# 8163A/B Lightwave Multimeter, 8164A/B Lightwave Measurement System, & 8166A/B Lightwave Multichannel System

Programming Guide

KEYSIGHT TECHNOLOGIES

# Notices

© Keysight Technologies 2016

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Keysight Technologies as governed by United States and international copyright laws.

## Manual Part Number

08164-90B65

## Edition

Edition 2.0, June 2016

Keysight Technologies Deutschland GmbH
Herrenberger Strasse 130,
71034 Böblingen, Germany

## Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

## U.S. Government Rights

The Software is “commercial computer software,” as defined by Federal Acquisition Regulation (“FAR”) 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement

(“DFARS”) 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at http://www.keysight.com/find/sweula. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

## Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS," AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT WILL CONTROL.

## Safety Notices

**CAUTION**

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

**WARNING**

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

# Contents

## 3 Instrument Setup and Status



## 4 Measurement Operations & Settings



## 5 Mass Storage, Display, and Print Functions

## 6 VISA Programming Examples



## 7 The Keysight 816x VXIplug&play Instrument Driver

# 1 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 8163A/B Lightwave Multimeter, 8164A/B Lightwave Measurement System, and 8166A/B 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*.
- The International Institute of Electrical and Electronics Engineers. *IEEE Standard 488.2-1992, IEEE Standard Codes, Formats, Protocols and Common Commands For Use with ANSI/IEEE Std 488.1-1987*.

To obtain a copy of either of these last two documents, look at:

http://standards.ieee.org/findstds/standard/instrumentation_and_measurement.html

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

For information about SCPI, and SCPI programming techniques, please refer to:

- *Standard Commands for Programmable Instruments*.
  Web: http://www.ivifoundation.org/docs/scpi-99.pdf
  See also: http://www.ivifoundation.org/scpi/

The interface of the 8163A/B Lightwave Multimeter, 8164A/B Lightwave Measurement System, and 8166A/B Lightwave Multichannel System to the GPIB is defined by the IEEE Standards 488.1 and 488.2.

Table 1 on page -9 shows the interface functional subset that the instruments implement.

**Table 1 GPIB Capabilities**

| Mnemonic | Function |
|---|---|
| SH1 | Complete source handshake capability |
| AH1 | Complete acceptor handshake capability |
| T6 | Basic talker; serial poll; no talk only mode; unaddressed to talk if addressed to listen |
| L4 | Basic listener; no listen only mode; unaddressed to listen if addressed to talk |
| SR0 | No service request capability |
| RL1 | Complete remote/local capability |
| PP0 | No parallel poll capability |
| DC1 | Complete device clear capability |
| DT0 | No device trigger capability |
| C0 | No controller capability. |

### 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 71.
- 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.

**NOTE** GPIB address 21 is often applied to the GPIB controller. If so, 21 cannot be used as an instrument address.

## LAN Interface

The current generation of mainframes 8163B (starting with S/N DE48204000) and 8164B (starting with S/N DE48202000) are also equipped with a LAN Interface. The LAN Interface can be operated either with fixed or variable IP address, depending on whether a DHCP server is available and the DHCP mode is enabled.

The GPIB conforms to IEEE standard 488-1978, ANSI standard MC 1.1 and IEC recommendation 625-1.

Available connections:

- LAN (VXI-11 (RPC))
- LAN (TCP Socket, port 5025)
- LAN (Telnet, port 5024)(Quit telnet sessions with Ctrl-D)

2 VXI-11 connections and up to 10 socket or telnet connections (socket + telnet ~ 10) are possible. The ports for socket and telnet connections cannot be modified.

Please note the following points:

- We recommend you use socket connections for applications. (Can be configured in Keysight Connection Expert)
- All LAN VXI-11 connections share their status information (Error queue, status registers).
- Socket and telnet connections are independent of each other.
- The status registers are updated if a new socket or telnet connection is opened.
- Be careful if there is more than one connection:
  - ·When using a VISA layer you can lock a device, so another controller accessing the same device with the same connection type (VISA VXI-11 or VISA socket) cannot access this device. However the device can be accessed with a different connection. This means, for example, if a device is locked in a VISA socket connection it can still be accessed with VXI-11.
  - A telnet connection cannot be locked.

  - Especially when using VXI-11 connections, lock the device. Otherwise one controller may get the reply for another controller's query.
  - If the device is locked in a VISA socket connection, you can still access it using raw sockets (that is, through directly using sockets).
- When using sockets (VISA and raw) there are some extra points to be aware of:
  - Always append a 'newline' to your command or query.
  - Don't use indefinite blocks.
  - It is the programmer's responsibility to make sure all the transmitted data is read.
  - The Agilent VISA Assistant is confused by 'newline' inside a reply. In this case, click the button [viScanf] until the whole reply is read.
- VXI-11 connections are about 50% slower than Socket

### Returning the Instrument to Local Control

If the instrument is in remote control, a screen resembling Figure 1 on page -12 will appear. Press [Local] if you wish to return the instrument to local control.

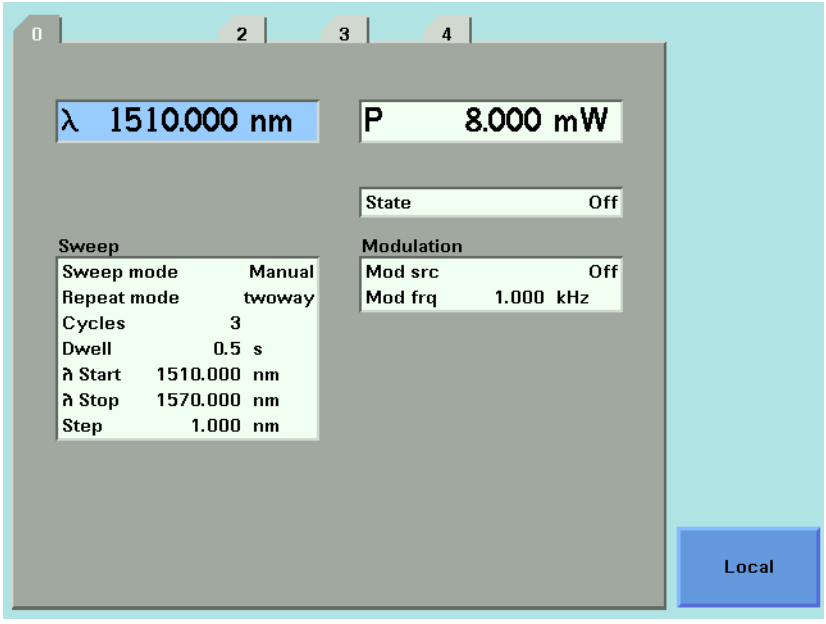



Figure 1 Remote Control

**NOTE** If your 8163A/B, 8164A/B or 8166A/B mainframe is in local lockout mode (refer to :DISPlay:LOCKout on page 226) the Local softkey is not available.

# 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. For example:

1 If no error has occurred, the error queue contains:
+ 0, "No error"

2 After a command such as wav:pow, the error queue now contains:
+ 0, "No error"
-113, "Undefined header"

3 If the command is immediately repeated, the error queue now contains:
+ 0, "No error"
-113, "Undefined header"
-113, "Undefined header"

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.

**Table 2 Units and allowed Mnemonics**

| 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 |

### 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>*. The block represents an array of numbers. Each number has the byte ordering least significant byte first, also called LSBfirst, little-endian or Intel byte ordering.

**NOTE** Some programming environments, like VEE, support reading these binary blocks directly, including selection of byte ordering. Other convenient ways can be using the 816x VXI Plug&Play driver, or using the 816x Lightwave Command Set together with the Keysight Command Expert for generating SCPI code. If the VISA COM interface is used, then its IFormattedIO488 interface is very convenient, setting the property InstrumentBigEndian to "false" and using the READIEEEBlock method.

**NOTE** Note that within your program, calculations with wavelengths may require double-precision 64-bit floats to provide the desired resolution.

### 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 8163A/B,
- from zero to four for the 8164A/B, and
- from one to seventeen for the 8166A/B.

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

**NOTE** **The 8164A/B slot for a back-loadable tunable laser module is numbered zero.**

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

Modules with two channels, for example, the Keysight 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 117, 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 8163A/B and 8166A/B mainframes.
- Slot 0 for the 8164A/B 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 120,
- :SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:FPDelta[l]? on page 120,
- :SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[l] on page 121, and
- :SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:FPDelta[l]? on page 120.

# 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 on page -20 provides a summary of the common commands.

**Table 3 Common Command Summary**

| Command | Parameter | Function | Page |
|---|---|---|---|
| *CLS | | Clear Status Command | page 47 |
| *ESE | | Standard Event Status Enable Command | page 47 |
| *ESE? | | Standard Event Status Enable Query | page 48 |
| *ESR? | | Standard Event Status Register Query | page 48 |
| *IDN? | | Identification Query | page 50 |
| *OPC | | Operation Complete Command | page 50 |
| *OPC? | | Operation Complete Query | page 51 |
| *OPT? | | Options Query | page 51 |
| *RST | | Reset Command | page 52 |
| *STB? | | Read Status Byte Query | page 52 |
| *TST? | | Self Test Query | page 53 |
| *WAI | | Wait Command | page 54 |

**NOTE** These commands are described in more detail in IEEE-Common Commands on page 46.

## 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 on page -21shows 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.

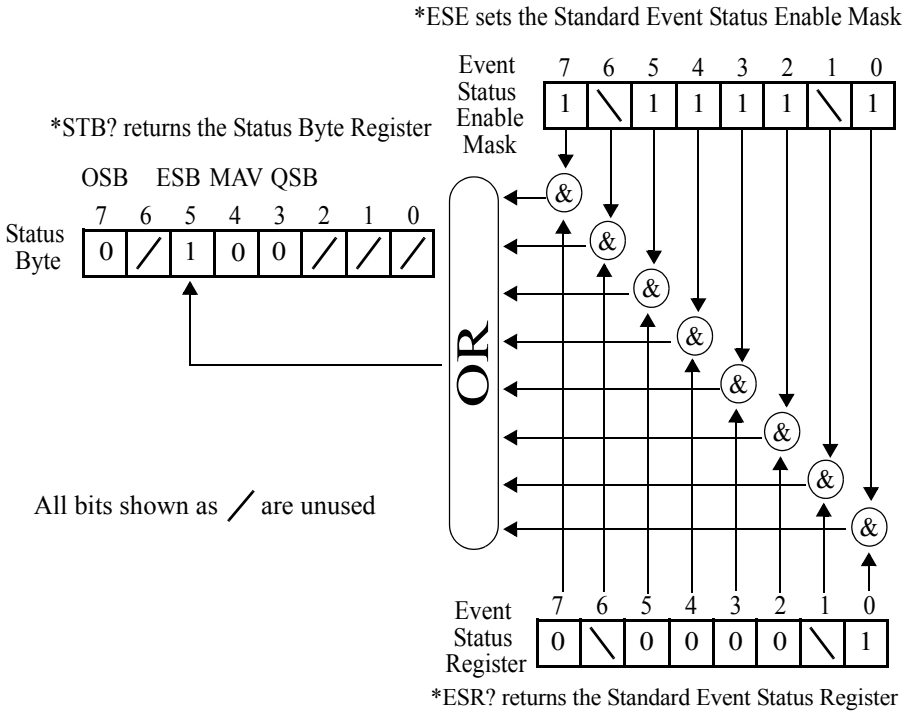



Figure 2 The Event Status Bit

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 28.

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 23 and Status Reporting – The STATus Subsystem on page 55.

**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 on page -26describes how the Questionable Status Bit (QSB) and the Operational Status Bit (OSB) of the Status Byte Register are determined.

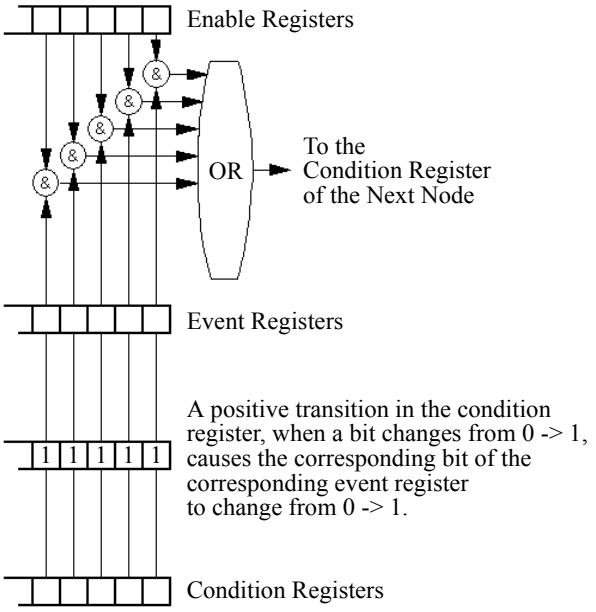



Figure 3 The Registers and Filters for a Node

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 28.

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 1 at 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 1 at 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 8163A/B Lightwave Multimeter and the 8164A/B Lightwave Measurement System consists of one level. These are described in Status System for 8163A/B & 8164A/B on page 26.

As the 8166A/B 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 8166A/B on page 26.

## Status System for 8163A/B & 8164A/B

The status system for the 8163A/B Lightwave Multimeter and the 8164A/B 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.

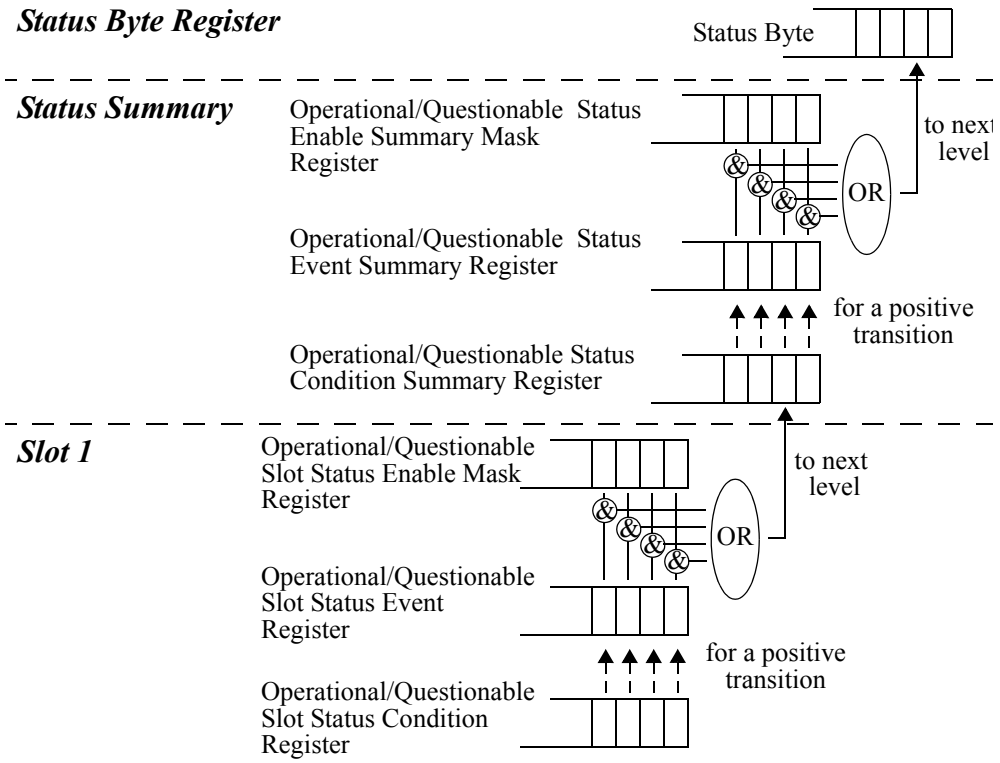



Figure 4 The Operational/Questionable Status System for 8163A/B & 8164A/B

### Status System for 8166A/B

The status system for the 8166A/B 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.

