" on page 93*).

NOTE Use [*l*] to query the attenuation level of the upper or lower wavelength laser source of a dual-wavelength laser source or of a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

parameters: Also allowed (for tunable laser modules only) are:
- `MIN`: minimum amplitude level
- `MAX`: maximum amplitude level
- `DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum amplitude level and the maximum amplitude level

response: attenuation level relevant to the current value or specified parameter (if MIN, MAX, or DEF are chosen as a parameter).

example:
```
sour2:pow:att? def → +3.10000000+E001<END>
```

affects: All tunable laser modules with an in-built optical attenuator, all laser source modules, and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:AUTO**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:AUTO<wsp>OFF|ON|0|1

description: Selects Automatic or Manual Attenuation Mode.

In Automatic Attenuation Mode, you specify the output power.
In Manual Attenuation Mode, you must specify both the laser output power, and the attenuation level.

parameters:
`OFF` or `0`: Attenuation Mode

`ON` or `1`: Power Mode

response: none

example:
```
sour2:pow:att:auto 1
```

affects: All tunable laser modules with an in-built optical attenuator

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:AUTO?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:AUTO?

description: Returns whether the instrument is in Automatic or Manual Attenuation Mode.

parameters: none

response:
`0`: Manual Attenuation Mode

`1`: Automatic Attenuation Mode

example:
```
sour2:pow:att:auto? → 1<END>
```

affects: All tunable laser modules with an in-built optical attenuator

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:DARK**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:DARK<wsp>OFF|ON|0|1

description: Sets or unsets the attenuator to 'dark' position.

Dark position blocks all light from the laser. You can use this as an alternative to disabling the laser. the advantage of doing this is that you avoid the laser rise time.

This command is available in Attenuation Mode Only.

parameters:
`OFF` or `0`: Unsets dark position

`ON` or `1`: Sets dark position

response: none

example:
```
sour2:pow:att:dark 1
```

affects: All tunable laser modules with an in-built optical attenuator

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:DARK?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation:DARK?

description: Queries whether the attenuator is set to 'dark' position (where all light is blocked by the laser).

parameters: none

response:
`0`: dark position not set

`1`: dark position set

example:
```
sour2:pow:att:dark? → 1<END>
```

affects: All tunable laser modules with an in-built optical attenuator

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel][:IMMediate][:AMPLitude]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel][:IMMediate][:AMPLitude]<wsp><value>
[PW|NW|UW|MW|Watt|DBM]

description: Sets the power of the laser output.

NOTE If an optical attenuator is installed, the power value returned is dependent on whether you are using power or attenuation mode (see *"[:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation:AUTO" on page 93*).

- If you are using power mode, the value returned is the output power.
- If you are using attenuation mode, the value returned is the laser output power, and you must also use the attenuation value to calculate the output power (see *"[:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation[l]" on page 92*).

The values for the output power that you set in the Power Mode, and the laser output power that you set in the Attenuation Mode, are stored and used independently.

NOTE The instrument may not be able to output a signal with the maximum programmable power, it will output a signal with the maximum power. Use the *"[:SOURce[n]][:CHANnel[m]]:POWer[:LEVel][:IMMediate][:AMPLitude]" on page 94* to query the power being output.

The default units are `DBM` or `W`, depending on the unit selected using the following command: *"[:SOURce[n]][:CHANnel[m]]:POWer:UNIT" on page 96*.

parameters: Any value in the specified range (see the appropriate *User's Guide*).

Also allowed are:

`MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value, but is half the sum of the minimum programmable value and the maximum programmable level

response: none

example:
```
sour2:pow 23uW
```

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel][:IMMediate][:AMPLitude[*l*]]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel][:IMMediate][:AMPLitude[*l*]]?<wsp>
[MIN|DEF|MAX]

description: Returns the amplitude level of the output power.

The value returned is the actual amplitude that is output, which may be dif-ferent from the value set for the output. If these two figures are not the same, it is indicated in the :STATus:OPERation register.

NOTE If an optical attenuator is installed, the power value returned is dependent on whether you are using power or attenuation mode (see *"[:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation:AUTO" on page 93*).

- If you are using power mode, the value returned is the output power.
- If you are using attenuation mode, the value returned is the laser output power, and you must also use the attenuation value to calculate the output power (see *"[:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation[l]" on page 92*).

The values for the output power that you set in the Power Mode, and the laser output power that you set in the Attenuation Mode, are stored and used independently.

parameters: Also allowed (for tunable laser modules only) are:
- MIN: minimum amplitude level
- MAX: maximum amplitude level
- DEF: This is not the preset (*RST) default value but is half the sum of, the minimum amplitude level and the maximum amplitude level

NOTE Use [*l*] to query the amplitude level of the output power of the upper or lower wavelength laser source of a dual-wavelength laser source or a return loss module with an internal dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

response: Amplitude level relevant to the current value or specified parameter (if MIN, MAX, or DEF are chosen as a parameter).

example: `sour2:pow? → +8.00000000E-004<END>`

affects: All laser source, DFB source, and tunable laser modules and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime<wsp><value>[NS|US|MS|S]

description: Sets the laser rise time of the chosen source.

parameters: Any value in the specified range (see the appropriate *User's Guide*).
Also allowed are:
- MIN: minimum programmable value
- MAX: maximum programmable value
- DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable level and the maximum programmable level

response: none

example: `sour2:pow:ris 10ns`

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime?<wsp>[MIN|DEF|MAX]

description: Returns the laser rise time of the chosen source.

parameters: Also allowed are:
`MIN`: minimum programmable value
`MAX`: maximum programmable value
`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable level and the maximum programmable level

response: The rise time as a **float** value in seconds.

example: `sour2:pow:ris?` → `+1.00000000E-009<END>`

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:STATe**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer[1|2]:STATe<wsp><boolean>

description: Switches the laser of the chosen source on or off.

parameters: A ***boolean*** value: 0: Laser Off
1: Laser On

response: none

example: `sour2:pow:stat 1`

affects: All laser source, DFB source, and tunable laser modules and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:STATe?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:STATe?

description: Queries the laser state of the chosen source.

parameters: none

response: A ***boolean*** value: 0: Laser Off
1: Laser On

example: `sour2:pow:stat?` → 1<END>

affects: All laser source, DFB source, and tunable laser modules and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:UNIT**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:UNIT<wsp>DBM|0|Watt|1

description: Sets the power units

parameters: 0 or DBM: dBm (default)
1 or W: Watts

response: none

example: `sour2:pow:unit w`

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:UNIT?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:POWer:UNIT?

description: Return the current power units

parameters:
`0`: dBm
`1`: Watts

response: none

example: `sour2:pow:unit?` → `+0<END>`

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:WAVelength**

syntax: [:SOURce[*n*]][:CHANnel[*m*]:POWer:WAVelength[<wsp>LOWer|UPPer|BOTH]

NOTE For compatibility reasons, WAVelength may be replaced with WAVE.

description: Sets the wavelength source for a dual-wavelength laser source.

parameters:

LOWer:	The lower wavelength source
UPPer:	The upper wavelength source
BOTH:	Both wavelength sources

response: none

example: `sour2:pow:wav upp`

affects: All dual-wavelength laser source modules and return loss modules with two internal sources

command: **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:WAVelength?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]:POWer:WAVelength?

NOTE For compatibility reasons, WAVelength may be replaced with WAVE.

description: Returns the wavelength source for a dual-wavelength laser source.

parameters: none

response:

LOW	The lower wavelength source
UPP	The upper wavelength source
BOTH	Both wavelength sources

example: `sour2:pow:wav?` → `LOW<END>`

affects: All dual-wavelength laser source modules and return loss modules with two internal sources

command: **[:SOURce[*n*]][:CHANnel[*m*]]:READout:DATA?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]:READout:DATA?

description: Returns the data as a binary stream from either a lambda logging operation or the maximum power the laser can produce at each wavelength.

parameters:

LLOGging: Returns a binary stream that contains each wavelength step of the lambda logging operation, see *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:LLOGging" on page 103*. Each binary block is an 8-byte long **double** in Intel byte order.

PMAX: Returns a binary stream that contains the maximum power the laser can produce at each wavelength. Each binary block is a 8-byte long **double** (the wavelength value) followed by a 4-byte long **float** (the power value). The stream is in Intel byte order.

response: A binary stream in Intel byte order.

example: `sour2:read:data? llog` → *the data as a binary stream*

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:READout:POINts?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]:READout:POINts?<wsp>LLOGging|PMAX

description: Returns the number of datapoints that the [:SOURce[*n*]][:CHANnel[*m*]]:READout:DATA? command will return.

parameters:

LLOGging: Returns the number of wavelength steps for a lambda logging operation, see *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:LLOGging" on page 103*.

PMAX: Returns the number of datapoints (each datapoint contains a value for wavelength and power) the [:SOURce[*n*]][:CHANnel[*m*]]:READout:DATA? PMAX command will return, number of datapoints depends on the calibration data for your module.

response: The number of datapoints as an ***integer*** value.

example: `sour2:read:poin? pmax` → `120<END>`

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength[:CW|:FIXED]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength[:CW|:FIXED]<wsp><value>[PM|NM|UM|MM|M]

description: Sets the absolute wavelength of the output.

parameters: Any wavelength in the specified range (see the specifications in the appropriate *User's Guide*).

The programmable range is larger than the range specified in the *User's Guide*. The programmable range is set individually for each instrument when it is calibrated during production.

Also allowed are:

`MIN`: minimum wavelength value

`MAX`: maximum wavelength value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum wavelength value and the maximum wavelength value

The default units are `M`.

response: none

example: `sour2:wav 1550NM`

affects: All tunable laser and DFB source modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength[:CW[*l*]|:FIXED[*l*]]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]:WAVelength[:CW[*l*]|:FIXED[*l*]]?[<wsp>[MIN|DEF|MAX]

description: Returns the wavelength value in meters.

NOTE Use [*l*] to query the upper or lower wavelength laser source of a dual-wavelength laser source. The default value of [*l*] is *1*, the lower wavelength source. The upper wavelength source is denoted by *2*.

parameters: none

Also allowed (for tunable laser modules only) are:

`MIN`: minimum wavelength

`MAX`: maximum wavelength

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum wavelength value and the maximum wavelength value

response: The wavelength as a ***float*** value in meters.

example:

`sour0:wav?` → `+1.5672030E-006<END>` Returns the current wavelength value for a tunable laser module.

`sour0:wav? min` → `+1.5500000E-006<END>` Returns minimum wavelength for a tunable laser module.

`sour2:wav:fixed2?` → `+1.61544494E-006<END>` Returns the wavelength value of the upper wavelength source of a dual-wavelength laser source.

affects: All laser source, DFB source, and tunable laser modules and return loss modules with an internal source

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ARA**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ARA

description: Realigns the laser cavity.

parameters: none

response: none

example: `sour2:wav:corr:ara`

affects: All tunable laser modules except HP 81689A

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO

description: Executes a wavelength zero.

parameters: none

response: none

example: sour2:wav:`corr:zero`

affects: All tunable laser modules except HP 81689A

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency<wsp><value> [THZ|GHZ|MHZ|KHZ|HZ]

description: Sets the frequency difference used to calculate a relative wavelength. The output wavelength is made up of the reference wavelength and this frequency difference.

The default units for frequency are Hertz.

The output wavelength (λ) is set from the base wavelength (λ_0) and the frequency offset (df). The formula for calculating the output wavelength is:

$$\lambda = \frac{(c)}{((\lambda_o df) + c)} \lambda_o$$

where c is the speed of light in a vacuum (2.990×10^8 ms^{-1})

parameters: The frequency difference is a **float** value in Hz.

response: none

example: sour2:`wav:freq -10THZ`

affects: All tunable laser and DFB modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency?

description: Returns the frequency difference used to calculate a relative wavelength.

parameters: none

response: Returns the frequency difference as a **float** value in Hz.

example: `wav:freq? → -1.00000000E+013<END>`

affects: All tunable laser and DFB modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:REFerence?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:REFerence?

description: Returns the reference wavelength (λ_0).

parameters: none

response: The wavelength as a ***float*** value in meters.

example: `sour2:wav:ref?` → +1.5500000E-006<END>

affects: All tunable laser and DFB modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:REFerence:DISPlay**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:REFerence:DISPlay

description: Sets the reference wavelength to the value of the output wavelength ($\lambda \rightarrow \lambda_0$), that is, sets the frequency offset (df) to zero.

parameters: none

response: none

example: `sour2:wav:ref:disp`

affects: All tunable laser and DFB modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes<wsp><value>|MIN|MAX|DEF

description: Sets the number of cycles.

parameters: The number of cycles is an integer value.

Also allowed are:

`MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

response: none

example: `wav:swe:cycl 3`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes? [<wsp>MIN|MAX|DEF]

description: Returns the number of cycles.

parameters: none

Also allowed are:

`MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

response: The number of cycles as an integer value.

example: `wav:swe:cycl?` → +3<END>

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:DWELl**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:DWELl<wsp><value>|MIN|MAX|DEF[NS|US|MS|S]

description: Sets the dwell time.

parameters: The dwell time as a ***float*** value.