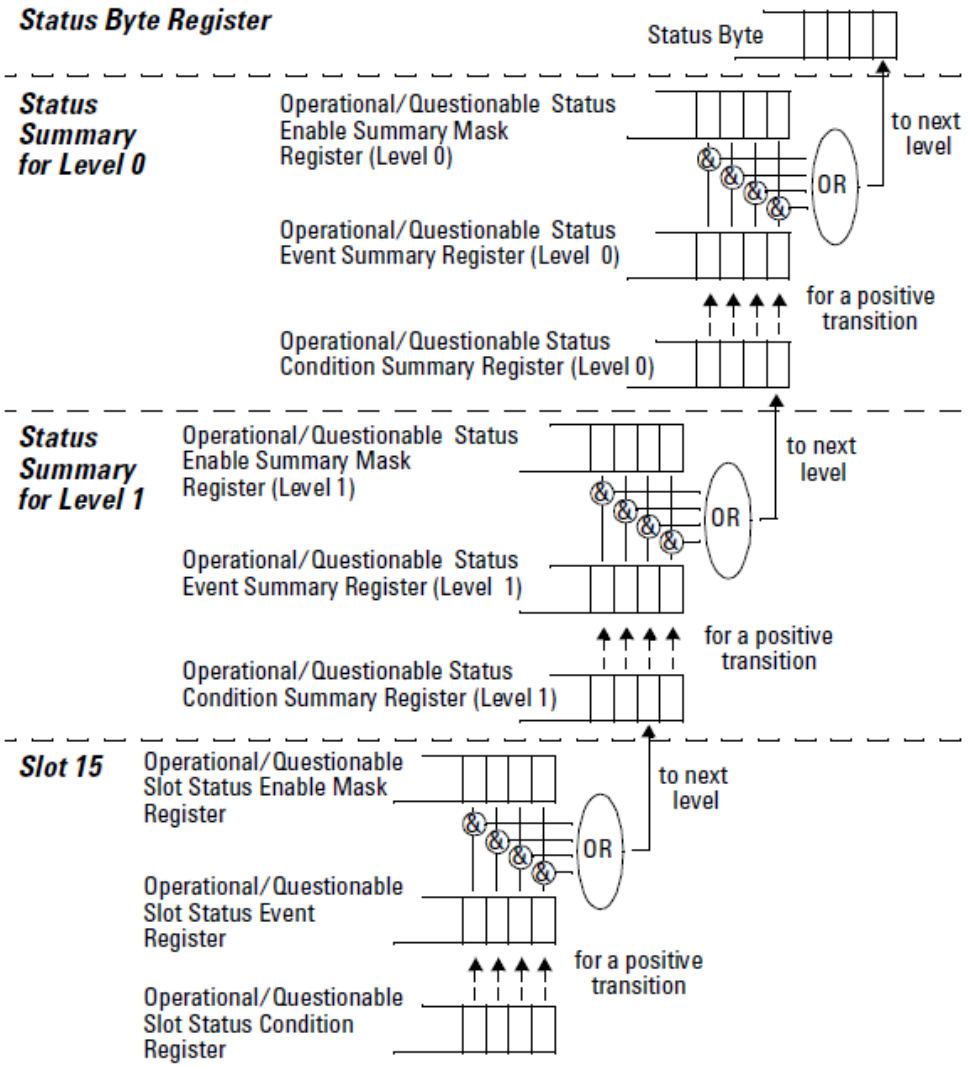



Figure 5 The Operational/Questionable Status System for 8166A/B

Annotations

Status Byte Register

- Bit 3, the QSB, is built from the questionable event status register and its enable mask.
- Bit 4, the MAV, is set if the message output queue is not empty.
- 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.
- Bit 4 is set if the attenuator output is enabled (shutter open).
- Bits 5 - 7 are set if the wavelength offset table is enabled (see page 56).
- 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 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, as during the automatic settling of a Tunable Laser module.
- Bit 5 is set if the module is out of specifications, or if lambda zeroing failed for a Tunable Laser module.
- Bit 6 is set if ARA (auto realign) is recommended.
- Bit 7 is set if the duty cycle is out of range.
- Bit 8 is set if coherence control is uncalibrated
- Bit 9 is set if attenuator beam path protection is enabled (shutter is closed)
- Bit 10 is set if lambda zeroing is recommended.
- Bit 11 is set if the 81490A transmitter needs to be calibrated.
- Bit 12 is set if the 81950A frequency offset does not equal zero.

- 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, and prevents the instrument from executing any further commands until all pending commands are completed. |
| *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.<br>**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:<br>For example, *OPT? -> 81682A, 81533B, 81532A, ,<br>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. |

# 2 Specific Commands



This chapter lists all the instrument specific commands relating to the 8163A/B Lightwave Multimeter, the 8164A/B Lightwave Measurement System, and the 8166A/B Lightwave Multichannel 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 on page 32 gives an overview of the command tree. You see the nodes, the subnodes, and the included commands.

**Table 4 Specific Command Summary**

| Command | Description | Page |
|---|---|---|
| **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet** | | |
| :WAVelength:REFerence/? | Sets or queries the slot and channel of the external reference power meter. | page 198 |
| :WAVelength:STATe/? | Switches or queries attenuator Offset Table on or off? | page 197 |
| :WAVelength:TABle? | Queries the complete offset table. | page 201 |
| :WAVelength:TABle:SIZE? | Queries the size of the offset table. | page 201 |
| :WAVelength:VALue | Adds a value pair (wavelength, offset) to the offset table. | page 198 |
| :WAVelength:VALue:DELete | Deletes an offset value pair. | page 200 |
| :WAVelength:VALue:DELete:ALL | Deletes all value pairs from the offset table. | page 201 |
| :WAVelength:VALue:OFFSet? | Queries an offset value according to wavelength or index. | page 199 |
| :WAVelength:VALue:PAIR? | Queries an offset/wavelength value pair according to wavelength or index. | page 200 |

| Command | Description | Page |
|---|---|---|
| :WAVelength:VALue:WAVelength? | Queries a wavelength value from its index in the offset table | page 199 |
| :DISPlay | | |
| :BRIGhtness/? | Controls or queries the current display brightness. | page 225 |
| :CONTrast/? | Controls or queries the current display contrast. | page 224 |
| :ENABle/? | Switches the display on or off, or queries whether the display is on or off. | page 226 |
| :LOCKout/? | Sets or queries local lockout mode. | page 226 |
| **:FETCh[*n*][:CHANnel[*m*]][:SCALar]** | | |
| :POWer[:DC]? | Returns a power value from a sensor. | page 88 |
| :RETurnloss? | Returns the current return loss value. | page 88 |
| **:INITiate[*n*]:[CHANnel[*m*]]** | | |
| [:IMMediate] | Starts a measurement. | page 89 |
| :CONTinuous/? | Starts or Queries a single/continuous measurement. | page 89 |
| :LOCK/? | Switches the lock on/off or returns the current state of the lock. | page 80 |
| **:INPUT[*n*][:CHANnel[*m*]]** | | |
| :ATTenuation/? | Sets or returns the attenuation factor for the instrument. | page 182 |
| :OFFset/? | Sets or returns the offset factor for the instrument. | page 183 |
| :OFFset:DISPlay | Sets the offset factor so that attenuation factor is zero. | page 183 |
| :OFFset:POWermeter | Sets the offset factor to the difference between the power measured with a powermeter and with the monitor diode. | page 184 |
| :ATTenuation:SPEed/? | Sets or queries the filter transition speed | page 184 |
| :WAVelength/? | Sets or queries the modules attenuation wavelength | page 185 |
| **:OUTPut[*n*][:CHANnel[*m*]]** | | |
| :APMode/? | Sets or queries whether power setting or attenuation value has been changed. | page 185 |
| :APOWeron/? | Sets or queries the shutter status at power on. | page 193 |
| :ATIMe/? | Sets or queries the powermeter averaging time. | page 194 |
| :CONNection/? | Selects or returns Analog Output parameter. | page 122 |
| :CORRection:COLLection:ZERO | Zeros the offsets of attenuators powermeter | page 195 |

| Command | Description | Page |
|---|---|---|
| :CORRection:COLLection:ZERO:ALL | Zeros all available powermeter channels in mainframe | page 195 |
| :CORRection:COLLection:ZERO? | Queries the status of the last zero operation | page 195 |
| :PATH/? | Sets or returns the regulated path. | page 122 |
| :POWer/? | Sets or queries the output power value. | page 187 |
| :POWer:CONTRol/? | Sets or queries power control mode status | page 191 |
| :POWer:OFFSet/? | Sets or queries the power offset value. | page 190 |
| :POWer:OFFSet:POWermeter | Calculates power offset from measured power values | page 190 |
| :POWer:REFerence/? | Sets or queries the reference power value. | page 189 |
| :POWer:REFerence:POWermeter | Copies power value from power meter to attenuator module ref. power parameter | page 189 |
| :POWer:UNit/? | Sets or queries power unit used (dBm or W) | page 192 |
| [:STATe]/? | Sets a source's or attenuators output terminals to open or closed or returns the current status of a source's or attenuators output terminals. | page 123 |
| **:READ[*n*][:CHANnel[*m*]]** | | |
| [:SCALar]:POWer[:DC]? | Reads the current power value from a sensor. | page 92 |
| :POWer[:DC]:ALL? | Reads all available power meter channels. | page 91 |
| :POWer[:DC]:ALL:CONFig? | Return all the slot and channel number of every available power meter channel. | page 92 |
| [:SCALar]:RETurnloss? | Reads the current return loss value. | page 93 |
| **:ROUTe[*n*]** | | |
| [:CHANnel[m]]/? | Sets or returns the channel route between two ports. | page 206 |
| [:CHANnel[m]]:CONFig? | Reads the switch configuration of an instrument. | page 207 |
| [:CHANnel[m]]:CONFig:ROUTe? | Reads the allowed switch routes of an instrument. | page 207 |

| Command | Description | Page |
|---|---|---|
| **:SENSe[*n*][:CHANnel[*m*]]:CORRection** | | |
| [:LOSS][:INPut][:MAGNitude]/? | Sets or returns the value of correction data for a sensor. | page 94 |
| :COLLECT:ZERO | Executes a zero calibration of a sensor module. | page 94 |
| :COLLECT:ZERO? | Returns the current zero state of a sensor module. | page 95 |
| :COLLECT:ZERO:ALL | Executes a zero calibration of all sensor modules. | page 95 |
| **:SENSe[*n*][:CHANnel[m]]:FUNCtion** | | |
| :PARameter:LOGGing/? | Sets or returns the number of samples and the averaging time, $t_{avg}$, for logging. | page 97 |
| :PARameter:MINMax/? | Sets or returns the minmax mode and the window size. | page 98 |
| :PARameter:STABility/? | Sets or returns the total time, delay time and the averaging time, $t_{avg}$, for stability. | page 99 |
| :RESult? | Returns the data array of the last function. | page 100 |
| :RESult:BLOCk? | Returns a specified binary block from the data array for the last power meter data acquisition function. | page 101 |
| :RESult:MAXBlocksize? | Returns the maximum block size for power meter data acquisition functions. | page 102 |
| :RESult:MONitor? | For return loss module, returns monitor diode data array of last function. | page 103 |
| :STATe/? | Enables/disables the function mode or returns whether the function mode is enabled. | page 104 |
| :THReshold/? | Sets or returns the threshold value and the start mode. | page 105 |
| **:SENSe[*n*][:CHANnel[*m*]]:POWer** | | |
| :ATIMe/? | Sets or returns the average time of a sensor. | page 106 |
| :RANGe[:UPPer]/? | Sets or returns the most positive signal entry expected for a sensor. | page 106 |
| :RANGe:MONitor[:UPPer]/? | Sets or returns the range of the monitor diode within a return loss module. | page 108 |
| :RANGe:AUTO/? | Sets or returns the range of a sensor to produce the most dynamic range without overloading. | page 109 |
| :REFerence/? | Sets or returns the reference level of a sensor. | page 110 |
| :UNIT/? | Sets or returns the units used for absolute readings on a sensor. | page 115 |
| :WAVelength/? | Sets or returns the wavelength for a sensor. | page 115 |

| Command | Description | Page |
|---|---|---|
| **:SENSe[*n*][:CHANnel[*m*]]:POWer:Reference** | | |
| :DISPlay | Sets the reference level for a sensor from the input power level. | page 112 |
| :STATe/? | Sets or returns whether sensor results are in relative or absolute units. | page 112 |
| :STATe:RATio/? | Sets or returns whether sensor results are displayed relative to a channel or to an absolute reference. | page 114 |
| :COLLect:VALues? | Returns current calibration values. | page 110 |
| :TERMination? | Returns T-Value | page 110 |
| **:SENSe[n]:[CHANnel[m]]:RETurnloss** | | |
| :CALibration:FACTory | Overwrites the current calibration values with the factory-set calibration settings for all sources. | page 117 |
| :CALibration:COLLect:REFLectance | Starts the calibration and saves the calibration values for a defined reflectance reference measurement for the currently selected source. | page 117 |
| :CALibration:COLLect:TERMination | Starts the calibration and saves the calibration values for a defined termination reference measurement for the currently selected source. | page 118 |
| :CALibration:TERMination? | Queries the T-value (termination calibration value) for the return loss module. | page 118 |
| :CALibration:VALues? | Returns the current calibration values. | page 119 |
| :CORRection:FPDelta[l]/? | Sets the front panel delta, that is, the loss correction value. | page 120 |
| :CORRection:REFLectance[l]/? | Sets the Return Loss Reference, the return loss value of your reference reflector. | page 121 |
| **:SLOT[*n*]** | | |
| :EMPTy? | Returns whether the module slot is empty. | page 81 |
| :IDN? | Returns information about the module. | page 81 |
| :OPTions? | Returns the module's options. | page 81 |
| :TST? | Returns the latest selftest results for a module. | page 82 |
| **:SLOT[*n*][:HEAD[*m*]]** | | |
| :EMPTy? | Returns whether an optical head is connected. | page 82 |
| :IDN? | Returns information about the optical head. | page 82 |
| :OPTions? | Returns the optical head's options. | page 83 |
| :TST? | Returns the latest selftest results for an optical head. | page 83 |

| Command | Description | Page |
|---|---|---|
| :WAVelength:RESPonse? | Returns the wavelength response from the module with wavelength calibration. | page 84 |
| :WAVelength:RESPonse:CSV? | Returns the wavelength response from the module with wavelength calibration. | page 85 |
| :WAVelength:RESPonse:SIZE? | Returns the no. of elements in the wavelength response table. | page 85 |
| **[:SOURce[*n*]][:CHANnel[*m*]]** | | |
| :MODout/? | Returns the mode of the modulation output mode of the BNC connector on the front panel of Tunable Laser modules. | page 139 |
| **[:SOURce[*n*]][:CHANnel[*m*]]:AM** | | |
| [:INTernal]:FREQuency[*{}*]/? | Sets or returns the frequency of an internal signal source. | page 124 |
| :SOURce[*{}*]/? | Sets or returns a source for the modulating system. | page 126 |
| :STATe[*{}*]/? | Turns Amplitude Modulation of a source on or off or queries whether Amplitude Modulation is on or off. | page 127 |
| :COHCtrl:COHLevel[*{}*]/? | Sets or returns the coherence level. | page 128 |
| **[:SOURce[*n*]][:CHANnel[*m*]]:FM** | | |
| :SOURce[*{}*]/? | Sets or returns the type of frequency modulaion employed, specifically Stimulated Brillouin Scattering (SBS) control. | page 129 |
| :STATe[*{}*]/? | Turns Frequency Modulation of a source on or off or queries whether Frequency Modulation is on or off. | page 130 |
| :SBSCtrl:FREQuency[*{}*]/? | Sets or returns the frequency of SBS Control modulation. | page 131 |
| :SBSCtrl:LEVel[*{}*]/? | Sets or returns the level of SBS Control modulation (as a percentage of maximum) | page 131 |
| **[:SOURce[*n*]][:CHANnel[*m*]:]POWer** | | |
| [:LEVel][:IMMediate][:AMPLitude[*{}*]] | Sets the laser output power of a source. | page 148 |
| [:LEVel][:IMMediate][:AMPLitude[*{}*]]? | Returns the laser output power of a source. | page 149 |
| [:LEVel]:RISetime[*{}*]/? | Sets or returns the laser rise time of a source. | page 150 |
| :ATTenuation[*{}*]/? | Sets or returns the attenuation level for a source. | page 145 |
| :STATe/? | Sets or returns the state of the source output signal. | page 151 |
| :UNIT/? | Sets or returns the power units. | page 151 |
| :WAVelength/? | Sets or returns the wavelength source of a dual-wavelength source. | page 152 |

| Command | Description | Page |
|---|---|---|
| **[:SOURce[*n*]][:CHANnel[*m*]:]POWer:ATTenuation[*l*]** | | |
| :AUTO/? | Selects Automatic or Manual Attenuation Mode for a source or returns the selected mode. | page 146 |
| :DARK/? | Enables/disables 'dark' position on a source or returns whether 'dark' position is active for a source. | page 147 |
| **[:SOURce[*n*]][:CHANnel[*m*]:]READout** | | |
| :DATA? | Returns number of datapoints returned by the [:SOURce[n]][:CHANnel[m]:READout:POINts? on page 155 command. | page 153 |
| :DATA:BLOCk? | Returns a specified binary block from either a lambda logging operation, or maximum power at wavelength characteristic. | page 153 |
| :DATA:MAXBlocksize? | Returns the maximum blocksize that a lambda logging, or maximum power at wavelength characteristic will return. | page 153 |
| :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. | page 155 |
| **[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength** | | |
| [:CW[*l*]:FIXED] | Sets the absolute wavelength of a source. | page 155 |
| [:CW[*l*]:FIXED[*l*]]? | Returns the absolute wavelength of a source. | page 156 |
| :FREQuency[*l*]/? | Sets the frequency difference used to calculate a relative wavelength for a source. | page 163 |
| :REFerence[*l*]? | Returns the reference wavelength of a source. | page 164 |
| **[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:CORRection** | | |
| :ARA | Realigns the laser cavity. | page 158 |
| :ARA:ALL | Realigns the laser cavity of every tunable laser source in the mainframe. | page 159 |
| :AUTocalib | Sets or returns tunable laser source Auto Calibration state. | page 159 |
| :ZERO | Executes a wavelength zero. | page 161 |
| :ZERO:ALL | Executes a wavelength zero on every tunable laser source in the mainframe. | page 161 |
| :ZERO:TEMPerature:ACTual? | Reports the current lambda zero temperature | page 161 |
| :ZERO:TEMPerature:DIFFerence? | Reports the temperature difference required to trigger an auto lamda zero. | page 162 |
| :ZERO:TEMPerature:LASTzero? | Reports the temperature at which the last auto lamda zero took place. | page 162 |

| Command | Description | Page |
|---|---|---|
| :ZERO:AUTO | Forces an auto lamda zero. This is quicker than the equilavent manual process. | page 162 |
| **[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:REFerence** | | |
| :DISPlay | Sets the reference wavelength of a source to the value of the output wavelength. | page 164 |

| Command | Description | Page |
|---|---|---|
| **[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:SWEep** | | |
| :CHECkparams? | Returns whether sweep parameters set are consistent. | page 165 |
| :CYCLes/? | Sets or returns the number of cycles. | page 166 |
| :DWELl/? | Sets or returns the dwell time. | page 167 |
| :EXP? | Returns number of triggers (used to configure power meter). | page 168 |
| :FLAG? | Returns whether waiting for trigger, or logging data available. | page 169 |
| :LLOGging/? | Switches lambda logging on or off or queries the state of lambda logging. | page 170 |
| :MODE/? | Sets or returns the sweep mode. | page 171 |
| :PMAX? | Returns the highest permissible power for a wavelength sweep. | page 171 |
| :REPeat/? | Sets or returns the repeat mode. | page 173 |
| :SOFTtrigger | Sends a soft trigger. | page 174 |
| :SPEed/? | Sets or returns the speed for continuous sweeping. | page 174 |
| :STARt/? | Sets or returns the start point of the sweep. | page 175 |
| :STOP/? | Sets or returns the end point of the sweep. | page 176 |
| [:STATe]/? | Stops, starts, pauses or continues a wavelength sweep or returns the the state of a sweep. | page 176 |
| **[:SOURce[*n*]][:CHANnel[*m*]:]WAVelength:SWEep:STEP** | | |
| :NEXT | Performs the next sweep step. | page 178 |
| :PREVious | Performs the previous sweep step again. | page 178 |
| [:WIDTh]/? | Sets or returns the width of the sweep step. | page 179 |

| Command | Description | Page |
|---|---|---|
| :SPECial | | |
| :REBoot | Reboots the mainframe and all modules. | page 85 |
| :STATus[*n*] | | |
| :PRESet | Presets all Enable Registers. | page 61 |
| **:STATus:OPERation** | | |
| [:EVENt]? | Returns the Operational Status Event Summary Register (OESR). | page 59 |
| [:EVENt]:LEVel*1*? | Returns the Operational Status Event Summary Register for slots 15 - 17 of the 8166A/B Lightwave Multichannel System. | page 57 |
| :CONDition? | Returns the Operational Status Condition Summary Register. | page 59 |
| :CONDition:LEVel*1*? | Returns the Operational Status Condition Summary Register for slots 15 - 17 of the 8166A/B Lightwave Multichannel System. | page 58 |
| :ENABle/? | Sets or queries the Operational Status Enable Summary Mask. | page 60 |
| :ENABle:LEVel*1*/? | Sets or queries the Operational Status Enable Summary Mask for slots 15 - 17 of the 8166A/B Lightwave Multichannel System. | page 58 |
| **:STATus*n*:OPERation** | | |
| [:EVENt]? | Returns the Operational Slot Status Event Register for slot *n*. | page 59 |
| :CONDition? | Returns the Operational Slot Status Condition Register for slot *n*. | page 59 |
| :ENABle/? | Sets or queries the Operation Slot Status Enable Mask for slot *n*. | page 60 |
| **:STATus:QUEStionable** | | |
| [:EVENt]? | Returns the Questionable Status Event Summary Register. | page 66 |
| [:EVENt]:LEVel*1*? | Returns the Questionable Status Event Summary Register for slots 15 - 17 of the 8166A/B Lightwave Multichannel System. | page 66 |
| :CONDition? | Returns the Questionable Status Condition Summary Register. | page 66 |
| :CONDition:LEVel*1*? | Returns the Questionable Status Condition Summary Register for slots 15 - 17 of the 8166A/B Lightwave Multichannel System. | page 65 |
| :ENABle/? | Sets or queries the Questionable Status Enable Summary Mask. | page 67 |
| :ENABle:LEVel*1*/? | Sets or queries the Questionable Status Enable Summary Mask for slots 15 - 17 of the 8166A/B Lightwave Multichannel System. | page 66 |

| Command | Description | Page |
|---|---|---|
| **:STATus*n*:QUEStionable** | | |
| [:EVENt]? | Returns the Questionable Slot Status Event Register for slot *n*. | page 66 |
| :CONDition? | Returns the Questionable Slot Status Condition Register for slot *n*. | page 66 |
| :ENABle/? | Sets or queries the Questionable Slot Status Enable Mask for slot n. | page 67 |
| **:SYSTem** | | |
| :DATE/? | Sets or returns the instrument's internal date. | page 68 |
| :ERRor? | Returns the contents of the instrument's error queue. | page 68 |
| :HELP:HEADers? | Returns a list of GPIB commands. | page 69 |
| :PRESet | Sets all parameters to their default values. | page 69 |
| :TIME/? | Sets or returns the instrument's internal time. | page 69 |
| :VERSion? | Returns the instrument's SCPI version. | page 70 |
| **:SYSTem:COMMunicate:** | | |
| :GPIB[:SELF]:ADDRess? | Sets or returns the GPIB address. | page 71 |
| :ETHernet:MACaddress? | Get the MAC address of the network adapter. | page 72 |
| :ETHernet:IPADdress:CURRent? | Get the current IP address of the instrument. | page 72 |
| :ETHernet:SMASk:CURRent? | Get the currently used subnet mask. | page 73 |
| :ETHernet:HOSTname:CURRent? | Get the current host name. | page 73 |
| :ETHernet:DOMainname:CURRent? | Get the currently used domain name. | page 73 |
| :ETHernet:DGATeway:CURRent? | Get the currently used default gateway. | page 74 |
| :ETHernet:DHCP:ENABle/? | Enable or disable DHCP. Returns whether DHCP is enabled or disabled. | page 74 |
| :ETHernet:HOSTname/? | Set and returns the host name. | page 75 |
| :ETHernet:IPADdress/? | Set and returns the IP address of the system manually. | page 75 |
| :ETHernet:DOMainname/? | Set and returns the domain name. | page 76 |
| :ETHernet:SMASk/? | Set and returns the subnet mask. | page 77 |
| :ETHernet:DGATeway/? | Set and returns the default gateway. | page 77 |
| :ETHernet:RESTart | Restart the system's network interface with the new parameters. | page 78 |

| Command | Description | Page |
|---|---|---|
| :TRIGger | Generates a hardware trigger. | |
| :CONFiguration/? | Sets or returns trigger configuration. | page 214 |
| **:TRIGger:CONFiguration** | | |
| :EXTended/? | Sets or returns extended trigger configuration. | page 217 |
| :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. | page 215 |
| **:TRIGger[*n*][CHANnel[*m*]]** | | |
| :INPut/? | Sets or returns the incoming trigger response . | page 210 |
| :OFFset/? | Sets or returns the number of incoming triggers received before data logging begins | page 212 |
| :INPut:REARm/? | Re-arms input trigger | page 211 |
| :OUTPut/? | Sets or returns the outgoing trigger response. | page 213 |
| :OUTPut:REARm/? | Re-arms output trigger | page 214 |

# 3 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 20.
- 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 20 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:<br>Error queue<br>Standard event status register (SESR)<br>Status byte register (STB)<br>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 canceled. |
| parameters: | none |
| response: | none |
| example: | *CLS |

| | | | |
|---|---|---|---|
| command: | ***ESE** | | |
| syntax: | *ESE<wsp><value><br>$0 \leq$ *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).<br>The register is cleared:<br>at power-on,<br>by sending a value of zero.<br>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>:<br>*For example.* |
| | MMMMMMMM — manufacturer, for example Keysight Technologies<br>*mmmm* — instrument model number (for example 8164B)<br>*ssssssss* — serial number<br>*rrrrrrrrr* — firmware revision level |
| example: | *IDN? -> Keysight Technologies,8164B,DE42100168,V5.25(72634)<END> |
| | Depending on the date of production, the manufacturer will be reported as Keysight Technologies, Agilent Technologies or HEWLETT-PACKARD.<br>See :SLOT[n]:IDN? on page 81 for information on module identity strings and :SLOT[n]:HEAD[n]:IDN? on page 82 for information on optical head identity strings. |

| | |
|---|---|
| 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.<br>Some module firmware includes commands that set a "StatNOPC" flag during execution to indicate that the module is busy. *OPC blocks the GPIB bus to all commands until every module hosted by the instrument is no longer busy.<br>The following actions cancel the *OPC command (and put the instrument into Operation Complete, Command Idle State):<br>Power-on<br>the Device Clear Active State is asserted on the interface.<br>*CLS<br>*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.<br>Some module firmware includes commands that set a "StatNOPC" flag during execution to indicate that the module is busy. If a module is executing a command which generates a "StatNOPC" flag, the GPIB bus is not blocked to a command to another module. A second command to a busy module is blocked until the module flag "StatOK" is set. Taking advantage of this feature, and using *OPC? in a loop to query until the instrument returns 1, can lead to useful gains in program execution efficiency.<br>The following actions cancel the *OPC? query (and put the instrument into Operation Complete, Command Idle State):<br>Power-on<br>the Device Clear Active State is asserted on the interface.<br>*CLS<br>*RST |
| parameters: | none |
| response: | 1<END> is returned if all modules are ready to execute a new operation.<br>0<END> is returned if any module is busy. |
| 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.<br>Slots are listed starting with the lowest slot number, that is, slot 0 for the 8164A/B and Slot 1 for the 8163A/B and 8166A/B.<br><br>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.<br>Pending *OPC? actions are cancelled.<br>The instrument is placed in the idle state awaiting a command. The *RST command clears the error queue.<br>The *RST command is equivalent to the *CLS command AND the syst:preset command.<br>The following are not changed:<br>GPIB (interface) state<br>Instrument interface address<br>Output queue<br>Service request enable register (SRE)<br>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 (OSB) | 128 |
| | 6 | Not used | 0 |
| | 5 | Event Status Bit (ESB) | 32 |
| | 4 | Message Available (MAV) | 16 |
| | 3 | Questionable Status (QSB) | 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 $0 \leq value \leq 4294967296$): | | |
| | 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.<br>If 18 is returned, the modules in slots 1 and 4 have failed.<br>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. Some module firmware includes commands that set a "StatNOPC" flag during execution to indicate that the module is busy. *WAI blocks the GPIB bus to all commands until every module hosted by the instrument is no longer busy. 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 23.

| 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 ≤ *value* ≤ 32767): | | | | |
| | Bits | Mnemonics | | | Decimal Value |
| | | 8163A/B | 8164A/B | 8166A/B | |
| | 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 ≤ *value* ≤ 32767): | | | | |
| | Bits | Mnemonics | | | Decimal Value |
| | | 8163A/B | 8164A/B | 8166A/B | |
| | 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).<br>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)<br>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*1*?** | | | |
| syntax: | :STATus:OPERation[:EVENt]:LEVel*1*? | | | |
| description: | Returns the Operational Status Event Summary Register (OSESR) for slots 15 to 17 of the 8166A/B Lightwave Multichannel System. | | | |
| 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 | |
| | | 8166A/B | | |
| | 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*1*?** | | |
| syntax: | :STATus:OPERation:CONDition:LEVel*1*? | | |
| description: | Returns the Operational Status Condition Summary Register for slots 15 to 17 of the Keysight 8166B 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 | Decimal Value |
| | | 8166A/B | |
| | 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*1*** |
| syntax: | :STATus:OPERation:ENABle:LEVel*1*<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 8166A/B Lightwave Measurement System to affect the Status Byte (STB).<br>Setting a bit in this register to 1 enables the corresponding bit in the OSESR for slots 15 - 17 of the 8166A/B 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)<br>The default value is 0. |
| response: | none |
| example: | stat:oper:enab:level1 128 |

| | |
|---|---|
| command: | **:STATus:OPERation:ENABle:LEVel*1*?** |
| syntax: | :STATus:OPERation:ENABle:LEVel*1*? |
| description: | Returns the OSESM for the OSESR for slots 15 - 17 of the 8166A/B 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 |
|---|---|---|
| 8-15 | Not used | 0 |
| 7 | Slot *n*: offset (λ) type bit 2 | 128 |
| 6 | Slot *n*: offset (λ) type bit 1 | 64 |
| 5 | Slot *n*: offset (λ) has been enabled | 32 |
| 4 | Slot *n*: shutter has been opened | 16 |
| 3 | Slot *n*: Zeroing ongoing | 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 |
| | 8-15 | Not used | 0 |
| | 7 | Slot *n*: offset (λ) type bit 2 | 128 |
| | 6 | Slot *n*: offset (λ) type bit 1 | 64 |
| | 5 | Slot *n*: offset (λ) enabled | 32 |
| | 4 | Slot *n*: shutter open | 16 |
| | 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> | | |
| | NOTE: | Only attenuator bits 5 to 7 are used to show whether the offset feature is used and which algorithm is used to calculate the wavelength dependent offset.<br>Bit 5 states if the feature is enabled or disabled. Bits 6 and 7 are decoded as shown below to say whether the attenuator uses saved, interpolated, or extrapolated values. | |

| Type | Bit 5 | Bit 6 | Bit 7 | Decimal Value |
|---|---|---|---|---|
| none | 0 | 0 | 0 | 0 |
| exact value | 1 | 0 | 0 | 32 |
| extrapolate below | 1 | 1 | 0 | 96 |
| extrapolate above | 1 | 0 | 1 | 160 |
| interpolated | 1 | 1 | 1 | 224 |

| | |
|---|---|
| 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 OSESR.<br>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 |
| | | 8163A/B | 8164A/B | 8166A/B | |
| | 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 |
| | | 8163A/B | 8164A/B | 8166A/B | |
| | 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).<br>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)<br>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*1*?** | | |
| syntax: | :STATus:QUEStionable[:EVENt]:LEVel*1*? | | |
| description: | Returns the Questionable Status Event Summary Register (QSESR) for slots 15 to 17 of the 8166A/B 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 | Decimal Value |
| | | 8166A/B | |
| | 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*1*?** | | |
| syntax: | :STATus:QUEStionable:CONDition:LEVel*1*? | | |
| description: | Returns the Questionable Status Condition Summary Register for slots 15 to 17 of the 8166A/B 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 | Decimal Value |
| | | 8166A/B | |
| | 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*1*** |
| syntax: | :STATus:QUEStionable:ENABle:LEVel*1*<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 8166A/B Lightwave Measurement System to affect the Status Byte (STB).<br>Setting a bit in this register to 1 enables the corresponding bit in the OSESR for slots 15 - 17 of the 8166A/B 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)<br>The default value is 0. |
| response: | none |
| example: | stat:oper:enab:level1 128 |

| | |
|---|---|
| command: | **:STATus:QUEStionable:ENABle:LEVel*1*?** |
| syntax: | :STATus:QUEStionable:ENABle:LEVel*1*? |
| description: | Returns the QSESM for the QSESR for slots 15 - 17 of the 8166A/B 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 |
|---|---|---|
| 11-15 | Not Used | 0 |
| 10 | Slot *n*: Lambda zeroing has been recommended | 1024 |
| 9 | Slot *n*: Beam Path Protection on (shutter off) | 512 |
| 8 | Slot *n*: Coherence control is uncalibrated | 256 |
| 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 |
|---|---|---|
| 11 - 15 | Not Used | |
| 10 | Slot *n*: Lambda zeroing is recommended | 1024 |
| 9 | Slot *n*: Beam Path Protection on (shutter off) | 512 |
| 8 | Slot *n*: Coherence control is uncalibrated | 256 |
| 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 QSESR.<br>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),<br>the second value is the month, and<br>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 14).<br>Each error has the error code and a *short* description of the error, separated by a comma, for example 0, "No error".<br>Error codes are numbers in the range -32768 and +32767.<br>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.<br>The following are not affected by this command:<br>the GPIB (interface) state,<br>the backlight and contrast of the display,<br>the interface address,<br>the output and error queues,<br>the Service Request Enable register (SRE),<br>the Status Byte (STB),<br>the Standard Event Status Enable Mask (SESEM), and<br>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),<br>the second value is the minute, and<br>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> |

## System Communicate - The :SYST:COMMunicate sub tree.

We recommend you change network settings using GPIB or the local user interface.

**NOTE** The instrument does not close open connections when restarting the network interface (:SYSTem:COMMunicate:ETHernet:RESTart). This means the number of possible connections is reduced by the number of previously open connections. However, the instrument does make sure connections are still alive. It should release unused open connections after about two minutes.

Some notes on DHCP/AutoIP/DNS

- If DHCP is enabled but no DHCP server is found, the instrument tries to use AutoIP as a fallback. This may take about 2 minutes.
- Depending on the available network capabilities, the instrument tries to tell the DNS server its host name or read the host and domain named it has been assigned.