If you specify no units in your command, seconds are used as the default.

Also allowed are: `MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

response: none

example: `wav:swe:dwel 500ms`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:DWELl?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:DWELl?[<wsp>MIN|MAX|DEF]

description: Returns the dwell time.

parameters: none

Also allowed are: `MIN`: minimum programmable value

`MAX`: maximum programmable value

`DEF`: This is not the preset (`*RST`) default value but is half the sum of, the minimum programmable value and the maximum programmable value

response: The dwell time in seconds.

example: `wav:swe:dwel?` → `+5.00000000E-001<END>`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:LLOGging**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:LLOGging<wsp>OFF|ON|0|1

description: Switches lambda logging on or off. Lambda logging is a feature that records the exact wavelength of a tunable laser module when a trigger is generated during a continuous sweep. You can read this data using the [:SOURce[*n*]][:CHANnel[*m*]]:READout:DATA? command.

NOTE The following settings are the prerequisites for Lambda Logging:

- Set *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:MODE" on page 103* to CONTinuous.
- Set *":TRIGger[n][:CHANnel[m]]:OUTPut" on page 112* to STFinished (step finished).
- Set *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:CYCLes" on page 101* to 1.
- Set *"[:SOURce[n]][:CHANnel[m]]:AM:STATe[1]" on page 90* to OFF.

If any of the above prerequisites are not met, then when the sweep is started the status "Sweep parameters inconsistent" will be returned and Lambda Logging will automatically be turned off.

NOTE Lambda logging is disabled at the end of a sweep.

parameters:

0 or OFF:	switch lambda logging off
1 or ON:	switch lambda logging on

response: none

example:
```
wav:swe:llog 1
```

affects: All tunable laser modules except HP 81689A

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:LLOGging?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:LLOGging?

description: Returns the state of lambda logging.

parameters: none

response: A ***boolean*** value: 0 – lambda logging is switched off; 1 – lambda logging is switched on

example:
```
wav:swe:llog? → 1<END>
```

affects: All tunable laser modules except HP 81689A

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:MODE**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:MODE<wsp><mode>

description: Sets the sweep mode.

parameters:

STEPped:	Stepped sweep mode
MANual:	Manual sweep mode
CONTinuous:	Continuous sweep mode

response: none

example:
```
wav:swe:mode STEP
```

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:MODE?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:MODE?

description: Returns the sweep mode.

parameters: none

response:

STEP:	Stepped sweep mode
MAN:	Manual sweep mode
CONT:	Continuous sweep mode

example: `wav:swe:mode?` → `STEP<END>`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:PMAX?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:PMAX?<wsp><start wavelength>, <stop wavelength>

description: Returns the power to the highest permissible power for the selected wavelength sweep.

parameters:

start wavelength:	The wavelength at which the sweep starts as a **float** value.
stop wavelength:	The wavelength at which the sweep starts as a **float** value.

response: The highest permissible power for the selected wavelength sweep as a **float** value.

example: `wav:swe:pmax? 1540,1550` → `+3.5500000E-004<END>`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat<wsp><mode>

description: Sets the repeat mode.

parameters:

ONEWay:	every sweep cycle starts at the start wavelength of the sweep and ends at the stop wavelength of the sweep
TWO-Way:	odd sweep cycles start at the start wavelength of the sweep and even sweep cycles start at the stop wavelength of the sweep

Set the start and stop wavelength of the sweep using *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STARt" on page 105* and *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STOP" on page 106* respectively.

response: none

example: `wav:swe:rep twow`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat?

description: Returns the repeat mode.

parameters: none

response: ONEW: every sweep cycle starts at the start wavelength of the sweep and ends at the stop wavelength of the sweep

TWOW: odd sweep cycles start at the start wavelength of the sweep and even sweep cycles start at the stop wavelength of the sweep

Set the start and stop wavelength of the sweep using *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STARt" on page 105* and *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STOP" on page 106* respectively.

example: `wav:swe:rep? → ONEW<END>`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed<wsp><speed>[NM/S|UM/S|MM/S|M/S|]

description: Sets the speed for continuous sweeping.

parameters: Speed as a **float** value in meters per second (m/s).

response: none

example: `wav:swe:spe 10nm/s`

affects: All tunable laser modules except HP 81689A

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed?

description: Returns the speed for continuous sweeping.

parameters: none

response: Speed as a **float** value in meters per second (m/s).

example: `wav:swe:spe? → +5.00000000E-008<END>`

affects: All tunable laser modules except HP 81689A

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt<wsp><start value>[PM|NM|UM|MM|M]

description: Sets the starting point of the sweep.

parameters: The wavelength at which the sweep starts as a **float** value.

If you specify no units in your command, meters are used as the default.

response: none

example: `wav:swe:star 1500nm`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt?

description: Returns the starting point of the sweep.

parameters: none

response: The wavelength at which the sweep starts as a **float** value in meters.

example: `wav:swe:star? → +1.50000000E-006<END>`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STOP**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STOP<wsp><stop value>[PM|NM|UM|MM|M]

description: Sets the end point of the sweep.

parameters: The wavelength at which the sweep ends as a **float** value in meters.

If you specify no units in your command, meters are used as the default.

response: none

example: `wav:swe:stop 1550nm`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STOP?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STOP?

description: Returns the end point of the sweep.

parameters: none

response: The wavelength at which the sweep ends as a **float** value in meters.

example: `wav:swe:stop? → +1.55000000E-006<END>`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:[STATe]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:[STATe]<wsp>STOP|0|STARt|1|PAUSe|2|CONTinue|3

description: Stops, starts, pauses or continues a wavelength sweep.

parameters:

0 or STOP:	Stop the sweep.
1 or STARt:	Start a sweep, run sweep.
2 or PAUSe:	Pause the sweep.
3 or CONTinue:	Continue a sweep.

NOTE When you enable lambda logging, see *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:LLOGging" on page 103*, and modulation, see *"[:SOURce[n]][:CHANnel[m]]:AM:STATe[1]" on page 90*, simultaneously, a sweep can not be started.

response: none

example: `wav:swe STOP`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:[STATe]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:[STATe]?

description: Returns the state of a sweep.

parameters: none

response: +0: Sweep is not running

+1: Sweep is running

example: `wav:swe?` → +0<END>

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:NEXT**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:NEXT

description: Performs the next sweep step, if a manual sweep is paused.

parameters: none

response: none

example: `wav:swe:step:next`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:PREVious**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:PREVious

description: Performs one sweep step backwards, if a manual sweep is paused.

parameters: none

response: none

example: `wav:swe:step:prev`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:[WIDTh]**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:[WIDTh]<wsp><value>[PM|NM|UM|MM|M]

description: Sets the width of the sweep step

parameters: The width of the sweep step as a ***float*** value.

If you specify no units in your command, meters are used as the default.

response: none

example: `wav:swe:step 5nm`

affects: All tunable laser modules

command: **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:[WIDTh]?**

syntax: [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:[WIDTh]?

description: Returns the width of the sweep step

parameters: none

response: The sweep step as a **float** value in meters.

example: `wav:swe:step?` → `+5.00000000E-009`<END>

affects: All tunable laser modules

Triggering - The TRIGger Subsystem

The TRIGger Subsystem allows you to configure how the instrument reacts to incoming or outgoing triggers.

Hardware Trigger	Trigger Rearming	Software Triggering		Data Acquisition Functions `sens:func:stat`	
`trig:inp`	`trig:inp:rearm`	`init:imm`	`init:cont`	`MINMax`	`LOGGing|STABility`
IGNore	-	One power measurement is performed.	Automatically performs power measurements.		Automatically performs power measurements until the function is finished.
SMEasure	ON	Every hardware trigger starts a new power measurement.			Every hardware trigger starts a new power measurement until the function is finished.
CMEasure	ON				The first hardware trigger starts the function. Subsequent power measurements are automatically performed until the function is finished.
SMEasure	OFF	The first hardware trigger starts a new power measurement. Further hardware triggers are ignored until you send `trig:inp:rearm` again.			Every hardware trigger starts a new power measurement until the function is finished.
CMEasure	OFF				The first hardware trigger starts the function. Subsequent power measurements are automatically performed until the function is finished.

Table 7 Triggering and Power Measurements

Hardware Trigger	Trigger Rearming	Software Triggering		Data Acquisition Functions sens:func:stat	
trig:outp	trig:outp:rearm	init:imm	init:cont	MINMax	LOGGing|STABility
DISabled	-	An output trigger will never be generated.			
AVGover	ON	An output trigger is generated for every new power measurement when the averaging time period finishes.			Applies for all subsequent data acquisition functions.
MEASure	ON	An output trigger is generated for every new power measurement when the averaging time period begins.			Applies for all subsequent data acquisition functions.
AVGover	OFF	An output trigger is generated when the averaging time period of the first power measurement finishes. A further hardware output trigger cannot be generated until you send `trig:outp:rearm`.			An output trigger is generated for every new power measurement when the averaging time period finishes. Applies for all subsequent data acquisition functions.
MEASure	OFF	An output trigger is generated when the averaging time period of the first power measurement begins. A further hardware output trigger cannot be generated until you send `trig:outp:rearm`.			An output trigger is generated for every new power measurement when the averaging time period begins. Applies for all subsequent data acquisition functions.

Table 8 Generating Output Triggers from Power Measurements

command: **:TRIGger**

syntax: :TRIGger<wsp>NODEA|1|NODEB|2

description: Generates a hardware trigger.

parameters: 1 or NODEA: Is identical to a trigger at the Input Trigger Connector.
2 or NODEB: Generates trigger at the Output Trigger Connector.

NOTE A hardware trigger cannot be effective in the DISabled triggering mode but can be effective in DEFault, PASSthrough or FEEDback triggering modes, see *":TRIGger:CONFiguration" on page 113* for information on triggering modes.

NOTE *":TRIGger" on page 115* describes the :TRIGger command for advanced users using *":TRIGger:CONFiguration:EXTended" on page 115*.

response: none

example:
```
trig 1
```

command: **:TRIGger[*n*][:CHANnel[*m*]]:INPut**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:INPut<wsp><trigger response>

description: Sets the incoming trigger response and arms the module.

parameters:

IGNore:	Ignore incoming trigger.
SMEasure:	Start a single measurement. If a measurement function is active, see *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77*, one sample is performed and the result is stored in the data array, see *":SENSe[n][:CHANnel[m]]:FUNCtion:RESult?" on page 77*.
CMEasure:	Start a complete measurement. If a measurement function is active, see *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77*, a complete measurement function is performed.
NEXTstep:	Perform next step of a stepped sweep.
SWStart:	Start a sweep cycle.

NOTE You must prearm a wavelength sweep or a measurement function before an action can be triggered:

First, set the incoming trigger response.

Then:

- prearm a wavelength sweep using *"[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:[STATe]" on page 106*. The wavelength of the tunable laser module is set to the start wavelength of the sweep.
- or prearm a measurement function using *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77*.
  NOTE: If a trigger signal arrives at the Input Trigger Connector at the same time that the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe command is executed, the first measurement value is invalid. You should always discard the first measurement value in this case.

The module performs the appropriate action when it is triggered.

response: none

example:
```
trig1:inp ign
```

affects: All tunable laser modules and HP 8163A Series power meters

NOTE If you use the HP 816x VXIplug&play Instrument Driver, you can trigger power measurements using HP 8153A Series power meters.

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:TRIGger[*n*][:CHANnel[*m*]]:INPut?**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:INPut?

description: Returns the incoming trigger response.

parameters: none

response:
- IGNore: Ignore incoming trigger.
- SMEasure: Start a single measurement. If a measurement function is active, see *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77*, one sample is performed and the result is stored in the data array, see *":SENSe[n][:CHANnel[m]]:FUNCtion:RESult?" on page 77*.
- CMEasure: Start a complete measurement. If a measurement function is active, see *":SENSe[n][:CHANnel[m]]:FUNCtion:STATe" on page 77*, a complete measurement function is performed.
- NEXTstep: Perform next step of a stepped sweep.
- SWStart: Start a sweep.

example: `trig1:inp? → ign<END>`

affects: All tunable laser modules and power meters

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:TRIGger[*n*][:CHANnel[*m*]]:INPut:REARm**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:INPut:REARm<wsp>OFF|ON|0|1

description: Sets the arming response of a channel to an incoming trigger.

NOTE See Table 7, for information on how this command affects triggering power measurements.

parameters: A ***boolean*** value: `OFF` or `0`: trigger rearming disabled; `ON` or `1`: trigger rearming enabled (default)

NOTE If you return to Local control, all modules return to the default setting.

response: none

example: `trig1:inp:rearm 0`

affects: All HP 8163A Series power meter modules

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:TRIGger[*n*][:CHANnel[*m*]]:INPut:REARm?**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:INPut:REARm?

description: Returns the arming response of a channel to an incoming trigger.

parameters: none

response: A ***boolean*** value: `0`: trigger rearming disabled; `1`: trigger rearming enabled (default)

example: `trig1:inp:rearm? → 0<END>`

affects: All HP 8163A Series power meter modules

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:TRIGger[*n*][:CHANnel[*m*]]:OUTPut**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:OUTPut

description: Specifies when an output trigger is generated and arms the module.

parameters:

DISabled:	Never.
AVGover:	When averaging time period finishes.
MEASure:	When averaging time period begins.
MODulation:	For every leading edge of a digitally-modulated (TTL) signal
STFinished:	When a sweep step finishes.
SWFinished:	When sweep cycle finishes.
SWSTarted:	When a sweep cycle starts.

response: none

example: `trig1:outp dis`

affects: All tunable laser modules and HP 8163A series power meters

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:TRIGger[*n*][:CHANnel[*m*]]:OUTPut?**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:OUTPut?

description: Returns the condition that causes an output trigger.

parameters: none

response:

DISabled:	Never.
AVGover:	When averaging time period finishes.
MEASure:	When averaging time period begins.
MODulation:	For every leading edge of a digitally-modulated (TTL) signal
STFinished:	When a sweep step finishes.
SWFinished:	When sweep cycle finishes.
SWSTarted:	When a sweep cycle starts.

example: `trig1:outp? → dis<END>`

affects: All tunable laser modules and HP 8163A series power meters

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:TRIGger[*n*][:CHANnel[*m*]]:OUTPut:REARm**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:OUTPut:REARm<wsp>OFF|ON|0|1

description: Sets the arming response of a channel to an outgoing trigger.

NOTE See Table 8, for information on how this command affects the generation of output triggers using power measurements.

parameters: A ***boolean*** value: `OFF` or `0`: trigger rearming disabled
`ON` or `1`: trigger rearming enabled (default)

NOTE If you return to Local control, all modules return to the default setting.

response: none

example: `trig1:outp:rearm 1`

affects: All HP 8163A series power meters

dual sensors: Can only be sent to master channel, slave channel is also affected.

command: **:TRIGger[*n*][:CHANnel[*m*]]:OUTPut:REARm?**

syntax: :TRIGger[*n*][:CHANnel[*m*]]:OUTPut:REARm?

description: Returns the arming response of a channel to an outgoing trigger.

parameters: none

response: A ***boolean*** value: `0`: trigger rearming disabled (default)
`1`: trigger rearming enabled.

example: `trig1:outp:rearm?` → `0<END>`

affects: All HP 8163A series power meters

dual sensors: Can only be sent to master channel, slave channel parameters are identical.

command: **:TRIGger:CONFiguration**

syntax: :TRIGger:CONFiguration<wsp><triggering mode>

description: Sets the hardware trigger configuration with regard to Output and Input Trigger Connectors.

parameters:

0 or DISabled:	Trigger connectors are disabled.
1 or DEFault:	The Input Trigger Connector is activated, the incoming trigger response for each slot *":TRIGger[n][:CHANnel[m]]:INPut" on page 110* determines how each slot responds to an incoming trigger, all slot events (see *":TRIGger[n][:CHANnel[m]]:OUTPut" on page 112*) can trigger the Output Trigger Connector.
2 or PASSthrough:	The same as DEFault but a trigger at the Input Trigger Connector generates a trigger at the Output Trigger Connector automatically.
3 or LOOPback:	The same as DEFault but a trigger at the Output Trigger Connector generates a trigger at the Input Trigger Connector automatically.

response: none

example: `trig:conf dis`

command: **:TRIGger:CONFiguration?**

syntax: :TRIGger:CONFiguration?

description: Returns the hardware trigger configuration.

parameters: none

response:

DIS: Trigger connectors are disabled.

DEF: The Input Trigger Connector is activated, the incoming trigger response for each slot *":TRIGger[n][:CHANnel[m]]:INPut" on page 110* determines how each slot responds to an incoming trigger, all slot events (see *":TRIGger[n][:CHANnel[m]]:OUTPut" on page 112*) can trigger the Output Trigger Connector.

PASS: The same as DEFault but a trigger at the Input Trigger Connector generates a trigger at the Output Trigger Connector automatically.

LOOP: The same as DEFault but a trigger at the Output Trigger Connector generates a trigger at the Input Trigger Connector automatically.

CUSTOM: A custom configuration is active using either the command *":TRIGger:CONFiguration:EXTended" on page 115* or the HP 816x VXI-*plug&play* Instrument Driver. See Appendix A "The HP 816x VXI-plug&play Instrument Driver".

example: `trig:conf? → DEF<END>`

command: **:TRIGger:CONFiguration:FPEDal**

syntax: :TRIGger:CONFiguration:FPEDal<wsp>OFF|ON|0|1

description: Enables or disables the Input Trigger connector to be triggered using a Foot Pedal.

parameters: A ***boolean*** value: `OFF` or `0`: foot pedal disabled (default)
`ON` or `1`: foot pedal enabled

response: none

example: `trig:conf? → DEF<END>`

command: **:TRIGger:CONFiguration:FPEDal?**

syntax: :TRIGger:CONFiguration:FPEDal?

description: Returns whether the Input Trigger connector can be triggered using a Foot Pedal.

parameters: none

response: A ***boolean*** value: `0`: foot pedal disabled
`1`: foot pedal enabled

example: `trig:conf? → DEF<END>`

Extended Trigger Configuration

This section includes information for advanced users about how to customize your use of the trigger system.

You can configure the ouputs and inputs from two nodes, Node A and Node B. See Figure 6 on page 116 for more information on Node A and Node B. You can

configure these nodes to be triggered by certain events and for these nodes to trigger particular actions.

command: **:TRIGger**

syntax: :TRIGger<wsp>NODEA|1|NODEB|2

description: Generates a hardware trigger.

parameters:

1 or NODEA:	Generates trigger at Node A.
2 or NODEB:	Generates trigger at Node B.

Use *":TRIGger:CONFiguration:EXTended" on page 115* to configure Node A and Node B.

NOTE *":TRIGger" on page 109* describes the :TRIGger command for basic users.

response: none

example:
```
trig 1
```

command: **:TRIGger:CONFiguration:EXTended**

syntax: :TRIGger:CONFiguration:EXTended<wsp><Node A Input Config.>, <Node B Input Config.>,<Output Matrix Config.>

description: Sets the extended hardware trigger configuration.

parameters:

Node A Input Configuration:	A **32-bit unsigned integer**, see below.
Node B Input Configuration:	A **32-bit unsigned integer**, see below.
Output Matrix Configuration:	A **32-bit unsigned integer**, see below.

response: none

example:
```
trig:conf:ext 0,0,0
```

command: **:TRIGger:CONFiguration:EXTended?**

syntax: :TRIGger:CONFiguration:EXTended?

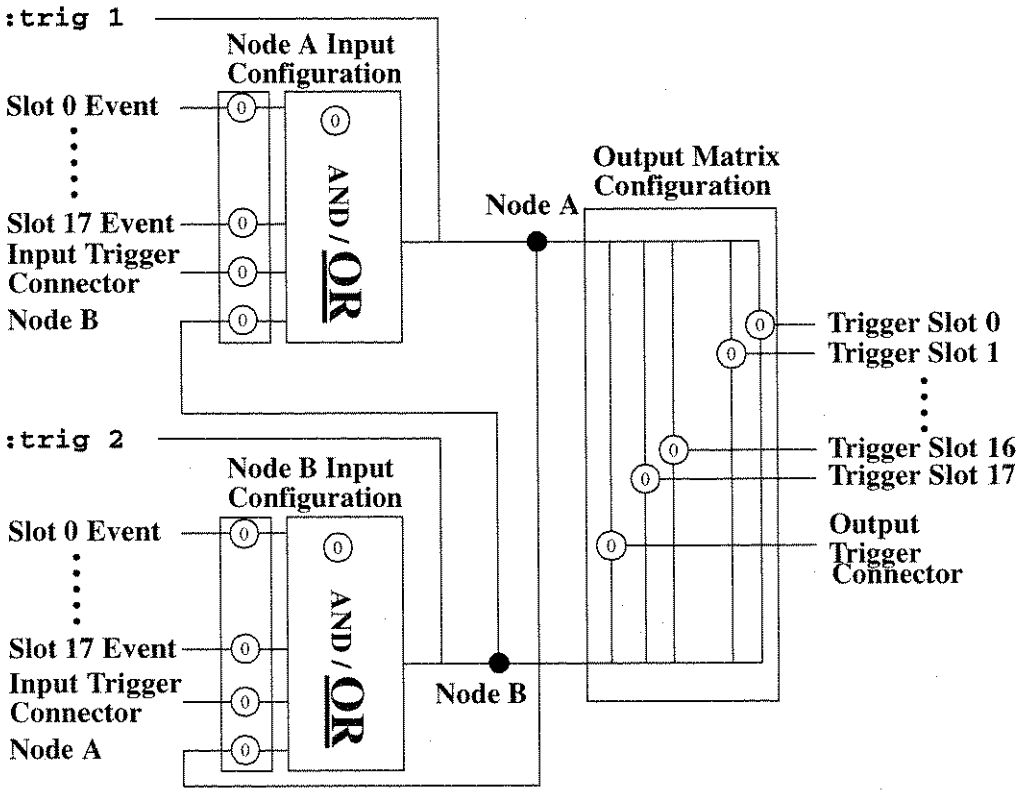
description: Returns the extended hardware trigger configuration.

parameters: none

response:

Node A Input Configuration:	A **32-bit unsigned integer**, see below.
Node B Input Configuration:	A **32-bit unsigned integer**, see below.
Output Matrix Configuration:	A **32-bit unsigned integer**, see below.

example:
```
trig:conf:ext? → +0,+0,+0<END>
```



Bits set in Node A/B Input Configuration determine the conditions that can cause a trigger at Node A/B.

Bits set in Output Matrix Configuration determine whether Node A OR Node B triggers particular module slots or generates an output trigger at the Output Trigger Connector.

":TRIGger[*n*][:CHANnel[*m*]]:OUTPut" explains how slot events can generate triggers.

":TRIGger[*n*][:CHANnel[*m*]]:INPut" explains how a slot responds to an incoming trigger.

":TRIGger" generates a trigger at Node A or Node B directly.

Figure 6 Extended Trigger Configuration

Node A Input Configuration

This **32-bit unsigned integer** determines how inputs to Node A are generated.

Bit	Mnemonic	Hexadecimal
31	Logic: 0 for OR, 1 for AND	#H80000000
30	Input Trigger Connector: 0 - Inactive, 1 - Trigger at Input Trigger Connector can trigger Node A	#H40000000
29	Node B: 0 - Inactive, 1 - Trigger at Node B can trigger Node A	#H20000000
18-28	Not used.	0
17	Slot 17: 0 - Inactive, 1 - Event at slot 17 can trigger Node A	#H20000
16	Slot 16: 0 - Inactive, 1 - Event at slot 16 can trigger Node A	#H10000
⋮	⋮	
2	Slot 2: 0 - Inactive, 1 - Event at slot 2 can trigger Node A	#H4
1	Slot 1: 0 - Inactive, 1 - Event at slot 1 can trigger Node A	#H2
0	Slot 0: 0 - Inactive, 1 - Event at slot 0 can trigger Node A	#H1

":TRIGger[n][:CHANnel[m]]:OUTPut" on page 112 explains how slot events can generate triggers.

Node B Input Configuration

This **32-bit unsigned integer** determines how inputs to Node B are generated.

Bit	Mnemonic	Hexadecimal
31	Logic: 0 for OR, 1 for AND	#H80000000
30	Input Trigger Connector: 0 - Inactive, 1 - Trigger at Input Trigger Connector can trigger Node B	#H40000000
29	Node A: 0 - Inactive, 1 - Trigger at Node A can trigger Node B	#H20000000
18-28	Not used.	0
17	Slot 17: 0 - Inactive, 1 - Event at slot 17 can trigger Node B	#H20000
16	Slot 16: 0 - Inactive, 1 - Event at slot 16 can trigger Node B	#H10000
⋮	⋮	
2	Slot 2: 0 - Inactive, 1 - Event at slot 2 can trigger Node B	#H4
1	Slot 1: 0 - Inactive, 1 - Event at slot 1 can trigger Node B	#H2
0	Slot 0: 0 - Inactive, 1 - Event at slot 0 can trigger Node B	#H1

":TRIGger[n][:CHANnel[m]]:OUTPut" on page 112 explains how slot events can generate triggers.

Output Matrix Configuration

This **32-bit unsigned integer** lets you choose Node A OR Node B to trigger each of the following:

- the Output Trigger Connector or
- individual module slots.

Bit	Mnemonic	Hexadecimal
31	Not used	0
30	Output Trigger Connector: 0 - a trigger at Node A is switched to the Output Trigger Connector, 1 - a trigger at Node B is switched to theOutput Trigger Connector	#H40000000
18-29	Not used	
17	Slot 17: 0 - Node A triggers slot 17, 1 - Node B triggers slot 17	0
16	Slot 16: 0 - Node A triggers slot 16, 1 - Node B triggers slot 16	#H20000
⋮	⋮	#H10000
2	Slot 2: 0 - Node A triggers slot 2, 1 - Node B triggers slot 2	
1	Slot 1: 0 - Node A triggers slot 1, 1 - Node B triggers slot 1	#H4
0	Slot 0: 0 - Node A triggers slot 0, 1 - Node B triggers slot 0	#H2
	":TRIGger[n][:CHANnel[m]]:INPut" on page 110 explains how a slot responds to an incoming trigger.	#H1

Extended Trigger Configuration Example

The short example below demonstrates how to use extended triggering configuration to make tunable laser source modules sweep simultaneously. Setup your mainframe with two HP 81689A modules in slots 1 and 2. The example below presumes you set up identical stepped sweeps for both modules, for example, by pressing PRESET.