MAC address:

The Media Access Control (MAC) number is a unique number associated with each network adapter.

| | | |
|---|---|---|
| command: | **:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess** | |
| syntax: | :SYSTem:COMMunicate:GPIB[:SELF]:ADDRess<wsp><GPIB Address> | |
| description: | Sets the GPIB address. | |
| parameters: | The GPIB Address | Values allowed 0-30<br>21 is often reseverved by the GPIB Controller. |
| 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> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:MACaddress?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:MACaddress? |
| description: | Get the MAC address of the network adapter. |
| parameters: | none |
| response: | response string (hexadecimal value without a prefix or separators). |
| example: | :syst:comm:eth:mac? –> “0007E014AE08”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:IPADdress:CURRent?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:IPADdress:CURRent? |
| description: | Get the current IP address of the instrument. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:ipad:curr? –> “192.132.13.2”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:SMASk:CURRent?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:SMASk:CURRent? |
| description: | Get the currently used subnet mask. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:smas:curr? –> “255.255.255.0”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:HOSTname:CURRent?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:HOSTname:CURRent? |
| description: | Get the current host name. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:host:curr? –> “A-8164B-1234567”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DOMainname:CURRent?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:DOMainname:CURRent? |
| description: | Get the currently used domain name. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:dom:curr? –> “.companyame.com”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DGATeway:CURRent?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:DGATeway:CURRent? |
| description: | Get the currently used default gateway. |
| parameters: | none |
| response: | string (maximum 79 characters) |
| example: | :syst:comm:eth:dgat:curr? –> “192.168.101.11“<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DHCP:ENABle** |
| syntax: | :SYSTem:COMMunicate:ETHernet:DHCP:ENABle |
| description: | Enable or disable DHCP |
| parameters: | boolean (0 \| 1 \| off \| on) |
| response: | none |
| example: | :syst:comm:eth:dhcp:enab on |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DHCP:ENABle?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:DHCP:ENABle? |
| description: | Check whether DHCP is enabled or disabled. |
| parameters: | none |
| response: | boolean (0 \| 1) |
| example: | :syst:comm:eth:dhcp:enab? –> 1<END> |

| command: | **:SYSTem:COMMunicate:ETHernet:HOSTname** |
|---|---|
| syntax: | :SYSTem:COMMunicate:ETHernet:HOSTname |
| description: | Set the host name. |
| parameters: | string (maximum 19 characters, though not all characters can be used)<br>The default host name is A-P...P-S...S; where P...P is the product Number, and S...S is as many of the last digits of the serial number as it takes to get a 15 character host name.<br>If you set an empty host name (“”), the host name will be set to its default value. |
| response: | none |
| example: | :syst:comm:eth:host “my8163B” |

| command: | **:SYSTem:COMMunicate:ETHernet:HOSTname?** |
|---|---|
| syntax: | :SYSTem:COMMunicate:ETHernet:HOSTname? |
| description: | Get the host name. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:host? –> “my8163B”<END |

| command: | **:SYSTem:COMMunicate:ETHernet:IPADdress** |
|---|---|
| syntax: | :SYSTem:COMMunicate:ETHernet:IPADdress |
| description: | Set the IP address of the system manually (used if DHCP is disabled). |
| parameters: | string (Up to four groups of up to 3 digits, groups separated by ".". Groups with leading zeroes are interpreted as octal numbers.) |
| response: | none |
| example: | :syst:comm:eth:ipad “192.132.13.2” |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:IPADdress?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:IPADdress? |
| description: | Get the manually set IP address of the system. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:ipad? –> “192.132.13.2”<END> |

| | |
|---|---|
| command: | **SYSTem:COMMunicate:ETHernet:DOMainname** |
| syntax: | SYSTem:COMMunicate:ETHernet:DOMainname |
| description: | Set the domain name (used if DHCP is disabled). |
| parameters: | string |
| response: | none |
| example: | :syst:comm:eth:dom “.companyname.com” |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DOMainname?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:DOMainname? |
| description: | Get the domain name. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:dom? –> “.companyname.com”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:SMASk** |
| syntax: | :SYSTem:COMMunicate:ETHernet:SMASk |
| description: | Set the subnet mask. |
| parameters: | string (Up to four groups of up to 3 digits, groups separated by ".". Groups with leading zeroes are interpreted as octal numbers.) |
| response: | none |
| example: | :syst:comm:eth:smas “255.255.255.0“ |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:SMASk?** |
| syntax: | :SYSTem:COMMunicate:ETHernet:SMASk? |
| description: | Get the subnet mask. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:smas? –> “255.255.255.0”<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DGATeway** |
| syntax: | :SYSTem:COMMunicate:ETHernet:DGATeway |
| description: | Set the default gateway. |
| parameters: | string (Up to four groups of up to 3 digits, groups separated by ".". Groups with leading zeroes are interpreted as octal numbers.) |
| response: | none |
| example: | :syst:comm:eth:dgat “192.168.101.11“ |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:DGATeway?** |
| syntax: | SYSTem:COMMunicate:ETHernet:DGATeway? |
| description: | Get the default gateway. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:dgat? –> “192.168.101.11“<END> |

| | |
|---|---|
| command: | **:SYSTem:COMMunicate:ETHernet:RESTart** |
| syntax: | :SYSTem:COMMunicate:ETHernet:RESTart |
| description: | Restart the system’s network interface with the new parameters.<br>This command only works if the instrument has a working network connection at the time the command is issued.<br>If not you either have to wait until the instrument decides on an IP address using AutoIP or reboot the instrument. |
| parameters: | none |
| response: | string |
| example: | :syst:comm:eth:rest |

# 4 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.
- Signal Conditioning commands that control Attenuator modules.
- **TRIGger** subsystem commands that control triggering.

## Root Layer Command

| | | |
|---|---|---|
| command: | **:LOCK** | |
| syntax: | :LOCK<wsp><boolean>, <value> | |
| description: | Switches the lock off and on.<br>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<br>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: | Queries the current state of the lock. | |
| parameters: | none | |
| response: | A *boolean* value: | 0: lock is switched off<br>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: | Queries whether the module slot is empty. | |
| parameters: | none | |
| response: | A *boolean* value: | 0: there is a module in the slot<br>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: | MMMMMMMM<br>*mmmm*<br>*sssssss*<br>*rrrrrrrrr* | manufacturer, for example Keysight Technologies<br>instrument model number (for example 81533B)<br>serial number<br>date of firmware revision |
| example: | slot1:idn? ->\<br>HEWLETT-PACKARD, 81533B,3411G06054,07-Aug-98<END> | |
| | Depending on the date of production, the manufacturer will be reported as Keysight Technologies, Agilent Technologies or HEWLETT-PACKARD.<br>See *IDN? on page 50 for information on mainframe identity strings. See :SLOT[n]:HEAD[n]:IDN? on page 82 for information on optical head 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. |
| | 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: | Queries whether an optical head is connected. | |
| parameters: | none | |
| response: | A *boolean* value: | 0: there is a module in the slot<br>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 |
| 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: | MMMMMMMM<br>*mmmm*<br>*ssssssss*<br>*rrrrrrrrr* | manufacturer<br>instrument model number (for example 81520A)<br>serial number<br>date of firmware revision |

| | |
|---|---|
| example: | slot1:head:idn? –><br>Keysight Technologies,81623B,DE41300402,V4.01(1126) |
| affects: | Optical heads |
| | Depending on the date of production, the manufacturer will be reported as Keysight Technologies, Agilent Technologies or HEWLETT-PACKARD.<br><br>See *IDN? on page 50 for information on mainframe identity strings,<br>and :SLOT[n]:HEAD[n]:IDN? on page 82 for information on module identity strings. |

| | |
|---|---|
| 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. |
| | This command does not perform a selftest. Use selfTeST command, (page 53), 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: | **:SLOT[*n*]:HEAD[*m*]:WAVelength:RESPonse?** |
| syntax: | :SLOT[*n*]:HEAD[*m*]:WAVelength:RESPonse? |
| description: | Returns the wavelength response from a wavelength calibrated module in binary format. |
| response: | Wavelength Response table as a *binary block*. |
| response format: | One 8 byte long wavelength calibration value pair consisting of a 4 byte long float for wavelength and a 4 byte long float for the scalar calibration factor.<br>For more information on binary block formats see Data Types on page 17. |
| example: | slot1:head1:wav:resp? -> #536570..... |
| affects: | Attenuator with power control, all powermeters, return loss modules |

| | |
|---|---|
| command: | **:SLOT[*n*]:HEAD[*m*]:WAVelength:RESPonse:CSV?** |
| syntax: | :SLOT[*n*]:HEAD[*m*]:WAVelength:RESPonse:CSV? |
| description: | Returns the wavelength response from the attenuator module in CSV format. |
| response: | Wavelength Response table as a *string* |
| response format: | The string is a comma separated value (CSV) list and can be written to a file and be processed with a spreadsheet program.<br>List format:<br>λ1, c1\n<br>λ2, c2\n<br>.....<br>λn, cn\n<br>"," separates wavelength and response factor<br>"\n" = ASCII code 10 separate value pairs |
| example: | slot1:head1:wav:resp:csv? -> 1200e-6,2.019\n 1210e-6,<br>1.956\n... |
| affects: | Attenuator with power control, all powermeters, return loss modules |

| | |
|---|---|
| command: | **:SLOT[*n*]:HEAD[*m*]:WAVelength:RESPonse:SIZE?** |
| syntax: | :SLOT[*n*]:HEAD[*m*]:WAVelength:RESPonse:SIZE? |
| description: | Returns the number of elements in the wavelength response table. |
| response | Number of elements in the wavelength table as an *integer* value |
| example: | slot2:head1:wav:resp:size? -> 50<END> |
| affects: | Attenuator with power control, all powermeters, return loss modules |

| | |
|---|---|
| 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.

## Keysight 81635A and Keysight 81619A - Master and Slave Channels

For the Keysight 81635A Dual Power Sensor and Keysight 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 6 can only be configured using the master channel.

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

| Command | Page |
|---|---|
| :INITiate[n]:[CHANnel[m]][:IMMediate] | page 89 |
| :INITiate[n]:[CHANnel[m]]:CONTinuous/? | page 91 |
| :READ[n][:CHANnel[m]][:SCALar]:POWer[:DC]? | page 92 |
| :SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO | page 94 |
| :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:LOGGing/? | page 97 |
| :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:MINMax/? | page 98 |
| :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:STABility/? | page 99 |
| :SENSe[n][:CHANnel[m]]:FUNCtion:STATe/? | page 104 |
| :SENSe[n]:[CHANnel[m]]:POWer:ATIME/? | page 105 |
| :SENSe[n]:[CHANnel[m]]:POWer:RANGe:AUTO/? | page 109 |
| :TRIGger[n][:CHANnel[m]]:INPut/? | page 210 |
| :TRIGger[n][:CHANnel[m]]:INPut:REARm/? | page 211 |
| :TRIGger[n][:CHANnel[m]]:OUTPut/? | page 213 |
| :TRIGger[n][:CHANnel[m]]:OUTPut:REARm/? | page 214 |

The commands listed in Table 7 are independent for both master and slave channels.

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

| Command | Page |
|---|---|
| :FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]? | page 88 |
| :ROUTe[n][:CHANnel[m]]/? | page 206 |
| :ROUTe[n][:CHANnel[m]]:CONFig? | page 207 |
| :ROUTe[n][:CHANnel[m]]:CONFig:ROUTe? | page 207 |
| :SENSe[n]:[CHANnel[m]]:CORRection[:LOSS][:INPut] [:MAGNitude]/? | page 94 |
| :SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO? | page 95 |
| :SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO:ALL | page 95 |
| :SENSe[n][:CHANnel[m]]:FUNCtion:RESult? | page 100 |
| :SENSe[n]:[CHANnel[m]]:POWer:RANGe[:UPPer]/? | page 106 |
| :SENSe[n]:[CHANnel[m]]:POWer:REFerence/? | page 110 |
| :SENSe[n]:[CHANnel[m]]:POWer:REFerence:DISPlay | page 112 |
| :SENSe[n]:[CHANnel[m]]:POWer:REFerence:STATe/? | page 112 |
| :SENSe[n]:[CHANnel[m]]:POWer:REFerence:STATe:RATio/? | page 114 |
| :SENSe[n]:[CHANnel[m]]:POWer:UNIT/? | page 115 |
| :SENSe[n]:[CHANnel[m]]:POWer:WAVelength/? | page 115 |

| | |
|---|---|
| command: | **:FETCh[*n*][:CHANnel[*m*]][:SCAlar]:POWer[:DC]?** |
| syntax: | :FETCh[*n*]:[CHANnel[*m*]][:SCAlar]:POWer[:DC]? |
| description: | Reads the current power meter value, or for a return loss module returns current power value at return loss diode (back reflection path). 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 91) or a directly preceding immediate software trigger (see :INITiate[n]:[CHANnel[m]][:IMMediate] on page 89).<br>It returns the 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 value as a float value in dBm,W or dB. |
| | If the reference state is absolute, units are dBm or W.<br>If the reference state is relative, units are dB. |
| example: | fetc1:pow? -> +6.73370400E-04<END> |
| affects: | All power meters, return loss modules, and attenuators with power sensors |
| 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 91) or a directly preceding immediate software trigger (see :INITiate[n]:[CHANnel[m]][:IMMediate] on page 89).<br>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 value as a float value in dB. |
| example: | fetc1:ret? -> +6.73370400E-00<END> |
| affects: | All return loss modules |

| | |
|---|---|
| command: | **:FETCh[*n*][:CHANnel[*m*]][:SCAlar]:MONitor?** |
| syntax: | :FETCh[*n*]:[CHANnel[*m*]][:SCAlar]:MONitor? |
| description: | Reads current power value at a return loss module's monitor diode (forward path). 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 91) or a directly preceding immediate software trigger (see :INITiate[n]:[CHANnel[m]][:IMMediate] on page 89).<br>It returns the monitor 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 value as a float value in W or dBm. |
| example: | fetc1:mon? -> +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, 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<br>1 or ON: measure continuously |
| response: | none | |

| | |
|---|---|
| example: | init2:cont 1 |
| affects: | All power meters, 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<br>1 or ON: measurement is continuous |
| example: | init2:cont? -> 1<END> |
| affects: | All power meters, return loss modules. |
| dual sensors: | Can only be sent to master channel, slave channel parameters are identical. |

| | |
|---|---|
| command: | **:READ[*n*][:CHANnel[*m*]][SCALar:]:POWer:ALL?** |
| syntax: | :READ[*n*]:[CHANnel[*m*]]:POWer:[:DC]:ALL? |
| description: | Reads all available power meter channels. It provides its own software triggering and does not need a triggering command. |
| | The power meters must be running for this command to be effective. |
| parameters: | none |
| response: | 4-byte Intel *float* values in a binary block in Intel byte order. The values are ordered by slot and channel order.<br>See Data Types on page 17 for more information on Binary Blocks. |
| | Data values are always in Watt. |
| example: | read1:pow:all? -> interpreted as<br>+1.33555600E-006\|+1.34789100E-006\|+1.37456900E-006<END> |
| affects: | All power meters (v3.0x firmware or later). |
| dual sensors: | Master channels receive a read command, see:<br>:READ[n][:CHANnel[m]][:SCALar]:POWer[:DC]? on page 92.<br>Slave channels receive a fetch command, see:<br>:FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]? on page 88. |

| | |
|---|---|
| command: | **:READ[*n*][:CHANnel[*m*]]:POWer:ALL:CONFig?** |
| syntax: | :READ[*n*]:[CHANnel[*m*]]:POWer[:DC]:ALL:CONFig? |
| description: | Returns the slot and channel numbers for all available power meter channels.<br>Use this command to match returned power values to the appropriate slot and channel number. |
| parameters: | none |
| response: | A binary block (Intel byte order) consisting of 2-byte unsigned integer value pairs (so each pair has 4 bytes). The first member of the pair represents the the slot number, the second member of the pair represents the channel number. |
| example: | read1:pow:all:conf? -> interpreted as<br>1\|1\|1\|2\|12\|1<END><br>This 12-byte block means that there are three powermeters present:<br>Slot 1, Channel 1<br>Slot 1, Channel 2<br>Slot 12, Channel 1 |
| affects: | All power meters (v3.0x firmware or later). |
| dual sensors: | |

| | |
|---|---|
| command: | **:READ[*n*][:CHANnel[*m*]][:SCALar]:POWer[:DC]?** |
| syntax: | :READ[*n*]:[CHANnel[*m*]][:SCALar]:POWer[:DC]? |
| description: | Reads the current power meter value, or for a return loss module the power value at the return loss diode (back reflection path). It provides its own software triggering and does not need a triggering command.<br>If the software trigger system operates continuously (see :INITiate[n]:[CHANnel[m]]:CONTinuous? on page 91), this command is identical to :FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]? on page 88.<br>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 89) and then reading the power meter value. |
| | 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. |
| | If the reference state is absolute, units are dBm or W.<br>If the reference state is relative, units are dB. |

| | |
|---|---|
| example: | read1:pow? -> +1.33555600E-006<END> |
| affects: | All power meters and return loss modules and attenuator with power control |
| dual sensors: | Can only be sent to master channel, slave channel is also triggered.<br>To read a simultaneous result from the slave channel, send :FETCh[n][:CHANnel[m]][:SCAlar]:POWer[:DC]? on page 88 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.<br>If the software trigger system operates continuously (see :INITiate[n]:[CHANnel[m]]:CONTinuous? on page 91), this command is identical to :FETCh[n][:CHANnel[m]][:SCAlar]:RETurnloss? on page 88.<br>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 89) and then reading the power meter value. |
| | 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: | **:READ[*n*][:CHANnel[*m*]][:SCALar]:MONitor?** |
| syntax: | :READ[*n*]:[CHANnel[*m*]][:SCALar]:MONitor? |
| description: | Reads the power value at the monitor diode (forward path). It provides its own software triggering and does not need a triggering command.<br>If the software trigger system operates continuously (see Chapter , ":INITiate[n]:[CHANnel[m]]:CONTinuous?," on page 91), this command is identical to :FETCh[n][:CHANnel[m]][:SCAlar]:MONitor? on page 89.<br>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 89) and then reading the power meter value. |
| | 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 W or dBm |
| example: | read1:mon? -> +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><br><value>[DB\|MDB] |
| description: | Enters a calibration value for a module. |
| parameters: | The calibration factor as a *float* value<br>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. |

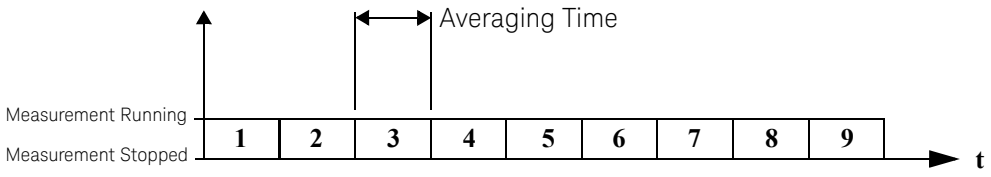
| | | |
|---|---|---|
| command: | **:SENSe[*n*]:[CHANnel[*m*]]:CORRection:COLLect:ZERO?** | |
| syntax: | :SENSe[*n*]:[CHANnel[*m*]]:CORREction:COLLect:ZERO? | |
| description: | Returns the status of the most recent zero command. | |
| parameters: | none | |
| response: | 0:<br>97<br>98<br>99<br>any other number: | zero succeeded without errors.<br>zero not done<br>zero failed<br>zero absorbed<br>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 on page 97 command to set parameters before you start a logging function using the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104 command.

| | | |
|---|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:LOGGing** | |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:LOGGing<wsp><data points>,<br><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 100 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. |
| | See :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104 for information on starting/stopping a data acquisition function. |
| | See :SENSe[n][:CHANnel[m]]:FUNCtion:RESult? on page 100 for information on accessing the results of a data acquisition function. |
| | See Table 8 on page -208 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 on page 98 command to set parameters before you start a MinMax function using the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104 command.