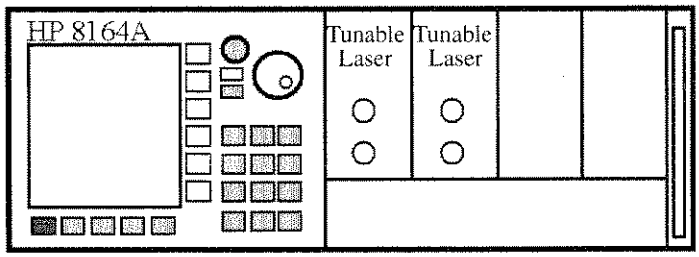


Figure 7 Setup for Extended Trigger Configuration Example

```
trig:conf:ext #H2,#H0,#H0
trig2:outp dis
trig2:inp next
sour2:wav:swe star
trig1:outp stf
trig1:inp ign
sour1:wav:swe star
```

`trig:conf:ext #H2,#H0,#H0` is described by Figure 4-1 and sets one bit:

- for Node A Input Configuration:
  - Bit 1 - an event at slot 1 can trigger Node A. As `trig1:outp stf` is set, Node A can be triggered if a sweep step finishes for a tunable laser module installed in slot 1.

The following explanation explains the sequence with which actions are triggered.

1 `sour2:wav:swe star` arms the sweep for for the tunable laser module in slot 2. Because `trig2:inp next` is set, the module waits for a trigger until it performs the first step of the sweep.

2 `sour1:wav:swe star` commands the tunable laser module in slot 1 to start sweeping. Because `trig1:inp ign` is set, the module performs a sweep as normal.

3 When the module in slot 1 finishes a step, because `trig1:outp stf` is set, Node A is triggered.

4 Node A triggers all modules because the Output Matrix Configuration is set to zero. Node A triggers the tunable laser module in slot 2 to perform a sweep step because `trig2:inp next` is set.

5 The sequence starts again at step 3 and continues until the sweep ends.

Mass Storage, Display, and Print Functions

This chapter gives descriptions of commands that you can use when you want to change the instrument's display.

Display Operations – The DISPlay Subsystem

The DISPlay subsystem lets you control what you see on the instrument's display.

command: **:DISPlay:BRIGhtness**
syntax: :DISPlay:BRIGhtness<wsp><value>
description: Controls the brightness for the display.
parameters: An **integer** value in the range 0 to 100
response: none
example: `disp:brig 75`
affects: HP 8163A Lightwave Multimeter

command: **:DISPlay:BRIGhtness?**
syntax: :DISPlay:BRIGhtness?
description: Requests the brightness for the display.
parameters: none
response: An **integer** value in the range 0 to 100
example: `disp:brig? → +75<END>`
affects: HP 8163A Lightwave Multimeter

command: **:DISPlay:CONTrast**
syntax: :DISPlay:CONTrast<wsp><value>
description: Controls the contrast for the display.
parameters: An **integer** value in the range 0 to 100
response: none
example: `disp:cont 50`
affects: HP 8163A Lightwave Multimeter

command: **:DISPlay:CONTrast?**
syntax: :DISPlay:CONTrast?
description: Requests the contrast for the display.
parameters: none
response: An **integer** value in the range 0 to 100
example: `disp:cont? → +50<END>`
affects: HP 8163A Lightwave Multimeter

command: **:DISPlay:ENABle**

syntax: :DISPlay:ENABle<wsp><boolean>

description: Enables or disables the display.

parameters: A ***boolean*** value: 0 – switch off the display
1 – switch on the display

NOTE If you press [LOCAL] softkey, the display is enabled automatically.

response: none

example:
```
disp:enab 1
```

command: **:DISPlay:ENABle?**

syntax: :DISPlay:ENABle?

description: Queries the state of the display.

parameters: none

response: A b***oolean*** value: 0 – the display is turned off
1 – the display is turned on

example:
```
disp:enab? → 1<END>
```

Programming Examples

These programming examples are implemented using MS Developer Studio. Regardless of the programming environment you use, keep the following in mind:

- The resultant application is a "console application"
- Make sure the include path spans `visa.h` and `visatype.h`
- Make sure the library path setting includes `visa32.lib`
- Assure that the `PATH` environment variable allows loading `visa32.dll`.

The programming examples do not cover the full command set for the instruments. They are intended only as an introduction, how to program the instrument using VISA library calls.

The VISA calls used, are explained in detail in the VISA User's Guide.

How to Use VISA Calls

The following example demonstrates how to communicate using VISA calls. Also, the use of instrument identification commands is demonstrated.

```
#include <stdio.h>
#include <stdlib.h>
#include <visa.h>

/* This function checks and displays errors, using the error query of the instrument;
Call this function after every command to make sure your commands are correct */

void checkError(ViSession session, ViStatus err_status )
 {
  ViStatus error;
  ViChar errMsg[256];
    /* queries what kind of error occurred */
    error = viQueryf(session,"%s\n","%t","SYST:ERR?",errMsg);
    /*if this command times out, a system error is probable;
      check the HPIB bus communication */
    if (error == VI_ERROR_TMO)
      {
      printf("System Error!\n") ;
      exit(1);
      }
    else
      {
      /* display the error number and the error message */
      if(errMsg[0] != '+')
      printf("error:%ld --> %s\n", err_status,errMsg) ;
      }

 }



void main (void)
 {

 ViStatus     errStatus;    /*return error code from visa call */
    ViSession      defaultRM;    /*default visa resource manager variable*
/
    ViSession      vi;          /*current session handle */
 ViChar            replyBuf[256]; /*buffer holding answers from the instrument*/
    ViChar         c;
  /* Initialize visa resource manger */
      errStatus = viOpenDefaultRM (&defaultRM);
  if(errStatus < VI_SUCCESS)
```

```
 {
  printf("Failed to open VISA Resource manager\n");
  exit(errStatus);
 }

/* Open session to HPIB device at address 20; the VI_NULL parameters 3,4
  are mandatory and not used for VISA 1.0*/
errStatus = viOpen (defaultRM, "GPIB::20::INSTR", VI_NULL,VI_NULL,&vi);
if(errStatus < VI_SUCCESS)
 {
  printf("Failed to open instrument\n");
  exit(errStatus);
 }

/*
 set timeout to 20 sec; this should work for all commands except for zeroing or
 READ commands with averaging times greater than the timeout;
*/
errStatus = viSetAttribute(vi,VI_ATTR_TMO_VALUE,20000);
checkError(vi,errStatus);


/* get the identification string of the instrument mainframe*/
errStatus = viQueryf(vi,"%s\n","%t","*IDN?",replyBuf);
if(errStatus < VI_SUCCESS)
 {
  checkError(vi,errStatus);
 }
else printf("%s",replyBuf);

/* identify the installed modules */
errStatus = viQueryf(vi,"%s\n","%t","*OPT?",replyBuf);
if(errStatus < VI_SUCCESS)
 {
  checkError(vi,errStatus);
 }
else printf("%s",replyBuf);

/* get information about the available options of a slot */
errStatus = viQueryf(vi,"%s","%t","SLOT1:OPT?\n",replyBuf);
if(errStatus < VI_SUCCESS)
 {
  checkError(vi,errStatus);
 }
else printf("%s",replyBuf);


/*loop, until a key is pressed */
while(!scanf("%c",&c));
/*close the session */
viClose(vi);
```

```
}
```

How to Set up a Fixed Laser Source

This example sets up a fixed laser source.

Install a Laser Source in Slot 2, before executing this example.

```
#include <stdio.h>
#include <stdlib.h>
#include <visa.h>

/* function prototypes for this examples */

/* function for simple error handling explained in example 1 */
void checkError(ViSession session, ViStatus err_status );




void main (void)
 {

 ViStatus     errStatus;    /* returned error code from visa call */
     ViSession      defaultRM;    /* default visa resource manager variable*
/
     ViSession      vi;          /* current session handle */
 ViChar             c;          /* used in the keyboard wait loop */
 ViReal32  wavelength;   /* wavelength of the laser source */

  /* initialize the visa library (see example 1) */
  errStatus = viOpenDefaultRM (&defaultRM);
  if(errStatus < VI_SUCCESS)
   {
    printf("Failed to open VISA Resource manager\n");
    exit(errStatus);
   }

  /* Open session to HPIB device at address 20;*/
  errStatus = viOpen (defaultRM, "GPIB::20::INSTR", VI_NULL,VI_NULL,&vi);
```

```
if(errStatus < VI_SUCCESS)
 {
  printf("Failed to open instrument\n");
  exit(errStatus);
 }

/*set timeout to 20 sec; this should work for all commands except zeroing */
errStatus = viSetAttribute(vi,VI_ATTR_TMO_VALUE,20000);
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* first get the wavelength of the laser source; to address the second channel
  of a dual laser source use "CHAN2" instead of "CHAN1"*/
errStatus = viQueryf(vi,"%s","%f","SOURCE2:CHAN1:WAV?\n",&wavelength);
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);
else
 {
 printf("Source Wavelength:%g\n",wavelength);
 }

/* to receive the maximum power the attenuation must be set to zero */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:ATT 0\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* turn off amplitude modulation */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:AM:STATE 0\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* turn the laser on */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:STATE 1\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* loop, until a key is pressed */

while(!scanf("%c",&c));


/* turn the laser off */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:STATE 0\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* close the session */
viClose(vi);


}


void checkError(ViSession session, ViStatus err_status )
 {
  ViStatus error;
  ViChar errMsg[256];
```

```
    error = viQueryf(session,"SYST:ERR?\n","%t",errMsg);
    if (error == VI_ERROR_TMO)
      {
      printf("System Error!\n") ;
      exit(1);
      }
    else
      {
      /* only errors should be displayed */
      if(errMsg[0] != '+')
      printf("error:%ld --> %s\n", err_status,errMsg) ;
      }

}
```

How to Measure Power using FETCh and READ

The example shows the difference between a "FETCh" and a "READ" command.

Install a power meter in Slot 1, before executing this example.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <visa.h>

/* function prototypes for this examples */

/* function for a simple error handling explained in example 1 */
void checkError(ViSession session, ViStatus err_status );


void main (void)
 {

 ViStatus     errStatus:    /* returned error code from visa call */
    ViSession     defaultRM;    /* default visa resource manager variable */
    ViSession     vi;           /* current session handle */
 ViChar           replyBuf[256]; /* buffer holding answers of the instrument*/
    ViChar        compBuf[256];  /* buffer used for comparsion */
    ViChar        c;            /* used in the keyboard wait loop */
 ViReal64  averagingTime;  /* averaging time */
 ViInt32   i;           /* loop counter */
```

```
errStatus = viOpenDefaultRM (&defaultRM);
if(errStatus < VI_SUCCESS)
 {
  printf("Failed to open VISA Resource manager\n");
  exit(errStatus);
 }

errStatus = viOpen (defaultRM, "GPIB::20::INSTR", VI_NULL,VI_NULL,&vi);
if(errStatus < VI_SUCCESS)
 {
  printf("Failed to open instrument\n");
  exit(errStatus);
 }


/*set timeout to 20 sec; this should work for all commands
 except zeroing */
errStatus = viSetAttribute(vi,VI_ATTR_TMO_VALUE,20000);
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* make sure that the reference is not used */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:REF:STATE 0\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* clear the error queue */
errStatus = viPrintf(vi,"*CLS\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* turn auto range on */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:RANGE:AUTO 1\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* change the power unit to watt */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:UNIT W\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/*set the averaging time for measuring to 0.5s*/
averagingTime = 0.5;

errStatus = viPrintf(vi,"SENS1:CHAN1:POW:ATIME %f\n",averagingTime);
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/* turn continous measuring off */
errStatus = viPrintf(vi,"INIT1:CHAN1:CONT 0\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);


/* trigger a measurement */
errStatus = viPrintf(vi,"INIT1:CHAN1:IMM\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);
```

```
/* read 10 values and display the result; */
for (i = 0; i < 10; i++)
 {
  /* Now because an averaged value is available, the value will be fetched*/
  errStatus = viQueryf(vi,"%s","%s","FETCH1:CHAN1:POW?\n",replyBuf);
  if (errStatus < VI_SUCCESS) checkError(vi,errStatus);
  /* two consecutive values are compared; if they are equal it will be marked;
     because no evaluation is triggered, all values will be the same*/
  if(i)
   {
   if(!strcmp(compBuf,replyBuf))
    {
    printf("Same:%s\n",replyBuf);
    }
   else printf("New:%s\n",replyBuf);
   }
  else printf("First:%s\n",replyBuf);
  strcpy(compBuf,replyBuf);
 }
/* now the read command is used in the same manner to
  demonstrate the difference between fetch and read*/

/* read also 10 values, compare them and display the result; */
 for (i = 0; i < 10; i++)
 {
 /*
  In comparision to the "FETCH" command, the "READ" command implies
  triggering a measurement.
  Make sure the timeout set is greater than the adjusted
  averaging time, so that the READ command will not time out.
 */

  /* send the read command */
  errStatus = viQueryf(vi,"READ1:CHAN1:POW?\n","%t",replyBuf);
  checkError(vi,errStatus);

  if(i)
   {
   if(!strcmp(compBuf,replyBuf))  printf("Same:%s",replyBuf);
   else printf("New  :%s",replyBuf);
   }
  else printf("\nFirst:%s",replyBuf);
  /*copy new value to compare buffer*/
  strcpy(compBuf,replyBuf);
 }


/* loop, until a key is pressed */
```

```
  while(!scanf("%c",&c));


  checkError(vi,errStatus);
  /* close the session */
  viClose(vi);


}


void checkError(ViSession session, ViStatus err_status )
 {
  ViStatus error;
  ViChar errMsg[256];
    error = viQueryf(session,"SYST:ERR?\n","%t",errMsg);
    if (error == VI_ERROR_TMO)
     {
     printf("System Error!\n") ;
     exit(1);
     }
    else
     {
     /* only errors should be displayed */
     if(errMsg[0] != '+')
     printf("error:%ld --> %s\n", err_status,errMsg) ;
     }

 }
```

How to Co-ordinate Two Modules

This example shows the interaction of two modules in the same frame.

Install a Power Sensor in Slot 1 and a Laser Source in Slot 2 and connect the Laser Source output to the Power Sensor input, before executing this example.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <visa.h>


/* function prototypes for this examples */

/* function for a simple error handling explained in example 1 */
void checkError(ViSession session, ViStatus err_status );



void main (void)
 {

 ViStatus     errStatus;    /* returned error code from visa call */
     ViSession      defaultRM;    /* default visa resource manager variable */
     ViSession      vi;           /* current session handle */
 ViChar             replyBuf[256]; /* buffer holding answers of the instrument */
     ViChar         c;            /* used in the keyboard wait loop */
 ViInt32   i;           /* loop counter */
 ViInt32   cmdDone;     /* return value for OPC command */

  /* First get initialized the visa library (see example 1) */
  errStatus = viOpenDefaultRM (&defaultRM);
  if(errStatus < VI_SUCCESS)
    {
     printf("Failed to open VISA Resource manager\n");
     exit(errStatus);
    }

  /* Open session to HPIB device at address 20: */
  errStatus = viOpen (defaultRM, "GPIB::20::INSTR", VI_NULL,VI_NULL,&vi);
  if(errStatus < VI_SUCCESS)
    {
     printf("Failed to open instrument\n");
     exit(errStatus);
    }



  /* set timeout to 20 sec; this should work for all commands except zeroing */
  errStatus = viSetAttribute(vi,VI_ATTR_TMO_VALUE,20000);
  if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

  /* clear error queue */
```

```
errStatus = viPrintf(vi,"*CLS\n");
checkError(vi,errStatus);

/* read the wavelength from the laser source */
errStatus = viQueryf(vi,"SOURCE2:CHAN1:WAV?\n","%s",replyBuf);
checkError(vi,errStatus);

/* feed the source wavelength into the power meter making
  sure to measure the maximum power of the source */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:WAV %s\n",replyBuf);
checkError(vi,errStatus);

/* turn auto range on */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:RANGE:AUTO 1\n");
checkError(vi,errStatus);

/* change the power unit of the power meter to dBm */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:UNIT 0\n");
checkError(vi,errStatus);

/*set the averaging time for measuring to 20 ms,
  therefore no timeout needs to implemented */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:ATIME 0.02\n");
checkError(vi,errStatus);

/* set the attenuation to zero for maximum power */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:ATT 0.0\n");
checkError(vi,errStatus);

/* set the reference mode to the internal one,
    which is now the last displayed value  */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:REF:STATE:RATIO TOREF,0\n");
checkError(vi,errStatus);

/* set  reference measuremant state to absolute units */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:REF:STAT 1\n");
checkError(vi,errStatus);

/* turn laser on */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:STATE 1\n");
checkError(vi,errStatus);
/*ask for command completion */
 do
    {
     errStatus = viQueryf(vi,"*OPC?\n","%d",&cmdDone);
     checkError(vi,errStatus);
    } while (!cmdDone);


/* set the power meter reference to the displayed value (display to reference) */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:REF:DISP\n");
```

```
checkError(vi,errStatus);


/*
  read 30 values and display the result; after ten measurements
  the source output will be halved by making use of the attenuation;
  after an other ten measurements the source output will be halved
  a second time;
  because of the display to reference command and using the
  reference, the value printed should be more or less equal to the
  adjusted source attenuation  */

for (i = 1; i <= 30; i++)
 {
  errStatus = viQueryf(vi,"READ1:CHAN1:POW?\n","%s",replyBuf);
  checkError(vi,errStatus);
  if(errStatus ==VI_SUCCESS)printf("power #%02d:%s\n",i,replyBuf);
  if(i == 10)
    {
    /* reduce the output power for 3.0 dB */
    errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:ATT 3.0\n");
    checkError(vi,errStatus);
    }
  if(i == 20)
    {
    /* reduce the output power for 6.0 dB */
    errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:ATT 6.0\n");
    checkError(vi,errStatus);
    }


 }



/* loop, until a key is pressed */
while(!scanf("%c",&c));

/* turn the laser off */
errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:STATE 0\n");
if (errStatus < VI_SUCCESS) checkError(vi,errStatus);

/*close the session */
viClose(vi);


}


void checkError(ViSession session, ViStatus err_status )
 {
```

```
  ViStatus error;
  ViChar errMsg[256];
    error = viQueryf(session,"SYST:ERR?\n","%t",errMsg);
    if (error == VI_ERROR_TMO)
      {
      printf("System Error!\n") ;
      exit(1);
      }
    else
      {
      /* only errors should be displayed */
      if(errMsg[0] != '+')
      printf("error:%ld --> %s\n", err_status,errMsg) ;
      }

}
```

How Power Varies with Wavelength

This example shows how the measured power depends on wavelength.

Install a Power Sensor in Slot 1 and a Tunable Laser Source in Slot 2 and connect the Tunable Laser Source output to the Power Sensor input, before executing this example.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <visa.h>

/* function prototypes for this examples*/

/* function for a simple error handling explained in example 1 */
void checkError(ViSession session, ViStatus err_status );



void main (void)
 {

 ViStatus     errStatus;    /* returned error code from visa call */
    ViSession     defaultRM;    /* default visa resource manager variable*
/
```

```
    ViSession      vi;          /* current session handle */
  ViChar           replyBuf[256]; /*buffer holding answers of the instrument */
    ViChar         c;           /* used in the keyboard wait loop */
  ViReal64  wavelength;   /* used to hold the wavelength of the tunable laser source */
  ViReal64  wavelength_max; /*used to hold the maximum wavelength of the tunable
laser source*/
  ViInt32   i;           /* loop counter */
  ViInt32   cmdDone;     /* return value for OPC command */

  errStatus = viOpenDefaultRM (&defaultRM);
  if(errStatus < VI_SUCCESS)
   {
    printf("Failed to open VISA Resource manager\n");
    exit(errStatus);
   }

  errStatus = viOpen (defaultRM, "GPIB::20::INSTR", VI_NULL,VI_NULL,&vi);
  if(errStatus < VI_SUCCESS)
   {
    printf("Failed to open instrument\n");
    exit(errStatus);
   }


  /*set timeout to 20 sec; this should work for all commands
    except zeroing */
  errStatus = viSetAttribute(vi,VI_ATTR_TMO_VALUE,20000);
  checkError(vi,errStatus);


  errStatus = viPrintf(vi,"*CLS\n");
  checkError(vi,errStatus);

  /* read the minimum wavelength from the tunable laser source*/
  errStatus = viQueryf(vi,"SOURCE2:WAV? MIN\n","%s",replyBuf);
  checkError(vi,errStatus);

  /* save this wavelength */
  wavelength = atof(replyBuf);

  /* set the minimum wavelength as initial wavelength in the tunable laser source */
  errStatus = viPrintf(vi,"SOURCE2:WAV %s\n",replyBuf);
  checkError(vi,errStatus);

  /* set the power meter to same wavelength like the tunable laser source */
  errStatus = viPrintf(vi,"SENS1:CHAN1:POW:WAV %s\n",replyBuf);
  checkError(vi,errStatus);


  /* read the maximum wavelength from the tunable laser source */
  errStatus = viQueryf(vi,"SOURCE2:WAV? MAX\n","%s",replyBuf);
```

```
checkError(vi,errStatus);

/*save this wavelength */
wavelength_max = atof(replyBuf);

/* change the power unit of the power meter to dbm  */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:UNIT DBM\n");
checkError(vi,errStatus);

/* read the default power from the tunable laser source */
errStatus = viQueryf(vi,"SOURCE2:POW? DEF\n","%s",replyBuf);
checkError(vi,errStatus);

/* set the default power */
errStatus = viPrintf(vi,"SOURCE2:POW %s\n",replyBuf);
checkError(vi,errStatus);


/* turn auto range on*/
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:RANGE:AUTO 1\n");
checkError(vi,errStatus);



/*set the averaging time for measuring to 20ms*/
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:ATIME 0.02\n");
checkError(vi,errStatus);


/* turn laser on */
errStatus = viPrintf(vi,"SOURCE2:POW:STATE 1\n");
checkError(vi,errStatus);


/* increase the wavelength of the tunable laser source 10 nm
   until the maximum is reached.
  read the results from the power meter and display it */

for(i=1;1;i++)
  {
   /*query the power */
   errStatus = viQueryf(vi,"READ1:CHAN1:POW?\n","%s",replyBuf);
   checkError(vi,errStatus);

   /* display the power read from power meter and wavelength */
   printf("#%02d power:%s   wavelength:%g\n",i,replyBuf,wavelength);

   /* increase the wavelength */
   wavelength += 10.0e-9;
   if(wavelength > wavelength_max) break;
   /*set the new wavelength*/
```

```
    errStatus = viPrintf(vi,"SOURCE2:WAV %g\n",wavelength);

    /*
    poll the instrument for completion of this command
    because adjusting a new wavelength takes some time
    */
    do
     {
      errStatus = viQueryf(vi,"*OPC?\n","%d",&cmdDone);
      checkError(vi,errStatus);
     } while (!cmdDone);

   }


  /* loop, until a key is pressed */
  while(!scanf("%c",&c));

  /* turn laser off */
  errStatus = viPrintf(vi,"SOURCE2:CHAN1:POW:STATE 0\n");
  checkError(vi,errStatus);

  /* close the session */
  viClose(vi);


}


void checkError(ViSession session, ViStatus err_status )
 {
  ViStatus error;
  ViChar errMsg[256];
    error = viQueryf(session,"SYST:ERR?\n","%t",errMsg);
    if (error == VI_ERROR_TMO) printf("System Error!\n") ;
    else
     {
      /* only errors should be displayed */
      if(errMsg[0] != '+')
      printf("error:%ld --> %s\n", err_status,errMsg) ;
     }