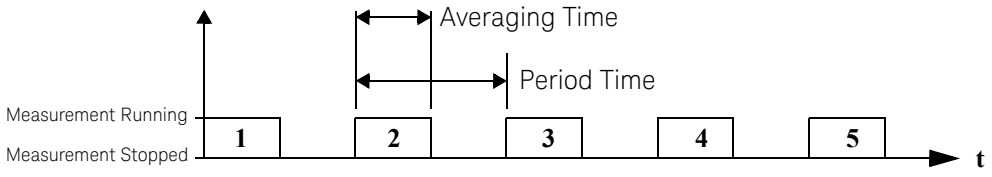
| | | |
|---|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:MINMax** | |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:MINMax<wsp><br>CONTinous\|WINDow\|REFResh,<data points> | |
| description: | Sets the MinMax mode and the number of data points for the MinMax data acquisition function. | |
| parameters: | CONTinous:<br>WINDow:<br>REFResh: | continuous MinMax mode<br>window MinMax mode<br>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.<br>See Chapter 3 of the 8163A/B Lightwave Multimeter, 8164A/B Lightwave Measurement System, & 8166A/B Lightwave Multichannel System User's Guide, for more information on MinMax mode. | |
| | See :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104 for information on starting/stopping a data acquisition function. | |
| | See :SENSe[n][:CHANnel[m]]:FUNCtion:RESult? on page 100 for information on accessing the results of a data acquisition function. | |
| | See Table 8 on page -208 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:<br>WIND:<br>REFR: | continuous MinMax mode<br>window MinMax mode<br>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 on page 99 command to set parameters before you start a stability function using the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104 command.

| | | |
|---|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:STABility** | |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:PARameter:STABility<wsp><br><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:<br>Period time:<br>Averaging time: | The total time from the start of stability mode until it is completed.<br>A new measurement is started after the completion of every period time.<br>A measurement is averaged over the averaging time. |



| | |
|---|---|
| | The total time should be longer than the period time.<br>The period time should be longer than the averaging time.<br>The number of data points is equal to the total time divided by the period time.<br>Total time, period time, and averaging time are time values in seconds.<br>If you specify no units in your command, seconds are used as the default. |
| | See :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:STABility? on page 100 for information on starting/stopping a data acquisition function. |
| | See :SENSe[n][:CHANnel[m]]:FUNCtion:RESult? on page 100 for information on accessing the results of a data acquisition function. |
| | See Table 8 on page -208 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,<br>+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.<br>For Logging and Stability Data Acquisition functions, one measurement value is a 4-byte-long **float** in Intel byte order.<br>For the MinMax Data Acquisition function, the query returns the minimum, maximum and current power values.<br>See Data Types on page 17 for more information on Binary Blocks. |

| | |
|---|---|
| | See How to Log Results on page 249 for information on logging using VISA calls. There are some tips about how to use float format specifiers to convert the binary blocks into **float** values. |
| | If you use LabView or Keysight VEE, we recommend using the Keysight 816x VXI*plug&play* Instrument Driver to perform the Logging and Stability Data Acquisition functions. |
| example: | sens1:func:res? ->\<br>returns a data array for Logging and Stability Data Acquisition functions\<br>sens1:func:res? -> #255\<br>Min: 7.24079E-04, Max: 7.24252E-04, Act: 7.24155E-04\<br>returns the minimum, maximum and current power values for the MinMax Data Acquisition function |
| affects: | All power meters and return loss modules |
| dual sensors: | Master and slave channels are independent. |
| | Return Loss modules:\<br>For Logging and Stability Data Acquisition functions, the data array contains power values.\<br>For the MinMax Data Acquisition function, the data array contains return loss values. |

| | |
|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:BLOCk?** |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:BLOCk?<wsp><offset>,<# of data points> |
| description: | Returns a specific binary block (Intel byte order) from the data array for the last data acquisition function. |
| parameters: | <*offset*> A zero based offset; the number of data points to ignore.\<br>*# data points* The number of data points (not bytes!) to return. |
| response: | The last stablility or logging data acquisition function's data array as a binary block.\<br>This function is not available for min-max measurements.\<br>One measurement value is a 4-byte-long float in Intel byte order.\<br>See Data Types on page 17 for more information on Binary Blocks. |
| example: | sens1:func:res:bloc? #5, 2 -> interpreted as\<br>7.24079E-04,7.24252E-04<end> |
| affects: | All power meters and return loss modules . |
| dual sensors: | Master and slave channels are independent. |
| | Return Loss modules:\<br>For Logging and Stability Data Acquisition functions, the data array contains power values. |

| | |
|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:MAXBlocksize?** |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:MAXBlocksize?<wsp><offset><# of data points> |
| description: | Returns the maximum block size for a single GPIB transfer for power meter data acquisition functions. If your application requires more data points please use SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:BLOCk? instead of SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult? |
| parameters: | none |
| response: | An ***integer*** value, number of data points.<br>See Data Types on page 17 for more information on Binary Blocks. |
| example: | |
| affects: | All power meters and return loss modules. |
| dual sensors: | Master and slave channels are independent. |

| | |
|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:MONitor?** |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:RESult:MONitor? |
| description: | Returns the monitor diode data array for the last data acquisition function. |
| parameters: | none |
| response: | The last data acquisition function's data array as a binary block.<br>For Logging and Stability Data Acquisition functions, one measurement value is a 4-byte-long **float** in Intel byte order.<br>For the MinMax Data Acquisition function, the query returns the minimum, maximum and current power values.<br>See Data Types on page 17 for more information on Binary Blocks. |
| | See How to Log Results on page 249 for information on logging using VISA calls. There are some tips about how to use float format specifiers to convert the binary blocks into **float** values. |
| | If you use LabView or Keysight VEE, we recommend using the Keysight 816x VXI*plug&play* Instrument Driver to perform the Logging and Stability Data Acquisition functions. |
| example: | sens1:func:res:mon? -><br>returns a data array for Logging and Stability Data Acquisition functions<br>sens1:func:res? -> #255<br>Min: 7.24079E-04, Max: 7.24252E-04, Act: 7.24155E-04<br>returns the minimum, maximum and current power values for the MinMax Data Acquisition function |
| affects: | All return loss modules |
| dual sensors: | Master and slave channels are independent. |
| | Return Loss modules:<br>For Logging and Stability Data Acquisition functions, the data array contains power values for the monitor diode.<br>For the MinMax Data Acquisition function, the data array contains return loss values for the monitor diode. |

| | | |
|---|---|---|
| command: | **:SENSe[*n*][:CHANnel[*m*]]:FUNCtion:STATe** | |
| syntax: | :SENSe[*n*][:CHANnel[*m*]]:FUNCtion:STATe<wsp><br>LOGGing\|STABility\|MINMax,STOP\|STARt | |
| description: | Enables/Disables the logging, MinMax, or stability data acquisition function mode. | |
| parameters: | LOGGing:<br>STABility:<br>MINMax:<br>STOP:<br>STARt: | Logging data acquisition function<br>Stability data acquisition function<br>MinMax data acquisition function<br>Stop data acquisition function<br>Start data acquisition function |
| | When you enable a logging data acquisition function for a 8163A/B 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 97 for more information on the logging data acquisition function. | |
| | 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<br>LOGGING_STABILITY<br>MINMAX<br>PROGRESS<br>COMPLETE | No function mode selected<br>Logging or stability data acquisition function<br>MinMax data acquisition function<br>Data acquisition function is in progress<br>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>,<br><threshold value>[PW\|NW\|UW\|MW\|Watt\|DBM] | |
| description: | Sets the start mode and the threshold value. | |
| parameters: | ABOVe:<br>BELow:<br>IMMediately:<br>Threshold Value: | Function starts when power is above the threshold value.<br>Function starts when power is below the threshold value.<br>Function starts immediately.<br>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 | |
| | Does NOT affect Keysight 8161x series return loss modules | |

| | | |
|---|---|---|
| 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:<br>BEL:<br>IMM:<br>Threshold Value: | Function starts when power is above the threshold value.<br>Function starts when power is below the threshold value.<br>Function starts immediately.<br>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 | |
| | Does NOT affect Keysight 8161x series return loss modules | |

| | |
|---|---|
| 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.<br>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. For a return loss module, sets the power range of the return loss diode. 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 mW | -80 dBm | 19.999 pW |
| -10 dBm | 199.99 mW | -90 dBm | 1.999 pW |
| -20 dBm | 19.999 mW | -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 and return loss modules. |
| 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. For a return loss module, returns the power range of the return loss diode. |
| parameters: | none |
| response: | The range setting as a *float* value in dBm<br>(-110 ≤ *value* ≤ +30). |
| example: | sens1:pow:rang? -> -2.00000000E+001<END> |
| affects: | All power meters and return loss modules. |
| dual sensors: | Master and slave channels are independent. |

| | |
|---|---|
| command: | **:SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:MONitor[:UPPer]** |
| syntax: | :SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:MONitor[:UPPer]<wsp><value>[DBM] |
| description: | Sets the power range for a retun loss module's monitor diode.<br>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 mW | -80 dBm | 19.999 pW |
| -10 dBm | 199.99 mW | -90 dBm | 1.999 pW |
| -20 dBm | 19.999 mW | -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:mon -20DBM |
| affects: | All return loss modules. |
| dual sensors: | Master and slave channels are independent. |

| | |
|---|---|
| command: | **:SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe:MONitor[:UPPer]?** |
| syntax: | :SENSe[*n*]:[CHANnel[*m*]]:POWer:RANGe[:UPPer]? |
| description: | Sets the power range for a retun loss module's monitor diode. |
| parameters: | none |
| response: | The range setting as a *float* value in dBm<br>(-110 ≤ *value* ≤ +30). |
| example: | sens1:pow:rang? -> -2.00000000E+001<END> |
| affects: | All return loss modules. |
| 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.<br>If automatic power ranging is enabled, ranging is automatically determined by the instrument. Otherwise, it must be set by the sensn:pow:rang command. Automatic ranging while other commands are sent to power meters has lead to timing conflicts in some configurations. Automation programs can often better control the range directly. | |
| parameters: | A *boolean* value: | 0 or OFF: automatic ranging disabled<br>1 or ON: automatic ranging enabled |
| response: | none | |
| example: | sens1:pow:rang:auto 1 | |
| affects: | All power meters and return loss modules | |
| | For return loss modules, affects return loss diode and monitor diode simultaneously. | |
| 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.<br>1: automatic ranging is being used. |

| | |
|---|---|
| example: | sens1:pow:rang:auto? -> 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: | **: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. | |
| | You must append a unit type<br>• dB if you use TOMODule or<br>• Watts or dBm if you use TOREF. | |
| | 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 114, 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<br>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<br>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><br><slot number>\|255\|TOREF,<channel number> | |
| description: | Selects the reference for the module. | |
| parameters: | slot number:<br>255 or TOREF:<br>channel number: | an integer value representing the slot number you want to reference<br>results are displayed relative to an absolute reference<br>an integer value representing the channel number you want to reference |
| | 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.<br>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<br>sens1:pow:ref:stat:rat TOREF,1 | References channel 2.1<br>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><br><br>sens1:pow:ref:stat:rat? -> +2,+1<END> | results are displayed relative to an absolute reference<br>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<br>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.<br>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. Frequent use of this command can conflict with the timing of autoranging in some configurations. Autorange can be disabled before and enabled after the command if needed. |
| parameters: | The wavelength as a *float* value in meters. |

| | | |
|---|---|---|
| | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>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 | |
| 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<br>MAX: maximum programmable value<br>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 | |
| 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: | For all sources, overwrites the current calibration values with the factory-set calibration settings. See :SENSe[n]:[CHANnel[m]]:RETurnloss:CALibration:COLLect:TERMination on page 118 and :SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[l]? on page 121 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: | For the currently selected source, start the calibration and save the calibration values for a defined reflectance reference measurement. See :SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[l] on page 121 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: | For the currently selected source, start the calibration and save the calibration values for a defined termination reference measurement. See :SENSe[n]:[CHANnel[m]]:RETurnloss:CORRection:REFLectance[l] on page 121 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:CALibration:TERMination?** |
| syntax: | :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:TERMination? |
| description: | Queries the T-value (termination calibration value) for the return loss module |
| parameters: | none |
| response: | Termination calibration value as a float in dB |
| example | sens1:ret:cal:term? -> +6.5000E+001 |
| affects: | All return loss modules |

| | |
|---|---|
| command: | **:SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:VALues?** |
| syntax: | :SENSe[*n*]:[CHANnel[*m*]]:RETurnloss:CALibration:VALues? |
| description: | Returns the the current calibration values<br>1. monitor diode reference power<br>2. return loss diode reference power<br>3. monitor diode parasitics power<br>4. return loss diode parasitics power. |
| parameters: | Returns power values in W |
| response: | none |
| example | sens1:ret:cal:val |
| 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. |
| | 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.<br>An external laser source is denoted by *0*. *0* is the default value of [*l*].<br>A lower wavelength source is denoted by *1*.<br>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: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. |
| | 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.<br>An external laser source is denoted by *0*. *0* is the default value.<br>A lower wavelength source is denoted by *1*.<br>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: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.<br>For example, the Keysight 81000BR reference reflector provides an accurate and stable 0.18 dB reference. |
| | 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.<br>An external laser source is denoted by *0*. *0* is the default value of [*l*].<br>A lower wavelength source is denoted by *1*.<br>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: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.<br>For example, the Keysight 81000BR reference reflector provides an accurate and stable 0.18 dB reference. |
| | 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.<br>An external laser source is denoted by *0*. *0* is the default value of [*l*].<br>A lower wavelength source is denoted by *1*.<br>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: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.<br>VPP: Output Voltage is proportional to optical power. |
| response: | none |
| example: | outp0:conn mod |
| affects: | Tunable laser modules with BNC ouput connector, except 81602A, 81606A, 81607A, 81608A, 81609A. |

| | |
|---|---|
| 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.<br>VPP: Output Voltage is proportional to optical power. |
| example: | outp0:conn? -> MOD<END> |
| affects: | All tunable laser modules with BNC ouput connector |

| | |
|---|---|
| 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.<br>LOWSse: The Low SSE output is regulated.<br>BHRegulated: Both outputs are active but only the High Power output is Regulated.<br>BLRegulated: Both outputs are active but only the Low SSE output is Regulated. |

| | |
|---|---|
| response: | none |
| example: | output0:path high |
| affects: | Tunable laser modules with two outputs. |

| | |
|---|---|
| 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.<br>LOWS: The Low SSE output is regulated.<br>BHR: Both outputs are active but only the High Power output is Regulated.<br>BLR: Both outputs are active but only the Low SSE output is Regulated. |
| example: | output0:path? -> HIGH<END> |
| affects: | Tunable laser modules with two outputs. |

| | |
|---|---|
| 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.<br>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.<br>For attenuator output see :OUTPutn[:CHANnel[m]]:[STATe] on page 193 |
| parameters: | 0 or OFF: switch laser current off<br>1 or ON: switch laser current on |
| response: | none |
| example: | outp 1 |
| affects: | All laser sources, DFB sources, tunable laser modules and return loss modules with an internal source |

| | |
|---|---|
| command: | **:OUTPut[*n*][:CHANnel[*m*]][:STATe]?** |
| syntax: | :OUTPut[*n*][:CHANnel[*m*]][:STATe]? |
| description: | Queries the current state of the laser current.<br>For attenuator output see :OUTPutn[:CHANnel[m]]:[STATe]? on page 193 |
| parameters: | none |
| response: | A *boolean* value: 0 – laser current off<br>1 – laser current on |
| example: | outp? -> 1<END> |
| affects: | All laser sources, DFB sources, 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<br>MAX: maximum programmable value<br>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.<br>The resolution of the frequency is always 1 Hz. |
| | 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 sources, DFB sources, and tunable laser modules except 81950A. |

| | |
|---|---|
| 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<br>MAX: maximum modulation frequency<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum modulation frequency and the maximum modulation frequency. |
| | 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 is chosen as a parameter). |
| example: | sour2:am:freq? min -> +2.00000000E+002<END> |
| affects: | All laser sources, DFB sources, and tunable laser modules except 81950A. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:AM:SOURce[*l*]** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:AM:SOURce[*l*]<wsp><br>INT\|INT1\|INT2\|COHC\|AEXT\|EXT\|DEXT\|WVLL\|BACK\|0\|1\|2\|3\|5\|6 | |
| description: | Selects the type or source of the modulation of the laser output. | |
| parameters: | 0, INT1, or INTernal:<br>1, COHCtrl, or INT2:<br>2, AEXTernal, or EXT:<br>3 or DEXTernal:<br>4 or LFCohctrl:<br>5 or WVLLocking:<br>6 or BACKplane: | internal digital modulation<br>coherence control<br>external analog modulation<br>external digital modulation<br>low frequency coherence control<br>wavelength locking<br>external digital modulation using Input Trigger Connector |
| | 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: | Internal digital modulation is available with fixed-wavelength laser sources (Fabry-Perot, DFB lasers), return loss modules containing an internal source, and tunable lasers, except 81602A, 81606A, 81607A, 81608A and 81609A.<br>Coherence control is available with DFB sources and tunable lasers except 81606A and 81607A. External wavelength locking is available with 81602A, 81606A and 81600B tunable lasers.<br>Other modulation modes are only available with tunable laser modules except 81602A, 81606A, 81607A, 81608A and 81609A. | |

| | |
|---|---|
| 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: | none |
| | 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: | 0: | internal digital modulation |
| | 1: | coherence control |
| | 2: | external analog modulation |
| | 3: | external digital modulation |
| | 5: | wavelength locking |
| | 6: | external digital modulation using Input Trigger Connector |
| example: | sour2:am:sour? -> +0<END> | |
| affects: | Internal digital modulation is available with fixed-wavelength laser sources (Fabry-Perot, DFB lasers), return loss modules containing an internal source, and tunable lasers, except 81602A, 81606A, 81607A, 81608A and 81609A.<br>Coherence control is available with DFB sources and tunable lasers except 81606A and 81607A. External wavelength locking is available with 81602A, 81606A and 81600B tunable lasers.<br>Other modulation modes are only available with tunable laser modules except 81602A, 81606A, 81607A, 81608A and 81609A. | |

| | | |
|---|---|---|
| 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: amplitude modulation disabled (default)<br>ON or 1: amplitude modulation enabled. |
| | 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*. | |
| | 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.<br>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 139). | |
| | When you enable lambda logging, see [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:LLOGging on page 170, and modulation simultaneously, a sweep cannot be started, see [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:[STATe] on page 176. | |
| response: | none | |
| example: | sour2:am:stat 0 | |
| affects: | All laser sources, DFB sources, tunable laser modules except 81950A, and return loss modules containing an internal source. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:AM:STATe[*l*]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:AM:STATe[*l*]? | |
| description: | Returns the current state of amplitude modulation. | |
| | 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<br>1: modulation is enabled |
| example: | sour2:am:stat? -> 0<END> | |
| affects: | All laser sources, DFB sources, tunable laser modules except 81950A, and return loss modules containing an internal source. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:AM:COHCtrl:COHLevel[*l*]** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:AM:COHCtrl:COHLevel[*l*]<wsp><value>[MIN \| MAX \| DEF] | |
| description: | Sets the level of coherence, when using coherence control, on an arbitrary scale from 1 to 99.98%. A 100% coherence level corresponds to maximum coherence length and minimum linewidth. The coherence level required for a specific linewidth and coherence length can vary between modules. | |
| parameters: | The excursion level as a percentage of its maximum value. | |
| | Also allowed: | MIN: minimum programmable value (0%)<br>MAX: maximum programmable value (100%)<br>DEF: default preset (*RST) value. |
| response: | none | |
| example: | source2:am:cohc 50 | |
| affects: | DFB sources and Keysight 81960A, 81980A, 81940A, 81989A, 81949A | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:AM:COHCtrl:COHLevel[*l*]?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:AM:COHCtrl:COHLevel[*l*]<wsp><value>?[MIN \| MAX \| DEF] |
| description: | Queries the current level of coherence, when using Coherence Control. Coherence is expressed on an arbitrary scale from 1 to 99.98%. A 100% coherence level corresponds to maximum coherence length and minimum linewidth. |

| | | |
|---|---|---|
| parameters: | Optional | MIN: returns the minimum programmable value (0%)<br>MAX: returns the maximum programmable value (100%)<br>DEF: returns the default preset (*RST) value. |
| response: | Returns the currently set excursion level as a percentage between 0 and 99.98 | |
| example: | source2:am:cohc? -> 50<END> | |
| affects: | DFB sources and Keysight 81960A, 81980A, 81940A, 81989A, 81949A | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:SOURce[*l*]** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:SOURce[*l*]<wsp>SBSCtrl\|0 |
| description: | Selects the type of the frequency modulation of the laser output.<br>Currently, only parameter strings that select SBS Control are valid.<br>Enable frequency modulation before issuing this command.<br>Refer to [:SOURce[n]][:CHANnel[m]]:FM:STATe[l] on page 130. |
| parameters: | 0, SBSCtrl — Stimulated Brillouin Scattering | |
| | SBSCtrl (Stimulated Brillouin Scattering) Control modulation suppresses SBS effects within high-power measurement setups. | |
| response: | none | |
| example: | sour2:fm:sour SBSC | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:SOURce[*l*]?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:SOURce[*l*]? |
| description: | Queries the type of frequency modulation currently set.<br>Currently, only SBS Control is available. |
| parameters: | none |
| response: | 0 — SBS Control |
| example: | sour2:fm:sour? -> +0<END> |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:STATe[*l*]** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:STATe[*l*]<wsp>OFF\|ON\|0\|1 | |
| description: | Enables and disables frequency modulation of the laser output. | |
| parameters: | A *boolean* value: | OFF or 0: disable frequency modulation<br>ON or 1: enable frequency modulation. |
| response: | none | |
| example: | sour2:fm:state 1 | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:STATe[*l*]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:STATe[*l*]? | |
| description: | Queries the current state of frequency modulation of the laser output. | |
| parameters: | none | |
| response: | A *boolean* value: | 0: frequency modulation is disabled<br>1: frequency modulation is enabled. |
| example: | sour2:fm:state? -> +1<END> | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:FREQuency[*l*]** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:FREQuency[*l*]<wsp><frequency>[MHZ\|KHZ\|HZ\|MIN\|MAX\|DEF] | |
| description: | Sets the frequency of the SBS Control modulation.<br>Enable frequency modulation before issuing this command.<br>Refer to [:SOURce[n]][:CHANnel[m]]:FM:SOURce[l] on page 129 and [:SOURce[n]][:CHANnel[m]]:FM:STATe[l] on page 130. | |
| parameters: | The modulation frequency as a *float* value.<br>The default units are HZ, although KHZ, MHZ, GHZ and THZ can also be specified. | |
| | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>DEF: default preset (*RST) value. |
| response: | none | |
| example: | sour2:fm:sbsc:freq 4000Hz | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:FREQuency[*l*]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:FREQuency[*l*]?<wsp>[MIN\|MAX\|DEF] | |
| description: | Queries the currently set frequency of the SBS Control modulation. | |
| parameters: | Optional | MIN: returns the minimum programmable value<br>MAX: returns the maximum programmable value<br>DEF: returns the default preset (*RST) value. |
| response: | The modulation frequency in Hz as a *float* value | |
| example: | sour2:fm:freq? -> +4.00000E+03<END> | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:LEVel[*l*]** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:LEVel[*l*]<wsp>[MIN\|MAX\|DEF] |
| description: | Sets the excursion of the SBS Control frequency modulation to a percentage of its maximum value. |

| | | |
|---|---|---|
| parameters: | The excursion level as a percentage of its maximum value. | |
| | Also allowed: | MIN: minimum programmable value (0%)<br>MAX: maximum programmable value (100%)<br>DEF: default preset (*RST) value. |
| response: | none | |
| example: | sour2:fm:sbsc:lev 80 | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:Level[*l*]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:FM:SBSCtrl:LEVel[*l*]?<wsp>[MIN\|MAX\|DEF] | |
| description: | Queries the currently set excursion level of the SBS Control frequency modulation. | |
| parameters: | Optional | MIN: returns the minimum programmable value (0%)<br>MAX: returns the maximum programmable value (100%)<br>DEF: returns the default preset (*RST) value. |
| response: | Returns the currently set excursion level as a percentage of its maximum value. | |
| example: | sour2:fm:sbsc:lev? -> +8.000E+01<END> | |
| affects: | Keysight 81960A, 81980A, 81940A, 81989A, 81949A compact tunable lasers. | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency<wsp><br><value>[HZ\|KHZ\|MHZ\|GHZ\|THZ]\|MIN\|MAX\|DEF |
| description: | Sets the laser frequency. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:freq 188THz |
| affects: | 81950A |
| Notes | Only available in "Auto<br>Mode". |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency?<wsp>[MIN\|MAX\|DEF] |
| description: | Returns the frequency value in Hz. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current frequency value or the minimum, maximum, or default frequency value. |
| example: | sour1:freq? +1.88000000E+014 |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:REFerence** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:REFerence<wsp><br><value>[HZ\|KHZ\|MHZ\|GHZ\|THZ]\|MIN\|MAX\|DEF |
| description: | Set the reference frequency for the frequency grid used by the instrument. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:freq:ref 193.1THz |
| affects: | 81950A |
| Notes | Only available in "Grid Mode" |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:REFerence?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:REFerence?<wsp>[MIN\|MAX\|DEF] |
| description: | Returns the reference frequency for the frequency grid used by the instrument. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current reference frequency value or the minimum, maximum, or default reference frequency value in Hz. |
| example: | sour1:freq:ref? +1.93100000E+014 |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:GRID** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:GRID<wsp><br><value>[HZ\|KHZ\|MHZ\|GHZ\|THZ]\|MIN\|MAX\|DEF |
| description: | Set the grid spacing for the frequency grid used by the instrument. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:freq:grid 50e9 |
| affects: | 81950A |
| Notes | Only available in "Grid Mode" |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:GRID?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:GRID?<wsp>[MIN\|MAX\|DEF] |
| description: | Returns the grid spacing for the frequency grid used by the instrument. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current grid spacing value or the minimum, maximum, or default grid spacing value in Hz. |
| example: | sour1:freq:grid? +5.00000000E+010 |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:CHANnel** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:CHANnel<wsp><value>\|MIN\|MAX\|DEF |
| description: | Sets the laser's grid channel. |
| parameters: | Any value resulting in a valid frequency, refer to the chapter “Output Frequency” on page 33.<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:freq:chan -20 |
| affects: | 81950A |
| Notes | Only available in "Grid Mode".<br>The minimum, maximum, and default value depend on the current reference frequency and grid spacing values. |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:CHANnel?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:CHANnel?<wsp>[MIN\|MAX\|DEF] |
| description: | Returns the laser's grid channel. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current grid channel value or the minimum, maximum, or default grid channel value. |
| example: | sour1:freq:chan? -20 |
| affects: | 81950A |
| Notes | The minimum, maximum, and default value depend on the current reference frequency and grid spacing values. |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:OFFSet** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:OFFset<wsp><br><value>[HZ\|KHZ\|MHZ\|GHZ\|THZ]\|MIN\|MAX\|DEF |
| description: | Set the frequency offset used by the instrument. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:freq:offs 0.1e9 |
| affects: | 81950A |
| Notes | Only available in "Grid Mode" |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:OFFSet?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:OFFSet?<wsp>[MIN\|MAX\|DEF] |
| description: | Returns the frequency offset used by the instrument in Hz. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current frequency offset value or the minimum, maximum, or default frequency offset value in Hz. |
| example: | sour1:freq:offs? +1.00000000E+008 |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:TOGRid** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:TOGRid <wsp><br><value>[HZ\|KHZ\|MHZ\|GHZ\|THZ] |
| description: | Set the frequency to the nearest grid point only changing the channel number. |
| parameters: | The frequency value. |
| response: | None |
| example: | sour4:freq:togr 188.5thz |
| affects: | 81950A |
| Notes | Only available in "Grid Mode"<br>The frequency offset is neither changed nor taken into account when calculating the nearest channel. |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:AUTO** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:AUTO<wsp>OFF\|0\|ON\|1 |
| description: | Switch between "Auto Mode" and "Grid Mode" |
| parameters: | OFF or 0: "Grid Mode" (= Auto Mode off)<br>ON or 1: "Auto Mode" |
| response: | None |
| example: | sour1:freq:auto on |
| affects: | 81950A |
| Notes | This command is equivalent to [:SOURce[n][:CHANnel[m]]:WAVelength:AUTO |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:FREQuency:AUTO?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:FREQuency:AUTO? |
| description: | Returns the current operation mode. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | 0: "Grid Mode" (= Auto Mode off)<br>1: "Auto Mode" |
| example: | sour1:freq:auto? 0 |
| affects: | 81950A |
| Notes | This query is equivalent to [:SOURce[n][:CHANnel[m]]:WAVelength:AUTO? |

| | | |
|---|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODout** | |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODout<wsp>FRQ\|FRQRDY\|0\|1 | |
| description: | Sets the modulation output mode of the BNC connector on the front panel of tunable laser modules. | |
| parameters: | FRQ or 0:<br>FRQRDY or 1: | modulation signal is output all the time<br>modulation is combined with the laser-ready signal.<br>In this case, the output is kept low when no optical signal is output (for example, while the laser<br>is settling after a change of wavelength). |
| response: | none | |
| example: | sour0:mod 0 | |
| affects: | Tunable laser sources with BNC output connector, except 81602A, 81606A, 81607A, 81608A and 81609A. | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:MODout?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:MODout? |
| description: | Queries the modulation output mode of the BNC connector on the front panel of tunable laser modules. |
| parameters: | none |
| response: | 0 : modulation signal is output all the time<br>modulation is combined with the laser-ready signal.<br>1 : 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: | sour0:mod? -> 0<END> |
| affects: | All tunable laser sources with BNC output connector. |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal[:STATe]** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal[:STATe]<wsp>OFF\|0\|ON\|1 |
| description: | Switches the internal modulation (SBS suppression) on or off. |
| parameters: | OFF or 0: Disable the modulation<br>ON or 1: Enable the modulation |
| response: | None |
| example: | sour1:modu:int on |
| affects: | 81950A |
| Notes | SBSControl must be set to an appropriate value, too. |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal[:STATe]?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal[:STATe]? |
| description: | Returns the internal modulation state. |
| parameters: | none |
| response: | 0 Off<br>:<br>1 On<br>: |
| example: | sour1:modu:int? 0 |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal:SBSControl[:LEVel]** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal:SBSControl[:LEVel] <wsp><br><value>[HZ\|KHZ\|MHZ\|GHZ\|THZ]\|MIN\|MAX\|DEF |
| description: | Sets the internal frequency modulation frequency. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:modu:int:sbsc 0.2GHz |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal:SBSControl[:LEVel]?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:INTernal:SBSControl[:LEVel]?<wsp>[MIN\|MAX\|DEF] |
| description: | Returns the internal frequency modulation frequency in Hz. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current modulation frequency value or the minimum, maximum, or default modulation frequency value in Hz. |
| example: | sour1:modu:int:sbsc? +2.00000000E+008 |
| affects: | 81950A |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal[:STATe]** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal[:STATe]<wsp>OFF\|0\|ON\|1 |
| description: | Sets the state of the external amplitude modulation. |
| parameters: | OFF or 0: off<br>ON or 1: on |
| response: | None |
| example: | sour1:modu:ext on |
| affects: | 81950A only |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal[:STATe]?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal[:STATe]? |
| description: | Returns the state of the external amplitude modulation. |
| parameters: | none |
| response: | 0 on<br>:<br>1 off<br>: |
| example: | sour1:modu:ext? 1 |
| affects: | 81950A only |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal:AM[:LEVel]** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal:AM[:LEVel] <wsp><br><value>\|MIN\|MAX\|DEF |
| description: | Sets the amplitude modulation level in %. |
| parameters: | Any value in the specified range (see appropriate User's Guide)<br>Also allowed:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | None |
| example: | sour1:modu:ext:am 2.0 |
| affects: | 81950A only |
| Notes | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal:AM[:LEVel]?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:MODUlation:EXTernal:AM[:LEVel]?<wsp><br>[MIN\|MAX\|DEF] |
| description: | Returns the amplitude modulation level in %. |
| parameters: | Optional:<br>MIN - minimum programmable value<br>MAX - maximum programmable value<br>DEF - default value |
| response: | The current amplitude modulation level or the minimum, maximum, or default amplitude modulation level. |
| example: | sour1:modu:ext:am? +2.00000000E+000 |
| affects: | 81950A only |
| Notes | |