 }
```

How to Log Results

This example demonstrates how to use logging functions.

Install a Power Sensor in Slot 1, before executing this example.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <visa.h>

#define MAX_LOG_VALUES 4000 /* max number of values the instrument is capable to
log */
#define HEADER_SIZE   7   /* includes 6 bytes header and 1 CR */

/* function prototypes for this examples/*

/* function for a simple error handling explained in example 1 */
void     checkError(ViStatus session, ViStatus err_status );

/* initialize the visa interface */
ViStatus  InitVisa ( ViSession *iHandle);

/*globals*/
static unsigned char logBuffer[MAX_LOG_VALUES * sizeof(ViReal64) + HEADER_SIZE];
/*array for the float results */
static ViReal32 logResults[MAX_LOG_VALUES];

void main (void)
 {

  ViStatus     errStatus;    /* returned error code from visa call */
  ViSession    vi;           /* current session handle */
  ViChar             replyBuf[256]; /* buffer holding answers of the instrument */
  ViChar             c;            /* used in the keyboard wait loop */
  ViInt32    slot;         /* slot number where the power meter is plugged */
  ViInt32    chan;         /* channel to be logged */
  ViInt32    i;           /* loop counter */
  ViInt32      noOfValues;    /* number of values to be logged*/
  ViReal64     averagingTime; /* aveaging time used in a logging cycle */
  ViPChar      replySubStr;   /* pointer to a substring of the instruments reply */
  ViInt32    noOfDigits;    /*number of digits, specifing the amount data
                     to be read */
  ViUInt32   retCnt;       /* returns the number of bytes read calling viRead */

  errStatus = InitVisa(&vi);

  if(errStatus < VI_SUCCESS)
```

```
 {
 exit(errStatus);
 }

/* clear instrument error queue */
errStatus = viPrintf(vi,"*CLS\n");
checkError(vi,errStatus);

/* turn auto range on */
errStatus = viPrintf(vi,"SENS1:CHAN1:POW:RANGE:AUTO 1\n");
checkError(vi,errStatus);

/* send the command sequence for continuous logging */
slot = 1;
chan = 1;
noOfValues = 100;    /* log 100 values */
averagingTime = 0.02; /* set averaging time to 20ms */
viPrintf(vi,"SENS%1d:CHAN%1d:FUNC:PAR:LOGG %d,%f\n",
      slot,
      chan,
      noOfValues,
      averagingTime);
        checkError(vi,errStatus);

/* start logging */
   viPrintf(vi,"SENS%1d:CHAN%1d:FUNC:STAT LOGG,START\n",slot,chan);
        checkError(vi,errStatus);
/* to display the results, logging should be completed */
/* the instrument has to be polled about the progress of the logging */
do
  {
   errStatus = viQueryf(vi,"SENS%1d:CHAN%1d:FUNC:STATE?\n","%t",slot,chan,r
eplyBuf);
   /* if an error occurs break the loop */
   if (errStatus < VI_SUCCESS)
    {
    checkError(vi,errStatus);
    break;
    }

   /* find the substring "COMPLETE" in the reply of the instrument */

   replySubStr = replyBuf;
         while(*replySubStr)
    {
     if(!strncmp(replySubStr,"COMPLETE",strlen("COMPLETE"))) break;
     replySubStr ++;
    }
  }while (!*replySubStr); /*substring "COMPLETE" not found */
            /*continue polling */
```

```
  /* The instrument returns the logging result in the following format:
  #xyyyffff...; the first digits after the hash denotes the
  number of ascii digits following (y) ; y specifies the number of binary data
   following; "ffff" represent the 32Bit floats as log result. */
  /* get the result */
  errStatus =  viPrintf(vi,"SENS%1d:CHAN%1d:FUNC:RES?\n",slot,chan);
  /* only query an error, if there is one, else the query will be interrupted ! */
  if(errStatus < VI_SUCCESS)checkError(vi,errStatus);

  /* read the data binary */
  errStatus = viRead(vi, logBuffer, MAX_LOG_VALUES * sizeof(ViReal32) + HEADER
_SIZE, &retCnt);
  checkError(vi,errStatus);

  if(logBuffer[0] != '#')
    {
    printf("invalid format returned from logging\n");
    exit(1);
    }
  else
    {
    noOfDigits = logBuffer[1] -'0';
    memcpy( logResults, &logBuffer[2 + noOfDigits ], MAX_LOG_VALUES * sizeof(V
iReal32));
    }

  /* stop logging */
  viPrintf(vi,"SENS%1d:CHAN%1d:FUNC:STAT LOGG,STOP\n",slot,chan);
  checkError(vi,errStatus);

  /* display the values */
  for ( i = 0; i < noOfValues; i++)
  printf("\t%g\n",logResults[i]);

  /* loop, until a key is pressed */
  while(!scanf("%c",&c));

  /* close the session */
  viClose(vi);

 }

void checkError(ViStatus session, ViStatus err_status )
 {
  ViStatus error;
  ViChar errMsg[256];
    error = viQueryf(session,"SYST:ERR?\n","%t",errMsg);
    if (error == VI_ERROR_TMO)
      {
      printf("System Error!\n") ;
      exit(1);
```

```
        }
      else
        {
        /* only errors should be displayed */
        if(errMsg[0] != '+')
        printf("error:%ld --> %s\n", err_status,errMsg) ;
        }

  }

ViStatus InitVisa ( ViSession *iHandle)
  {
   ViStatus    errStatus;    /* returned error code from visa call */
       ViSession    defaultRM;    /* default visa resource manager variable */
   /* First get initialized the visa library (see example 1) */
   errStatus = viOpenDefaultRM (&defaultRM);
   if (errStatus < VI_SUCCESS)
       printf("Failed to open VISA Resource manager\n");

   /* Open session to HPIB device at address 20; */
   errStatus = viOpen (defaultRM, "GPIB::20::INSTR", VI_NULL,VI_NULL,iHandle);
   if (errStatus < VI_SUCCESS)
       printf("Failed to open instrument\n");

   return errStatus;
  }
```

The HP 816x VXIplug&play Instrument Driver

This chapter gives you extra information about installing and getting started with the HP 816x VXI*plug&play* instrument driver.

There are details about opening and closing an instrument session, data types and constants used, error handling, and the programming environments supported.

Installing the HP 816x Instrument Driver

The HP 816x VXI*plug&play* Instrument Driver comes as a self-extracting archive with an installation wizard. The installation wizard extracts all the files to preset destinations, asking you appropriate questions as it does so.

You install the driver by running the executable `hp816x.exe`.

1 Run `hp816x.exe`, you see a Welcome screen telling you that the HP 816x VXI*plug&play* Instrument Driver will be installed and the instruments that support the driver.

2 Press `Next>` to continue.

If you are not an administrator, you see a VXI*plug&play* window, and a message telling you that some if you proceed with the installation, some information will NOT be visible to all users. This means that any program menu options will only be available to the user that performed the installation. If you are the administrator all program menu options will be visible for all users.

If you see the message in Figure 8, press `Yes` to install the driver or press `No` and contact your administrator.

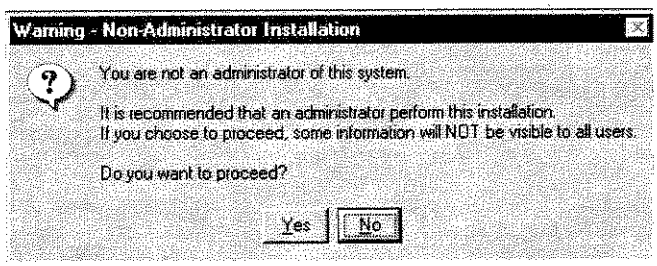


Figure 8 Non-Administrator Installation Pop-Up Box

NOTE If HP 816x VXI*plug&play* Instrument Driver is already installed on your system, you see a message asking you if you want to uninstall the old version.
Press `Yes`, if required, then wait until you see a message telling you that the uninstall has been successful. You may be asked for permission to remove shared files.
Then press `Yes` to continue.

3 You see a message, as shown in Figure 9, advising you to close the programs that you have running.

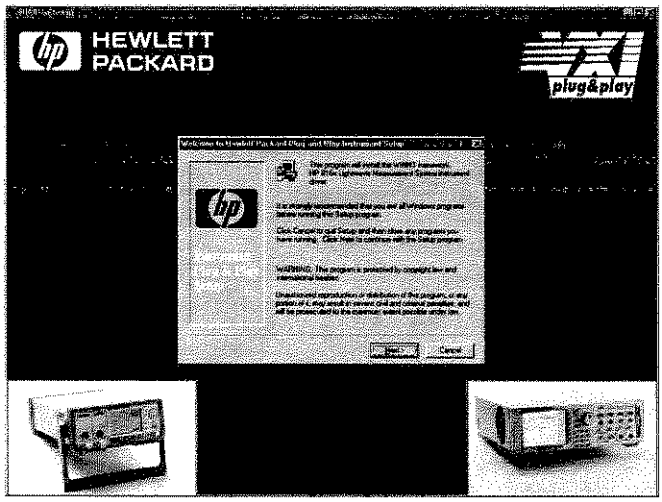


Figure 9 Message Screen

4 Close these programs and press `Next>` to continue. Then, you see a message informing you if VISA is installed on your PC.

NOTE If you do not have VISA installed, press **`Cancel`** to temporarily exit this installation procedure; install VISA on your PC, then run `hp816x.exe` again.

If you have VISA installed, press `Next>` to continue. You see a window that requests you to choose your Setup.

5 You can choose a `Typical`, `Compact`, or `Custom` Setup. Choose a setup option and press `Next>` to continue.

NOTE If you choose the `Custom` Setup, you may choose the options you want to install from the screen in Figure 10.

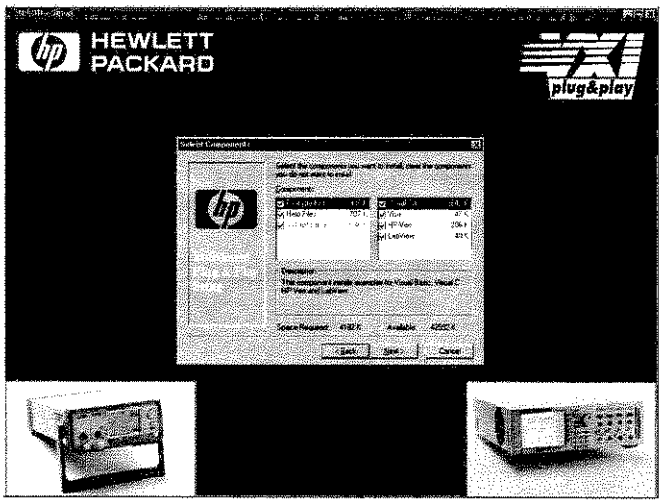


Figure 10 Customizing Your Setup

Select the components you want to install and press `Next>` to continue.

You see a window showing you the Program Folder options you can install, see Figure 11.

6 Select any or all of `Read Me`, `Help` and `Uninstall`, then press `Next>` to continue.

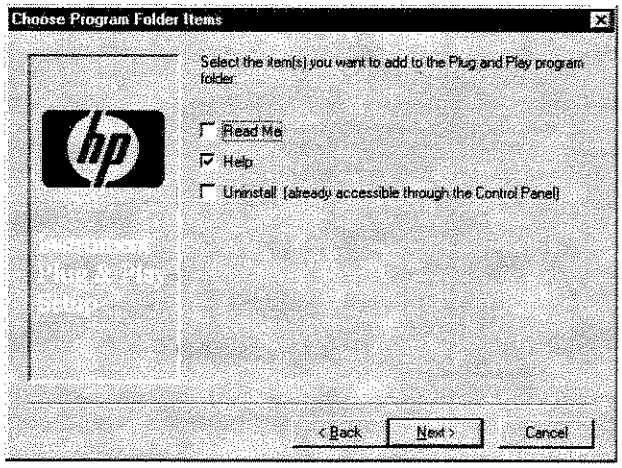


Figure 11 Program Folder Item Options

You see a window asking you in which folder you want to install the files.

7 Select the default, VXIPNP, or choose another folder. Press Next> to continue.

You see a message summarizing the options you have chosen.

If you are satisfied, with the options press Next> to continue.

If you want to review or change any settings press Back>.

8 If you press Next> to continue, the installation is performed and you see a message saying that setup is complete, giving you an option to view the Readme file.

9 Press Finish to complete installation, viewing the Readme file if you wish. A webpage explaining how to get started with the HP 816x VXI*plug&play* Instrument Driver using HP VEE or LabView appears.

Using Visual Programming Environments

Getting Started with HP VEE

Hewlett-Packard Visual Engineering Environment (HP VEE) is a visual programming language optimized for instrument control applications. To develop programs in HP VEE, you connect graphical 'objects' instead of writing lines of code. These programs resemble easy-to-understand block diagrams with lines.

HP VEE allows you to leverage your investment in textual languages by integrating with languages such as C, C++, Visual Basic, FORTRAN, Pascal, and HP BASIC.

HP VEE controls GPIB, VXI, Serial, PC Plug-in, and LAN instruments directly over the interfaces or by using instrument drivers.

HP VEE supports VXI*plug&play* drivers in the WIN, WIN95, WINNT, and HP-UX frameworks. In addition, versions 3.2 and above of HP VEE support the graphical Function Panel interface, providing a function tree of the hierarchy of the driver.

NOTE This appendix assumes that you are using Windows 95. If you are using Windows NT, please replace every reference to win95 with winnt. Windows 95 and Windows NT are registered trademarks of Microsoft corporation.

HP VEE automatically calls the *initialize* and *close* functions to perform automatic error checking.

GPIB Interfacing in HP VEE

HP VEE supports interfacing with an instrument from a GPIB card. Before you can do this, you must do the following:

1 Select INSTRUMENT MANAGER from the IO menu.

2 Double-click on the Add button to bring up the Device Configuration screen, see Figure 12.

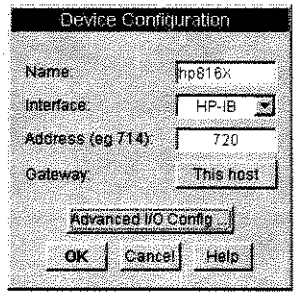


Figure 12 Device Configuration

3 Enter the following information:

- **Name**: enter `hp816X`.
- **Interface**: `HP-IB`
- **Address**: Enter the GPIB address of your GPIB interface board (the default is 7). Append the GPIB address of your instrument (the default is 20).

NOTE To find out or change the instrument's GPIB address, press the CONFIG hardkey on the instrument's front panel and choose GPIB address. The instrument's GPIB address appears, you may edit it if you wish.

- **Gateway**: `This host`.

4 Press `Advanced I/O Config ...`, the Advanced Device Configuration box pops up. Select the Plug&play Driver tab, the box in Figure 13 appears.

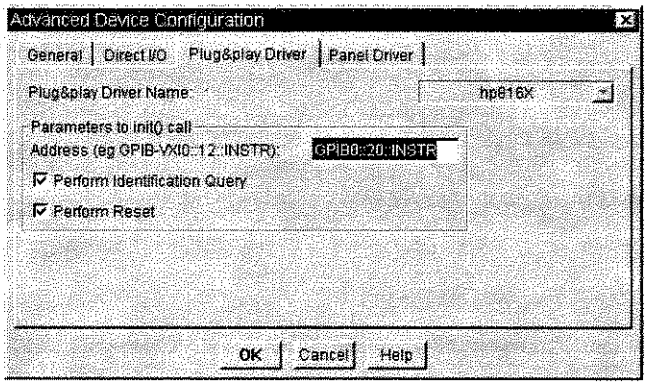


Figure 13 Advanced Device Configuration - Plug&play Driver

5 Select hp816X from the Plug&play Driver Name drop-down list.

NOTE If you do not see this driver in the list, the driver has not installed properly.

6 Enter the Parameters to the init() call by entering GPIB::xx::INSTR where xx is your instrument's GPIB address.

NOTE 20 is the default GPIB address for your instrument.

7 Select whether to Perform Reset or to Perform Identification Query whenever HP VEE opens the instrument for interaction.

8 Confirm the selections pressing the OK button.

9 Return to the Instrument Manager screen and select OK to save the configuration.

Getting Started with LabView

The 32-bit HP 816x driver can be used with LabView 5.0 and above. LabView 5.0 is a 32-bit version of LabView which runs on Windows 95 and Windows NT.

To access the functions of the HP 816x instrument driver convert the driver using the following steps:

1 Run LabView.

2 Select Update VXIplug&play Drivers... from the File menu.

3 The Update VXIPNP Drivers box appears, hp816x* should be listed as a CVI Instrument Driver to be converted.

4 Press `FP Options`, the FP Conversion Options box appears. Check that the options are the same as displayed in Figure 14.

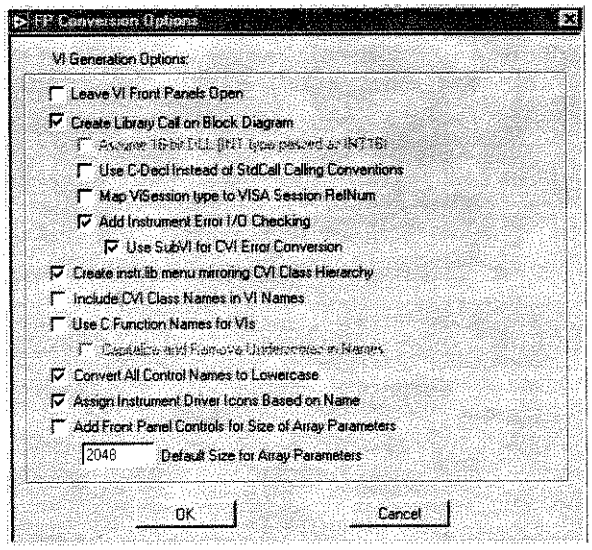


Figure 14 FP Conversion Options Box

5 Press `OK` and confirm the conversion pressing `OK` in the parent window. LabView will create the instrument VI library.

6 If the hp816x driver does not appear in the list for being updated, select `Convert CVI FP file` from the `File` menu.

7 You are asked for a function panel file. Locate the `hp816x.fp` file, which is normally installed into the path `<drive>:VXIPNP\winXX\hp816x`, where XX stands for NT, 95, or 98. Select `hp816x.fp` and the driver file `hp818x_32.dll`, which is installed in the folder `<drive>:VXIPNP\winXX\bin`. Start the conversion with the same FP options as shown above. After the conversion is done the file `hp816x.llb` is created in the same folder, where the `hp816x.fp` file is located. Move `hp816x.llb` to the folder `instr.lib` of LabView.

LabView will create a series of VI's, one per driver function. It will create a file called `hp816x.llb` which contains these VI's. This library of VI's can then be accessed like any other VI library in LabView.

NOTE You must use the 32-bit version of the HP 816x driver with LabView 5.0.

NOTE LabView is a trademark of National Instruments Corporation.

Getting Started with LabWindows

The 32-bit HP 816x driver can be used with LabWindows 4.0 and above. LabWindows 4.0 is a 32-bit version of LabWindows which runs on Windows 95 and Windows NT.

To access the functions of the HP 816x driver from within LabWindows, select INSTRUMENT from the main menu, and then select the LOAD... submenu item.

In the file selection dialog box which appears, select `hp816x.fp` and click on the OK button. LabWindows loads the function panel and instrument driver.

The driver now appears as a selection on the Instrument menu, and can be treated like any LabWindows driver.

NOTE LabWindows is a trademark of National Instruments Corporation.

Features of the HP 816x Instrument Driver

The HP 816x VXI*plug&play* instrument driver conforms to all aspects of the VXI*plug&play* driver standard which apply to conventional rack and stack instruments.

The following features are available:

- The HP 816x VXI*plug&play* Instrument Driver conforms with the VXI*plug&play* standard.

  There is one exception as the HP 816x driver does not have a soft front panel or a knowledge-based file.

- The HP 816x VXI*plug&play* Instrument Driver is built on top of VISA, and uses the services provided.

  VISA supports GP-IB and VXI protocols. The driver can be used with any GP-IB card for which the manufacturer has provided a VISA DLL.

- The HP 816x VXI*plug&play* Instrument Driver includes a Function Panel (.fp) file.

  The .fp file allows the driver to be used with visual programming environments such as HP-VEE, LabWindows, and LabView.

- The HP 816x VXI*plug&play* Instrument Driver includes a comprehensive on-line help file which complements the instrument manual.

  The help file contains application programming examples, a cross-reference between instrument commands and driver functions, and detailed documentation of each function with examples.

- The HP 816x VXI*plug&play* Instrument Driver includes a Visual Basic (.BAS) file which contains the function calls in Visual Basic syntax, and allows the driver functions to be called from Visual Basic.

  You should only use Visual Basic with this driver if you are familiar with C/C++ function declarations. You must take particular care when working with C/C++ pointers.

Directory Structure

The setup program which installs the HP 816x instrument driver creates the `VXIPNP` directory if it does not already exist. The structures for the Windows NT and Windows 95 vxipnp subdirectory tree are shown in Figure 15.

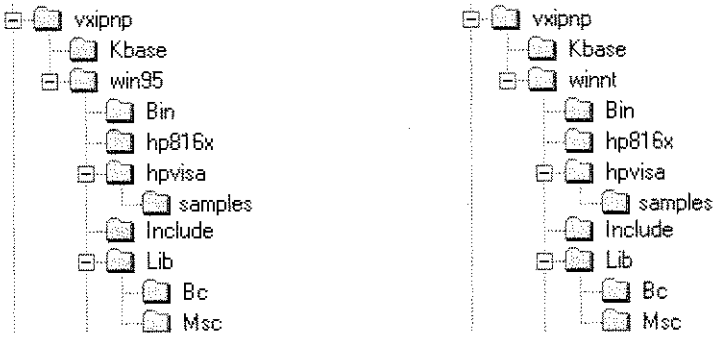


Figure 15 Windows 95 and Windows NT VXIPNP Directory Structure

In the directory example, hp816x is a directory containing the instrument driver. There would be a directory for each instrument driver.

Opening an Instrument Session

To control an instrument from a program, you must open a communication path between the computer/controller and the instrument. This path is known as an instrument session, and is opened with the function

```
ViStatus hp816x_init( ViRsrc InstrDesc, ViBoolean
id_query, ViBoolean reset, ViPSession
instrumentHandle );
```

Instruments are assigned a handle when the instrument session is opened. The handle, which is a pointer to the instrument, is the first parameter passed in all subsequent calls to driver functions.

The parameters of the function `hp816x_init` include:

- **`ViRsrc InstrDesc`**: the address of the instrument
- **`ViBoolean id_query`**: a Boolean flag which indicates if in-system verification should be performed.
  Passing `VI_TRUE` (1) will perform an in-system verification; passing `VI_FALSE` (0) will not.
  If you set `id_query` to false, you can use the generic functions of the instrument driver with other instruments.
- **`ViBoolean reset`**: a Boolean flag which indicates if the instrument should be reset when it is opened.
  Passing `VI_TRUE` (1) will perform a reset when the session is opened; passing `VI_FALSE` (0) will not perform a reset,
- **`ViPSession instrumentHandle`**: a pointer to an instrument session. `InstrumentHandle` is the handle which addresses the instrument, and is the first parameter passed in all driver functions.

Successful completion of this function returns `VI_SUCCESS`

Closing an Instrument Session

Sessions (instrumentHandle) opened with the `hp816x_init()` function are closed with the function:

```
hp816x_close( ViSession instrumentHandle);
```

When no further communication with an instrument is required, the session must be explicitly closed (`hp816x_close()` function).

VISA does not remove sessions unless they are explicitly closed. Closing the instrument session frees all data structures and system resources allocated to that session.

VISA Data Types and Selected Constant Definitions

The driver functions use VISA data types. VISA data types are identified by the `Vi` prefix in the data type name (for example, `ViInt16`, `ViUInt16`, `ViChar`).

The file `visatype.h` contains a complete listing of the VISA data types, function call casts and some of the common constants.

NOTE You can find a partial list of the type definitions and constant definitions for the `visatype.h` in the HP 816x VXI*plug&play* Instrument Driver Online Help.

Error Handling

Events and errors within a instrument control program can be detected by polling (querying) the instrument. Polling is used in application development environments (ADEs) that do not support asynchronous activities where callbacks can be used.

Programs can set up and use polling as shown below.

1 Declare a variable to contain the function completion code.

```
ViStatus errStatus;
```

Every driver function returns the completion code `ViStatus`.

If the function executes with no I/O errors, driver errors, or instrument errors, `ViStatus` is 0 (`VI_SUCCESS`).

If an error occurs, `ViStatus` is a negative error code.

Warnings are positive error codes, and indicate the operation succeeded but special conditions exist.

2 Enable automatic instrument error checking following each function call.

```
hp816x_errorQueryDetect
  (instrumentHandle, VI_TRUE);
```

When enabled, the driver queries the instrument for an error condition before returning from the function.

If an error occurred, `errStatus` (Step 1) will contain a code indicating that an error was detected (`hp816x_INSTR_ERROR_DETECTED`).

3 Check for an error (or event) after each function.

```
errStatus = hp816x_cmd(instrumentHandle,
 "SENS1:POW:RANG");
check(instrumentHandle, errStatus);
```

After the function executes, `errStatus` contains the completion code.

The completion code and instrument ID are passed to an error checking routine. In the above statement, the routine is called 'check'.

4 Create a routine to respond to the error or event. This example queries whether an error has occured, checks if the error is an instrument error and then checks if the error is a driver error.

```
void check (ViSession instrumentHandle, ViStatus errStatus)
{
 /* variables for error code and message */
 ViInt32 inst_err;
 ViChar err_message[256];

 /* VI_SUCCESS is 0 and is defined in VISATYPE.h */
 if(VI_SUCCESS > errStatus)
 {
 /* hp816x_INSTR_ERROR_DETECTED defined in hp816x.h */
  if(hp816x_INSTR_ERROR_DETECTED == errStatus)

  {
  /* query the instrument for the error */
  hp816x_error_query(instrumentHandle, &inst_err, err_message);

  /* display the error */
  printf("Instrument Error : %ld, %s\n", inst_err, err_message);
  }
  else /* driver error */
  {
  /* get the driver error message */
  hp816x_error_message(instrumentHandle, errStatus, err_message);

  /* display the error */
  printf("Driver Error : %ld, %s\n", errStatus, err_message);

  }
  /* optionally reset the instrument, close the instrument handle */
  hp816x_reset(instrumentHandle);
  hp816x_close(instrumentHandle);
  exit(1);
 }
 return;
```

Introduction to Programming

Example Programs

See the Online Help and *"Programming Examples" on page 123.*

VISA-Specific Information

The following information is useful if you are using the driver with a version of VISA.

Instrument Addresses

When you are using HP VXI*plug&play* instrument drivers, you should enter the instrument addresses using only upper case letters. This is to ensure maximum portability.

For example, use `GPIB0::22` rather than `gpib0::22`.

Callbacks

Callbacks are not supported by this driver.

Development Environments

These sections contains suggestions as to how you can use `hp816x_32.dll` within various application development environments.

Microsoft Visual C++ 4.0 (or higher) and Borland C++ 4.5 (or higher)

Please refer to your Microsoft Visual C++ or Borland C++ manuals for information on linking and calling DLLs.

Microsoft Visual Basic 4.0 (or higher)

Please refer to your Microsoft Visual Basic manual for information on calling DLLs.

The BASIC include file is `hp816x.bas`. You can find this file in the directory `~vxipnp\win95\include`, where ~ is the directory in the VXIPNP variable.

By default, ~ is equivalent to `C:\`. This means that the file is in `C:\vxipnp\win95\include`.

You may also need to include the file `visa.bas`. `visa.bas` is provided with your VISA DLL.

HP VEE 5.01 (or higher)

Your copy of HP VEE for Windows contains a document titled *Using VXIplug&play drivers with HP VEE for Windows*. This document contains the detailed information you need for HP VEE applications.

LabWindows CVI/ (R) 4.0 (or higher)

The HP 816x VXI*plug&play* Instrument Driver is supplied as a Dynamic Link Library (.DLL) file.