| | | |
|---|---|---|
| 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<br>MAX: maximum programmable value<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable value and the maximum programmable value |
| | 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*. | |
| | Tunable laser modules with in-built optical attenuators need to be in Manual Attenuation Mode (see [:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation[l]:AUTO on page 146) 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[l]] on page 148). | |
| | 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: | sour0:pow:att 22.32db | |
| affects: | All tunable laser modules with an built-in optical attenuator, and all laser source modules. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]?[MIN\|DEF\|MAX] | |
| description: | Queries the attenuation level.<br>When using a tunable laser module with a built-in optical attenuator, the value returned applies only to the attenuation mode (see [:SOURce[n]][:CHANnel[m]]:POWer:ATTenuation[l]:AUTO on page 146). | |
| | 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<br>MAX: maximum amplitude level<br>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: | sour0:pow:att? def -> +3.10000000+E001<END> |
| affects: | All tunable laser modules with an in-built optical attenuator, and all laser source modules. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:AUTO** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:AUTO<wsp>OFF\|ON\|0\|1 |
| description: | Selects Automatic or Manual Attenuation Mode.<br>In Automatic Attenuation Mode, you specify the output power.<br>In Manual Attenuation Mode, you must specify both the laser output power, and the attenuation level. |
| parameters: | OFF or 0: Attenuation Mode<br>ON or 1: Power Mode |
| response: | none |
| example: | sour0:pow:att:auto 1 |
| affects: | All tunable laser sources with a built-in optical attenuator. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:AUTO?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:AUTO? |
| description: | Queries whether the instrument is in Automatic or Manual Attenuation Mode. |
| parameters: | none |
| response: | 0: Manual Attenuation Mode<br>1: Automatic Attenuation Mode |
| example: | sour0:pow:att:auto? -> 1<END> |
| affects: | All tunable laser modules with a built-in optical attenuator. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:DARK** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:DARK<wsp>OFF\|ON\|0\|1 | |
| description: | Sets or unsets the attenuator to 'dark' position.<br>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.<br>This command is available in Attenuation Mode Only. | |
| parameters: | OFF or 0:<br>ON or 1: | Unsets dark position<br>Sets dark position |
| response: | none | |
| example: | sour0:pow:att:dark 1 | |
| affects: | All tunable laser modules with a built-in optical attenuator. | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]:DARK?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer:ATTenuation[*l*]: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<br>1: dark position set |
| example: | sour0:pow:att:dark? -> 1<END> |
| affects: | All tunable laser modules with a built-in optical attenuator. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel][:IMMediate][:AMPLitude[*l*]]** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel][:IMMediate][:AMPLitude[*l*]]<wsp><value> [PW\|NW\|UW\|MW\|Watt\|DBM] |
| description: | Sets the power of the laser output. |
| | 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[l]:AUTO on page 146).<br>If you are using power mode, the value returned is the output power.<br>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 145).<br>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. |
| | 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[l]]? on page 149 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 151. |
| parameters: | Any value in the specified range (see the appropriate *User's Guide*). |
| | Also allowed are: MIN: minimum programmable value<br>MAX: maximum programmable value<br>DEF: This is not the preset (*RST) default value, but is the maximum programmable level |
| | Use [*l*] to set 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: | none |
| example: | sour2:pow 23uW |
| affects: | All tunable laser including 81950A 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.<br>The value returned is the actual amplitude that is output, which may be different from the value set for the output. If these two figures are not the same, it is indicated in the :STATus:OPERation register. |
| | 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[l]:AUTO on page 146).<br>If you are using power mode, the value returned is the output power.<br>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 145).<br>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:<br>MIN: minimum amplitude level<br>MAX: maximum amplitude level<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum amplitude level and the maximum amplitude level |
| | 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 sources, DFB sources, and tunable laser modules including 81950A and return loss modules with an internal source |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime[*l*]** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime[*l*]<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<br>MAX: maximum programmable value<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable level and the maximum programmable level |
| | Use [*l*] to set the risetime 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:pow:ris 10ns | |
| affects: | Fixed-wavelength laser sources (Fabry-Perot, DFB lasers), return loss modules containing an internal source, and tunable lasers, except 81602A, 81606A, 81607A, 81608A and 81609A. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime[*l*]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:POWer[:LEVel]:RISetime[*l*]?<wsp>[MIN\|DEF\|MAX] | |
| description: | Returns the laser rise time of the chosen source. | |
| parameters: | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable level and the maximum programmable level |
| | Use [*l*] to query the risetime 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: | The rise time as a float value in seconds. | |
| example: | sour2:pow:ris? -> +1.00000000E-009<END> | |
| affects: | All laser sources, DFB sources, 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<wsp><boolean> |
| description: | Switches the laser of the chosen source on or off. |
| parameters: | A *boolean* value: 0: Laser Off<br>1: Laser On |
| response: | none |
| example: | sour2:pow:stat 1 |
| affects: | All laser source, DFB source, and tunable laser modules including 81950A 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<br>1: Laser On |
| example: | sour2:pow:stat? -> 1<END> |
| affects: | All laser source, DFB source, and tunable laser modules including 81950A 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)<br>1 or W: Watts |
| response: | none |
| example: | sour2:pow:unit w |
| affects: | All tunable laser including 81950A 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<br>1: Watts |
| response: | none |
| example: | sour2:pow:unit? -> +0<END> |
| affects: | All tunable laser including 81950A and DFB source modules |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:POWer:WAVelength** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]:POWer:WAVelength[<wsp><br>EXTernal\|LOWer\|UPPer\|BOTH\|0\|1\|2\|3] | |
| | For compatibility reasons, WAVelength may be replaced with WAVE. | |
| description: | Sets the wavelength source for a dual-wavelength laser source. | |
| parameters: | EXTernal: or 0<br>LOWer: or 1<br>UPPer: or 2<br>BOTH: or 3 | External<br>The lower wavelength source<br>The upper wavelength source<br>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? | |
| | For compatibility reasons, WAVelength may be replaced with WAVE. | |
| description: | Returns the wavelength source for a dual-wavelength laser source. | |
| parameters: | none | |
| response: | LOW<br>UPP<br>BOTH | The lower wavelength source<br>The upper wavelength source<br>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 170. 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:DATA:BLOCk?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]:READout:DATA:BLOCk?<wsp>LLOGging\|PMAX,<offset>,<# of data points> | |
| description: | Returns a specified binary block from either a lambda logging operation, or maximum power at wavelength characteristic. | |
| parameters: | LLOGging: | Returns the data block from lambda logging. The binary block is an 8-byte long double in Intel byte order. |
| | PMAX: | Returns the data block from the power curve characteristic. 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. |
| | <offset> | A zero based offset that specifies the index of the first value within the block to be transferred. |
| | <# of data points> | The number of points (not bytes!) in the transferred block. |
| response: | A binary stream in Intel byte order. | |
| example: | sour0:read:data:block? llog,100,20000 -> *the data as a binary stream* | |
| affects: | All tunable laser and DFB source modules | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:READout:DATA:MAXBlocksize?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]:READout:DATA:MAXBlocksize? |
| description: | Returns the maximum block size for a single GPIB transfer for lambda logging functions. If your application requires more data points please use SOURce[n]][:CHANnel[m]]:READout:DATA:BLOCk? instead of SOURce[n]][:CHANnel[m]]:READout:DATA? |

| | |
|---|---|
| parameters: | none |
| response: | The maximum number of data points (not bytes!) in the transferred block, as an *integer* value. |
| example: | sour0:read:data:maxb? -> 120<END> |
| 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 <$paraext on page 170. |
| | 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[*l*]:FIXED[*l*]]** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength[:CW[*l*]:FIXED[*l*]]<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*).<br>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<br>MAX: maximum wavelength value<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum wavelength value and the maximum wavelength value |
| | Use [*l*] to set 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*. For 81950A, only available in "Auto Mode". "CW" and "Fixed" are just present for backwards compatibility. | |
| response: | none | |
| example: | sour2:wav 1550NM | |
| affects: | All tunable laser including 81950A 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. | |
| parameters: | none | |
| | Also allowed, for tunable laser modules only, are | MIN: minimum wavelength<br>MAX: maximum wavelength<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum wavelength value and the maximum wavelength value |
| | 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*. | |
| 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 including 81950A and return loss modules with an internal source | |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:WAVelength:TOGRid** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:WAVelength:TOGRid<wsp><br><value>[PM\|NM\|UM\|MM\|M] |
| description: | Set the wavelength to the nearest grid point only changing the channel number. |
| parameters: | The wavelength value. |
| response: | none |
| example: | sour4:wav:togr 1.6um |
| affects: | 81950A |
| Notes | Only available in "Grid Mode"<br>The frequency offset is neither changed nor taken into account when calculating the nearest channel. |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:WAVelength:AUTO** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:WAVelength:AUTO<wsp>OFF\|0\|ON\|1 |
| description: | Switch between "Auto Mode" and "Grid Mode" |
| parameters: | OFF or 0: "Grid Mode" (= Auto Mode off)<br>ON or 1: "Auto Mode" |
| response: | none |
| example: | sour1:wav:auto on |
| affects: | 81950A |
| Notes | This command is equivalent to [:SOURce[n][:CHANnel[m]]:FREQuency:AUTO |

| | |
|---|---|
| command: | **[:SOURce[n]][:CHANnel[m]]:WAVelength:AUTO?** |
| syntax: | [:SOURce[n]][:CHANnel[m]]:WAVelength:AUTO? |
| description: | Returns the current operation mode. |
| parameters: | none |
| response: | 0: "Grid Mode" (= Auto Mode off)<br>1: "Auto Mode" |
| example: | sour1:wav:auto → 0 |
| affects: | 81950A |
| Notes | This query is equivalent to [:SOURce[n][:CHANnel[m]]:FREQuency:AUTO? |

| | |
|---|---|
| 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: | sour0:wav:corr:ara |
| affects: | All tunable laser modules except 81649A, 81689A/B, and 81950A |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ARA:ALL** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ARA:ALL |
| description: | Realigns the laser cavity of every tunable laser source in a mainframe. |
| parameters: | none |
| response: | none |
| example: | sour0:wav:corr:ara:all |
| affects: | All tunable laser modules except 81649A, 81689A/B, and 81950A |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:AUTocalib** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:AUTocalib<wsp>ON (1) \| OFF (0) |
| description: | Sets the Auto Calibration feature On or OFF. Switching it OFF enables the TLS to operate for a long period without interruption from the "auto lambda zeroing" or settling. When Auto Calibration is disabled, it is possible to operate the TLS at a temperature that differs more than 4.4 K from the last Lambda Zeroing temperature. In this case, the accuracy and wavelength performance of the TLS can become less optimal due to temperature variation. The relevent accuracy class is indicated on the user interface when Auto Calibration is off. |
| parameters: | a boolean value: 1 or ON: enable Autocalibration<br>0 or OFF: disable Autocalibration |
| response: | none |
| example: | sour0:wav:corr:aut 0 |
| affects: | All tunable laser modules except 81649A, 81689A/B and 81980A, 81940A, 81989A, 81949A, 81602A, 81606A, 81607A, 81608A, 81609A, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:AUTocalib?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:AUTocalib? |
| description: | Returns whether Autocalibration is enabled or disabled |
| parameters: | none |

| | |
|---|---|
| response: | 0 Autocalibration disabled<br>1 Autocalibration enabled |
| example: | sour0:wav:corr:aut? -> 1 |
| affects: | All tunable laser modules except 81649A, 81689A/B and 81980A, 81940A, 81989A, 81949A, 81602A, 81606A, 81607A, 81608A, 81609A, and 81950A. |

| | |
|---|---|
| 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 81649A, 81689A/B, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:ALL** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:ALL |
| description: | Executes a wavelength zero on every tunable laser source in a mainframe. |
| parameters: | none |
| response: | none |
| example: | sour2:wav:corr:zero:all |
| affects: | All tunable laser modules except 81649A, 81689A/B, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:TEMPerature:ACTual?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:TEMPerature:ACTual? |
| description: | Reports the current lambda zero tempearture. |
| parameters: | none |
| response: | float value; temperature in °C |
| example: | sour0:wav:corr:zero:temp:act? |
| affects: | All tunable laser modules except 81649A, 81689A/B and 819xxA/B, 81602A, 81606A, 81607A, 81608A, 81609A, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:TEMPerature:DIFFerence?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:TEMPerature:Difference? |
| description: | Reports the temperature difference required to trigger an auto lamda zero. |
| parameters: | none |
| response: | **float** value; temperature in °C |
| example: | sour0:wav:corr:zero:temp:diff? |
| affects: | All tunable laser modules except 81649A, 81689A/B and 819xxA/B, 81602A, 81606A, 81607A, 81608A, 81609A, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:TEMPerature:LASTzero?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:TEMPerature:LASTzero? |
| description: | Reports the temperature at which the last auto lamda zero took place. |
| parameters: | none |
| response: | float value; temperature in °C |
| example: | sour0:wav:corr:zero:temp:last? |
| affects: | All tunable laser modules except 81649A, 81689A/B and 819xxA/B, 81602A, 81606A, 81607A, 81608A, 81609A, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:AUTO** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:CORRection:ZERO:AUTO |
| description: | Forces an auto lamda zero. This is quicker but a little less accurate than the equilavent manual process because some checks are omitted: |
| parameters: | none |
| response: | none |
| example: | sour0:wav:corr:zero:auto |
| affects: | All tunable laser modules except 81649A, 81689A/B and 819xxA/B, 81602A, 81606A, 81607A, 81608A, 81609A. and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency[*l*]** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency[*l*]<wsp><value><br>[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.<br>The default units for frequency are Hertz.<br>The output wavelength[$\lambda$] is set from the base wavelength ($\lambda_0$) and the frequency offset (df). The formula for calculating the output wavelength is: |
| | $$\lambda = \frac{(c)}{((\lambda_o df) + c)} \quad \lambda_o$$ |
| | where c is the speed of light in a vacuum (2.990 x $10^8$ $ms^{-1}$) |
| | Use [l] to set the 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*. |
| parameters: | The frequency difference is a **float** value in Hz. |
| response: | none |
| example: | sour0:wav:freq -10THZ |
| affects: | All tunable laser sources |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency[*l*]?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:FREQuency[*l*]? |
| description: | Returns the frequency difference used to calculate a relative wavelength. |
| | Use [*l*] to query the 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*. |
| parameters: | none |
| response: | Returns the frequency difference as a float value in Hz. |
| example: | wav:freq? -> -1.00000000E+013<END> |
| affects: | All tunable laser sources |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:REFerence[*l*]?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:REFerence[*l*]? |
| description: | Returns the reference wavelength ($\lambda_0$). |
| 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 ($\lambda$ -> $\lambda_0$), 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:CHECkparams?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CHECkparams? | |
| description: | Returns whether the currently set sweep parameters (sweep mode, sweep start, stop, width, etc.) are consistent. If there is a sweep configuration problem, the laser source is not able to pass a wavelength sweep. | |
| parameters: | none | |
| response: | A *string* with a detailed description of a configuration problem, or "OK" if the sweep os configured correctly. The responses shown below are all the possible configuration problem strings: | |
| | Message | Description |
| | 368,LambdaStop <=LambdaStart | start wavelength must be smaller than stop wavelength |
| | 369,sweepTime < min | the total time of the sweep is too small |
| | 370,sweepTime > max | the total time of the sweep is too large |
| | 371,triggerFreq > max | the trigger frequency (calculated from sweep speed divided by sweep step) is too large |
| | 372,step < min | step size too small |
| | 373,triggerNum > max | the number of triggers exceeds the allowed limit |
| | 374,LambdaLogging = On AND Modulation = On AND ModulationSource! = CoherenceControl | The only allowed modulation source with the lambda logging function is coherence control. |
| | 375,LambdaLogging = On AND TriggerOut! = StepFinished | lambda logging only works "Step Finished" output trigger configuration |
| | 376,Lambda logging in stepped mode | lambda logging can only be done in continuous sweep mode |
| | 377,step not multiple of <x> | the step size must be a multiple of the smallest possible step size |
| | 378, triggerFreq < min | the number of triggers exceeds the allowed limit |
| | 379, Continuous Sweep AND Modulation = On | 379,Continuous Sweep = On for 81602A, 81606A, 81607A, 81608A, 81609A |
| example: | sour0:wav:swe:chec? -> "triggerNum > max" | |
| affects: | All tunable laser modules except Keysight 81649A, Keysight 81689A/B, and 81950A. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes** | |
| syntax: | [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:CYCLes<wsp><br><value>\|MIN\|MAX\|DEF\|0 | |
| description: | Sets the number of cycles. | |
| | Cannot be set while a sweep is running. | |
| parameters: | The number of cycles is an integer value. | |
| | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable value and the maximum programmable value<br>0: cycles continuously. |
| response: | none | |
| example: | wav:swe:cycl 3 | |
| affects: | All tunable laser modules, except 81950A. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:CYCLes?<br>[<wsp>MIN\|MAX\|DEF] | |
| description: | Returns the number of cycles. | |
| parameters: | none | |
| | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>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 a signed integer value. | |
| example: | wav:swe:cycl? -> +3<END> | |
| affects: | All tunable laser modules, except 81950A. | |

| | | |
|---|---|---|
| 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. Can only be used when sweep is stepped. | |
| | Cannot be set while a sweep is running. | |
| parameters: | The dwell time as a *float* value.<br>If you specify no units in your command, seconds are used as the default. | |
| | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>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, except 81950A. | |

| | | |
|---|---|---|
| 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. Can only be used when sweep is stepped. | |
| parameters: | none | |
| | Also allowed are: | MIN: minimum programmable value<br>MAX: maximum programmable value<br>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, except 81950A. | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:EXP?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:EXP? |
| description: | Returns the number of triggers. A tunable laser wavelength sweep causes a number of triggers, this number is required to configure a triggering data acquisition function on a power meter. The number returned by this function can be used to configure a Power Meter for coordinated measurements with a tunable laser source (see command :SENSe[n][:CHANnel[m]]:FUNCtion:PARameter:LOGGing on page 97). |
| parameters: | none |
| response: | the number of expected triggers as a signed integer value. |
| example: | sour0:wav:swe:exp? -> +12001 |
| affects: | All tunable laser modules except 81649A, 81689A/B, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:FLAG?** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:FLAG? |
| description: | The sweep flag is used to find out when logging data is available and when the next sweep cycle may be triggered.<br>It may also be used as a sweep cycle counter, where: flag/2 = number of sweep cycles<br>The flag is:<br>- only used in continuous sweep<br>- set to 0 at start/end of sweep<br>- incremented when the sweep is waiting for a trigger<br>- incremented when logging data is available<br>- an odd number when, waiting for a trigger<br>- an even number when, logging data may be read<br>If the trigger input isn't configured to start a sweep cycle the flag is increased by two when the logging data is available<br>If no logging data is calculated, because the user doesn't want lambda logging, the flag is incremented at the end of the sweep cycle regardless |