There are several advantages to using the .DLL form of the driver, including those listed below:

- transportability across different computer platforms,
- a higher level of support for the compiled driver from Hewlett-Packard,
- a faster load time for your project.

LabWindows/CVI (R) will attempt by default to load the source version of the instrument driver. To load the DLL, you must include the file `hp816x.fp` in your project. `hp816x.fp` can be found in the directory `vxipnp\win95\hp816x`.

Do not include `hp816x.C` in your project.

You must provide an include file for `hp816x.H`. You do this by ensuring that the directory `~vxipnp\win95\include` is added to the include paths (CVI Project Option menu).
~ is the directory in the VXIPNP variable. By default, ~ is equivalent to `C:\`. This means that the file is in `C:\vxipnp\win95\include`.

Online Information

The latest copy of this driver and other HP VXI*plug&play* drivers can be obtained via anonymous ftp from `fcext3.external.hp.com` from the directory `~dist/mxd/vxipnp/pnpdriver.lis`.

It may also be obtained on the World Wide Web from

`ftp://fcext3.external.hp.com/dist/mxd/vxipnp/pnpdriver.lis`.

The HP 816x driver is located in a self-extracting archive file called `hp816x.EXE`.

If you do not have ftp or web access, please contact your HP supplier, or use the version of `hp816x.exe` on your installation CD.

GPIB Command Compatibility List

This chapter gives information about adapting programs developed for use with HP 8153A Lightwave Multimeter or HP 8167B/8D/8E/8F Tunable Laser Source.

Compatibility Issues

For each table entry in this chapter, it is noted whether the compatibility change affects either:

- the HP 8153A Lightwave Multimeter - 8153,
- the HP 8167B/8D/8E/8F Tunable Laser Source - 8167/8, or
- both of these instruments - Both.

GPIB Bus Compatibility

These commands are incompatible.

Command	Change	Affects
LLO - local lockout		Both
DCL - device clear		Both
GET - group execute trigger		Both

Table 9 Incompatible GPIB Bus Commands

Status Model

The status model is completely incompatible with the HP 8153A and HP 8167/8.

Preset Defaults

The preset defaults are different.

Removed Command

Table 10 contains details of commands that have been removed without replacement.

Command	Change	Affects
*SRE/?	No support for this command/query.	Both
*TRG	No support for triggered commands.	8153
ABORt	This command is not supported; in every case, the bus is blocked during command execution.	8153
STATus:OPERation: NTRansition/?	These status model features are not supported.	8153
STATus:OPERation: PTRansition/?		
STATus:QUEStionable: NTRansition/?		
STATus:QUEStionable: PTRansition/?		
SYSTem:BEEPer:STATe/?	Beeper access is not supplied.	8153
*SAV *RCL	User interface or GPIB settings cannot be stored or re-called.	8167/8
BDATA? DOSMODE/?	Memory card access is not provided.	8167/8
TRACe:CATalog? TRACE:DATA? TRACE:POINts?	The TRACe tree is not supported; the CC_UNCAL curve does not exist.	8167/8
WAVEACT	Alignment to external wavemeter is not supported.	8167/8
misc 200	Risetime control is not supported yet.	8167/8

Table 10 Removed Commands

Obsolete Commands

Table 10 contains details of commands that have been directly replaced.

Old Command	New Command	Affects
DISPlay:STATe/?	`DISPlay:ENABle/?`	8153
PROGram command tree	`SENSe:FUNCtion` command tree. Some commands from the `PROGram` command tree have not been replaced. The HP 8153A application interface on the GPIB is not supported. Stability/Logging and Min/Max are available via a new interface.	8153
Return Loss Module Commands	The commands for the return loss modules will be completely different than those for the HP 8153A.	8153

Table 11 Obsolete Commands

Changed Parameter Syntax and Semantics

Table 12 details commands whose parameter syntax or semantics have changed.

Command	Change	Affects
SOUR:AM:FREQ/?	This command does not accept the value CW, instead use SOUR:AM:STAT ON|OFF to switch from and to CW mode. The commands accepts floating point values.	8153
DISP:BRIG	This command now supports integers between 1 and 100, instead of float values between 1 and 0.	8153
SENS:CORR:COLL:ZERO?	This command returns the last zero state, instead of the last remote zero state.	8153
SENS:POW:REF	Accepts `TOMODule` and `TOREF` for the first parameter, instead of accepting `TOA | TOB` as the HP 8153A does. The numbers `0 | 1 | 2` cannot be used, only the strings above.	8153
SENS:POW:REF:STAT:RAT	Accepts `TOREF`, `0` or values for slot,channel, instead of accepting `TOA | TOB` as the HP 8153A does. The numbers have a different meaning.	8153
SYST:DATe	`SYST:DATe` from HP 8167/8 is not supported, but `SYST:DATe` from HP 8153 is supported.	8167/8

Table 12 Commands with Different Parameters or Syntax

Changed Query Result Values

Table 13 details queries that respond with different return codes than the old instruments.

Command	Change	Affects
*IDN?	Returns new instrument and module identifiers.	Both
*OPT?	Returns new module options.	Both
*TST	Selftest result codes are completely new. 0 still means passed.	Both
	A head adapter is not overwritten with the head when it is inserted.	8153
SENS:POW:UNIT?	Returns W | DBM not a number.	8153
SOUR:POW:WAV?	Returns LOW | UPP | BOTH | EXT and not the wavelength; use SOUR:WAV? to query the wavelength. SOUR:WAV:FIXED1? returns the wavelength of the first laser and SOUR:WAV:FIXED2? returns the wavelength of the second laser. For the HP 8153A, SOUR:POW:WAV? returned the wavelength of the active laser.	8153
SYSTem:ERRor?	Same functionality but different numbers and errors are returned for instrument specific errors.	8153
SOURce:AM:SOURce?	Returns different enum values than the HP 8167/8.	8167/8

Table 13 Queries with Different Result Values

Timing Behavior

Table 14 details the ways in which timing behavior is different.

Change	Affects
Command execution may be different.	Both
GPIB will block during command execution, except when executing functions, such as logging and sweep, that don't tolerate blocking. This is identical to the behavior of the 8167/8. A side effect of this is that *OPC? always returns 1.	8153
When continuous triggering and averaging times are greater than 1 second, the read-out values reset after the averaging time is over; there is no sliding behavior.	8153

Table 14 Timing Behavior Changes

Error Handling

Most error commands and error texts for all instruments are new.

The HP 8153A timed out for every error. Errors are handled differently by the HP 8163A/4A; instead of timing out for every error, special values are returned for erroneous queries. Table 15 and Table 16 detail the new errors

The error queue is written to as before.

Expected Return Value	Returned Value	Affects
FLOAT(32/64)	FLT/DBL_MAX	8153
(U)INT(16/32)	(U)INT(16/32)_MA	8153
Block	" "	8153
Boolean Value	0	8153
Enum	Time out	8153

Table 15 Error Handling Changes

Command	Change	Affects
FETCh:POWer? - without using a preceding trigger	Returns the last valid value instead of timing out. No error is generated.	8153

Table 16 Specific Errors

Command Order

It is not yet known if there are any changes in the command order behavior.

Instrument Status Settings

The trigger configuration automatically overrides other instrument setting and control capabilities. This applies to both the HP 8153A and HP 8167/8.

Error Codes

This chapter gives information about error codes used with the HP 8163A Lightwave Multimeter, the HP 8164A Lightwave Measurement System, and the HP 8166A Lightwave Multichannel System.

GPIB Error Strings

New/Old/Standard	Error Number	String
-100 to -199 Command Errors		
Standard	-100	"Command Error"
Standard	-101	"Invalid character"
Standard	-102	"Syntax error"
Standard	-103	"Invalid separator"
Standard	-104	"Data type error"
Standard	-105	"GET not allowed"
Standard	-108	"Parameter not allowed"
Standard	-109	"Missing parameter"
Standard	-112	"Program mnemonic too long"
Standard	-113	"Undefined header"
Standard	-120	"Numeric data error"
Standard	-121	"Invalid character in number"
Standard	-123	"Exponent too large"
Standard	-124	"Too many digits"
Standard	-128	"Numeric data not allowed"
Standard	-131	"Invalid suffix"
Standard	-134	"Suffix too long"
Standard	-138	"Suffix not allowed"
Standard	-141	"Invalid character data"
Standard	-148	"Character data not allowed"
Standard	-150	"String data error"
Standard	-151	"Invalid string data"
Standard	-158	"String data not allowed"
Standard	-161	"Invalid block data"
Standard	-168	"Block data not allowed"
Standard	-170	"Expression error"
Standard	-171	"Invalid expression"

Table 17 Overview for Supported Strings

New/Old/Standard	Error Number	String
Standard	-178	"Expression data not allowed"
Standard	-181	"Invalid outside macro definition"
Standard	-183	"Invalid inside macro definition"
New	-185	"Subop out of range"
-200 to -299 Execution Errors		
Standard	-200	"Execution error"
New	-205	"Powermeter not running"
Old	-211	"Trigger ignored"
Old	-212	"Arm ignored"
Old	-213	"Init ignored"
Old	-220	"Parameter error"
Old	-221	"Settings conflict"

NOTE If error -221 is returned after you try to start a wavelength sweep, one of the following cases of sweep parameter inconsistency has occurred:

- Continuous Sweep mode AND λ Start is less than λ Stop.
- Continuous Sweep mode AND Sweep Time is too short. Adjust Sweep Speed, λ Start, or λ Stop.
- Continuous Sweep mode AND Sweep Time is too long. Adjust Sweep Speed, λ Start, or λ Stop.
- Continuous Sweep mode AND Trigger Frequency is too high. Adjust Step Size. Trigger Frequency is the Sweep Speed divided by the Step Size.
- Stepped Sweep mode AND Lambda Logging Enabled.
- Continuous Sweep mode AND Lambda Logging Enabled AND Output trigger mode not set to STFinished (Step finished).
- Continuous Sweep mode AND Lambda Logging is Enabled AND Modulation Source is not set to OFF.
- Continuous Sweep mode AND Lambda Logging is Enabled AND Sweep Cycles is not set to 1.
- Continuous Sweep mode AND Coherence Control is Enabled.

Standard	-222	"Data out of range"
Standard	-223	"Too much data"
Standard	-224	"Illegal parameter value"

Table 17 Overview for Supported Strings

New/Old/Standard	Error Number	String
Old	-231	"Data questionable"
Old	-261	"Math error in expression"
Standard	-272	"Macro execution error"
Standard	-273	"Illegal macro label"
Standard	-276	"Macro recursion error"
Standard	-277	"Macro redefinition not allowed"
Standard	-278	"Macro header not found"
Old	-284	"Function currently running"
Old	-286	"No function currently running"
New	-290	"Application currently running - no HP-IB support"

NOTE When an application is running error -290 will be returned if any command apart from the following are sent:

- *WAI
- *OPC?
- :SPECial:REBoot
- :SYSTem:ERRor?
- :SYSTem:VERSion?

-300 to -399 or between 1 and 32767 Device-Specific Errors (Module)

Old	-300	"Internal error"
New	-301	"Module doesn't support this command"
New	-302	"Internal timeout error"
New	-303	"Module slot empty or slot / channel invalid"
New	-304	"Command was aborted"
New	-305	"Internal messaging error"
New	-306	"Channel doesn't support this command"
New	-307	"Channel without head connection"
Standard	-310	"System error"
Standard	-321	"Out of memory"
New	-322	"Flash programming error"
Old	-330	"Self-test failed"
New	-340	"Printing error"
New	-341	"Printing error - paper out"

Table 17 Overview for Supported Strings

New/Old/Standard	Error Number	String
New	-342	"Printing error - offline"
Standard	-350	"Queue overflow"
-400 to -499 Query Errors		
Standard	-400	"Query error"
Standard	-410	"Query INTERRUPTED"
Standard	-420	"Query UNTERMINATED"
Standard	-430	"Query DEADLOCKED"
Standard	-440	"Query UNTERMINATED after indef resp"

Table 17 Overview for Supported Strings

	Error	
New/Old/Standard	**Number**	**String**
Old	all positive errors	
Old	-110	"Command header error"
Old	-111	"Header seperator error"
Old	-114	"Header suffix out of range"
Old	-130	"Suffix error"
Old	-140	"Character data error"
Old	-144	"Character data too long"
Old	-160	"Block data error"
Old	-201	"Invalid while in local"
Old	-202	"Settings lost due to ???"
Old	-210	"Trigger error"
Old	-214	"Trigger deadlock"
Old	-215	"Arm deadlock"
Old	-230	"Data corrupt or stale"
Old	-240	"Hardware error"
Old	-241	"Hardware missing"
Old	-260	"Expression error"
Old	-280	"Program error"
Old	-281	"Cannot create program"
Old	-282	"Illegal program name"
Old	-283	"Illegal variable name"
Old	-285	"Program syntax error"
Old	-286	"Program runtime error"
Old	-311	"Memory error" [checksum or parity]
Old	-312	"Protect user data memory lost"
Old	-313	"Calibration memory lost"
Old	-314	"Save/Recall Memory lost"
Old	-315	"Configuration memory lost"

Table 18 Overview for Unsupported Strings

Index

T

U

W