| | Sweep state | Flag |
|---|---|---|
| | start | 0 |
| | sweep waiting for trigger | 1 |
| | trigger ->`<br>`first cycle \| start moving back \| do some postprocessing \|<br>`logging data available | 2 |
| | sweep waiting for next trigger | 3 |
| | | ..... |
| | sweep stopped or finished | 0 |
| parameters: | none | |
| response: | the current sweep flag value as a signed integer value | |
| example: | sour0:wav:swe:flag? -> +30 | |
| affects: | All tunable laser modules except 81649A, 81689A/B, 81949A, 81989A, 81950A, and 81609A. | |

| | |
|---|---|
| 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. |
| | The following settings are the prerequisites for Lambda Logging:<br>Set [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:MODE on page 171 to CONTinuous.<br>Set :TRIGger[n][:CHANnel[m]]:OUTPut on page 213 to STFinished (step finished).<br>Set [:SOURce[n]][:CHANnel[m]]:AM:STATe[l] on page 127 to OFF.<br>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. |
| | Lambda logging is disabled at the end of a sweep. |
| | Generally, a continuous sweep can only be started if:<br>the trigger frequency, derived from the sweep speed and sweep step, is <= 40kHz<br>the number of triggers, calculated from the sweep span and sweep span, is <=100001<br>the start wavelength is less than the stop wavelength.<br>In addition, a continuous sweep with lambda logging requires:<br>the trigger output to be set to step finished<br>modulation set to coherence control or off. |
| parameters: | 0 or OFF: switch lambda logging off<br>1 or ON: switch lambda logging on |
| response: | none |
| example: | wav:swe:llog 1 |
| affects: | All tunable laser modules that support continuous sweep. |

| | |
|---|---|
| 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<br>1 – lambda logging is switched on |
| example: | wav:swe:llog? -> 1<END> |
| affects: | All tunable laser modules that support continuous sweeps. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:MODE** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:MODE<wsp><mode> | |
| description: | Sets the sweep mode. | |
| | Cannot be set while a sweep is running. | |
| parameters: | STEPped:<br>MANual:<br>CONTinuous: | Stepped sweep mode<br>Manual sweep mode<br>Continuous sweep mode |
| response: | none | |
| example: | wav:swe:mode STEP | |
| affects: | All tunable laser modules except Keysight 81649A, Keysight 81689A/B, 819x9A, and 81950A. | |

| | | |
|---|---|---|
| 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:<br>MAN:<br>CONT: | Stepped sweep mode<br>Manual sweep mode<br>Continuous sweep mode |
| example: | wav:swe:mode? -> STEP<END> | |
| affects: | All tunable laser modules, except 81950A. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:PMAX?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:PMAX?<wsp><start wavelength>,<br><stop wavelength> | |
| description: | Returns the power to the highest permissible power for the selected wavelength sweep. | |
| parameters: | start wavelength:<br>stop wavelength: | The wavelength at which the sweep starts as a **float** value.<br>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? 1540nm,1550nm -> +3.5500000E-004<END> |
| affects: | All tunable laser modules, and except 81950A. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat<wsp><mode> | |
| description: | Sets the repeat mode. Applies in stepped-sweep and manual-sweep modes. The 81960A, 81602A, 81606A, 81607A and 81608A also support TWOWay mode in continuous sweep mode. | |
| parameters: | ONEWay: | Every stepped or continuous sweep cycle starts at the start wavelength of the sweep and ends at the stop wavelength of the sweep |
| | TWOWay: | Every odd sweep cycle starts at the start wavelength of the sweep, and every even sweep cycle starts at the stop wavelength of the sweep.<br>Set the start and stop wavelength of the sweep using<br>[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STARt on page 175 and<br>[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STOP on page 176 respectively. |
| response: | none | |
| example: | wav:swe:rep twow | |
| affects: | All tunable laser modules, except 81950A. | |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:REPeat? | |
| description: | Returns the repeat mode. | |
| parameters: | none | |
| response: | ONEWay: | Every stepped or continuous sweep cycle starts at the start wavelength of the sweep and ends at the stop wavelength of the sweep |
| | TWOWay: | Every odd sweep cycle starts at the start wavelength of the sweep, and every even sweep cycle starts at the stop wavelength of the sweep.<br>Set the start and stop wavelength of the sweep using<br>[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STARt on page 175 and<br>[:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STOP on page 176 respectively. |
| example: | wav:swe:rep? -> ONEW<END> | |
| affects: | All tunable laser modules, except 81950A. | |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SOFTtrigger** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SOFTtrigger |
| description: | Softtrigger does the same as a normal (hardware) trigger at the backplane, but it doesn't cause a PM to take a measurement because it is only a (software) message sent to the tunable laser source. It only works in continuous sweep, but for 81602A, 81606A, 81607A, 81608A, 81609A it works in stepped mode, too.<br>Usage:<br>- Trigger input configuration: Start Sweep<br>- Start Sweep<br>- SoftTrigger |
| parameters: | none |
| response: | none |
| example: | sour0:wav:sweep:soft |
| affects: | All tunable laser modules except 81649A, Keysight 81689A/B, 81949A, 81989, and 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed<wsp><speed><br>[NM/S\|UM/S\|MM/S\|M/S\|] |
| description: | Sets the speed for continuous sweeping. |
| | Cannot be set while a sweep is running. |
| 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 that support continuous sweeps. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:SPEed?[<wsp>MIN\|MAX] | |
| description: | Returns the speed for continuous sweeping. | |
| parameters: | optional | MIN Returns the minimum sweep speed available.<br>MAX Returns the maximum sweep speed available. |

| | |
|---|---|
| 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 that support continuous sweeps. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt<wsp><start value><br>[PM\|NM\|UM\|MM\|M] |
| description: | Sets the starting point of the sweep. |
| | Cannot be set while a sweep is running. |
| parameters: | The wavelength at which the sweep starts as a **float** value.<br>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, except 81950A. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STARt?[<wsp>MIN\|MAX] | |
| description: | Returns the starting point of the sweep. | |
| parameters: | optional | MIN Returns the minimum start wavelength available.<br>This value is wavelength dependent.<br>MAX Returns the maximum start wavelength available.<br>This value is wavelength dependent. |
| 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, except 81950A. | |

| | |
|---|---|
| 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. |
| | Cannot be set while a sweep is running. |
| parameters: | The wavelength at which the sweep ends as a **float** value in meters.<br>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, except 81950A. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STOP?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STOP?[<wsp>MIN\|MAX] | |
| description: | Returns the end point of the sweep. | |
| parameters: | optional | MIN Returns the minimum start wavelength available. This value is wavelength dependent.<br>MAX Returns the maximum start wavelength available. This value is wavelength dependent. |
| 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:<br>1 or STARt:<br>2 or PAUSe:<br>3 or CONTinue: | Stop the sweep.<br>Start a sweep, run sweep.<br>Pause the sweep. (doesn't apply for continuous sweep)<br>Continue a sweep. (doesn't apply for continuous sweep) |
| | If you enable lambda logging (see [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:LLOGging on page 170 ) and modulation (see [:SOURce[n]][:CHANnel[m]]:AM:STATe[l] on page 127 ) simultaneously, a sweep cannot be started. | |
| | Generally, a continuous sweep can only be started if:<br>the trigger frequency, derived from the sweep speed and sweep step, is <= 40kHz, or <=1MHz for 81602A, 81606A, 81607A, 81608A, and 81960A.<br>the number of triggers, calculated from the sweep span and sweep span, is <=100001<br>the start wavelength is less than the stop wavelength.<br>In addition, a continuous sweep with lambda logging requires:<br>the trigger output to be set to step finished<br>modulation set to coherence control or off. | |
| response: | none | |
| example: | wav:swe STOP | |
| affects: | All tunable laser modules, except 81950A. | |

| | |
|---|---|
| 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<br>+1: Sweep is running |
| example: | wav:swe? -> +0<END> |
| affects: | All tunable laser modules, except 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:NEXT** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:NEXT |
| description: | Performs the next sweep step in stepped sweep if it is paused or in manual sweep. |
| parameters: | none |
| response: | none |
| example: | wav:swe:step:next |
| affects: | All tunable laser modules, except 81950A. |

| | |
|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:PREVious** |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:PREVious |
| description: | Performs one sweep step backwards in stepped sweep if it is paused or in manual sweep**.** |
| parameters: | none |
| response: | none |
| example: | wav:swe:step:prev |
| affects: | All tunable laser modules, except 81950A. |

| | |
|---|---|
| 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.<br>In continuous sweep mode, the end of a step is used for triggering. |
| parameters: | The width of the sweep step as a *float* value.<br>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, except 81950A. |

| | | |
|---|---|---|
| command: | **[:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:[WIDTh]?** | |
| syntax: | [:SOURce[*n*]][:CHANnel[*m*]]:WAVelength:SWEep:STEP:[WIDTh]?[<wsp>MIN\|MAX] | |
| description: | Returns the width of the sweep step | |
| parameters: | optional | MIN Returns the minimum step width available.<br>MAX Returns the maximum step width available. |
| response: | The sweep step as a **float** value in meters. | |
| example: | wav:swe:step? -> +5.00000000E-009<END> | |
| affects: | All tunable laser modules, except 81950A. | |

| | |
|---|---|
| command: | **:SOURce<*SlotNr*>:TRANsmitter:RECalibration** |
| syntax: | :SOURce<SlotNr>:TRANsmitter:RECalibration |
| description: | Recalibrates a reference transmitter module.<br>The result can be queried using :SYSTem:ERRor?:<br>It is either '+0,"No error"' or e.g. '-200,"Execution error (StatExecError)"'<br>Recalibration takes about 6.8 seconds, so it also takes this time to get the reply to the :SYSTem:ERRor? query. |
| parameters: | none |
| response: | none |
| example: | :SOUR1:TRAN:REC |
| affects: | 81490A |

| | |
|---|---|
| command: | **:SOURce<*SlotNr*>:TRANsmitter:RECalibration?** |
| syntax: | :SOURce<SlotNr>:TRANsmitter:RECalibration? |
| description: | Recalibrates a reference transmitter module.<br>Recalibration takes about 6.8 seconds, so it also takes this time to get the reply to this query. |
| parameters: | none |
| response: | The result is returned as an unquoted string:<br>“OK” or “ERROR - <error message>”. |
| example: | :SOUR1:TRAN:REC? —> OK<END> |
| affects: | 81490A |

| | |
|---|---|
| command: | **:SOURce<*SlotNr*>:TRANsmitter:OPOint<*laser (1,2)*>** |
| syntax: | :SOURce<SlotNr>:TRANsmitter:OPOint<*laser (1,2)*> <*data*> |
| description: | Sets the operating point for one of the reference transmitter's two laser diodes.<br>You can also set the minimum, maximum or default value. |
| parameters: | Operating point or MINimum \| MAXimum \| DEFault. |
| response: | none |
| example: | :sour1:tran:opo1 -17 |
| affects: | 81490A |

| | |
|---|---|
| command: | **:SOURce<*SlotNr*>:TRANsmitter:OPOint<*laser (1,2)*>?** |
| syntax: | :SOURce<SlotNr>:TRANsmitter:OPOint<*laser (1,2)*>? |
| description: | Get the operating point for one of the reference transmitter's two laser diodes.<br>You can also query the minimum, maximum or default value. |
| parameters: | none \| MINimum \| MAXimum \| DEFault |
| response: | The operating point. |
| example: | :sour1:tran:opo2 —> -17<END> |
| affects: | 81490A |

| | |
|---|---|
| command: | **:SOURce<*SlotNr*>:TRANsmitter:TCHeck?** |
| syntax: | :SOURce<*SlotNr*>:TRANsmitter:TCHeck? |
| description: | Get the temperature check status of the transmitter reference module. |
| parameters: | none |
| response: | A value ranging from 0.0 (recalibration necessary) to 1.0.<br>Values near 1.0 indicate the current temperature is close to the temperature at the last recalibration.<br>Values near 0.0 indicate the current temperature has drifted towards its maximum tolerance limit to trigger the UNCAL bit. |
| example: | :SOUR1:TRAN:TCH? —> 1.0<END> |
| affects: | 81490A |

# Signal Conditioning

The commands in this section allow you to control 8156x, and 8157x Attenuator modules

## The INPut and OUTput commands

| | | |
|---|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:ATTenuation** | |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:ATTenuation<wsp><value>[dB] \| MIN \| DEF \| MAX | |
| description: | Sets the attenuation factor ($a$) for the instrument. The attenuation factor is used, together with an offset factor ($a_{Offset}$) to set the filter attenuation ($a_{filter}$).<br>$a_{(new)}$ (dB) = $a_{filter\ (new)}$ (dB) + $a_{Offset}$ (dB)<br>Set the attenuation factor by sending a value (the default units are dB), or by sending MIN, DEF, or MAX. | |
| parameters: | <value>[dB]<br>MIN \| DEF<br>MAX | The attenuation in dB.<br>The values where $a_{filter}$ = 0dB<br>The value where $a_{filter}$ is at its greatest. |
| response: | none | |
| example: | INP1:ATT 14dB | |
| affects: | All attenuator modules | |

| | | |
|---|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:ATTenuation?** | |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:ATTenuation?<wsp> MIN \| DEF \| MAX | |
| description: | Returns the current attenuation factor ($a$), in dB.<br>$a$ (dB) = $a_{filter}$ (dB) + $a_{Offset}$ (dB) | |
| parameters: | MIN \| DEF \| MAX | Returns the minimum, default, or maximum value of the attenuation factor possible. |
| response: | 4 byte Intel floating point; attenuation in dB. | |
| example: | INP1:ATT? -> 14<END> | |
| affects: | All attenuator modules | |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:OFFSet** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:OFFSet<wsp><value>[dB] \| MIN \| DEF \| MAX |
| description: | Sets the offset factor ($a_{\text{Offset}}$) for the instrument. This factor does not affect the filter attenuation ($a_{\text{filter}}$). It is used to offset the attenuation factor values. This offset factor is used, with the attenuation factor, to set the attenuation of the filter. In this way it is possible to compensate for external losses.<br>$a_{(\text{new})}$ (dB) = $a_{\text{filter}}$ (dB) + $a_{\text{Offset (new)}}$ (dB)<br>Set the offset factor by sending a value (the default units are dB), or by sending MIN, DEF, or MAX. |
| parameters: | <value>[dB] The offset factor ($a_{\text{Offset}}$) in dB.<br>MIN Sets the minimum value for $a_{\text{Offset}}$ = - 200dB.<br>DEF Sets the default value for $a_{\text{Offset}}$ = 0dB.<br>MAX Sets the maximum value for $a_{\text{Offset}}$ = + 200dB. |
| response: | none |
| example: | INP1:OFFS 2dB |
| affects: | All attenuator modules |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:OFFSet?** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:OFFSet?<wsp>MIN \| DEF \| MAX |
| description: | Returns the current value of the offset factor ($a_{\text{Offset}}$), in dB. |
| parameters: | MIN \| DEF \| MAX Returns the minimum, default, or maximum value of the offset factor. |
| response: | 4 byte Intel floating point; offset in dB. |
| example: | INP1:OFFS? -> 2<END> |
| affects: | All attenuator modules |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:OFFSet:DISPlay** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:OFFSet:DISPlay |
| description: | Sets the offset factor ($a_{\text{Offset}}$) such that the attenuation factor is zero.<br>$a_{\text{Offset (new)}}$ (dB) = $a_{\text{Offset (old)}}$ (dB) - $a_{(\text{old})}$ (dB) = - $a_{\text{filter}}$ (dB) |
| parameters: | none |

| | |
|---|---|
| response: | none |
| example: | INP1:OFFS:DISP |
| affects: | All attenuator modules |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:OFFSet:POWermeter** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:OFFSet:POWermeter<wsp><slot>,<channel> |
| description: | Sets the offset factor ($a_{Offset}$) to the difference between a power value measured by another powermeter (hosted by the same mainframe) ($P_{ext}$) and the power value measured by the attenuator module's monitor diode ($P_{att}$).<br>$a_{Offset}$ (dB) = $P_{att}$ *(dBm)* - $P_{ext}$ (dBm) |
| parameters: | <slot> Slot number of the external powermeter.<br><channel> Channel number of the external powermeter. |
| response: | none |
| example: | INP1:OFFS:POW 4,2 |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:ATTenuation:SPEed** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:ATTenuation:SPEed<wsp><value> \| MIN \| MAX \| DEF |
| description: | Sets the filter transition speed; the speed at which the module moves from one attenuation to another (in dBs). |
| parameters: | <value> The filter transition speed in dB/s.<br>MIN \| MAX \| DEF Sets the filter transition speed to the module limits, or the module default. |
| response: | none |
| example: | INP1:ATT:SPE 2 |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:ATTenuation:SPEed?** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:ATTenuation:SPEed?<wsp> MIN \| MAX \| DEF |
| description: | Without the optional parameter, queries the transition speed of the filter. |
| parameters: | MIN \| MAX \| DEF Queries the transition speed limits, or the module default. |

| | |
|---|---|
| response: | 4 byte Intel floating point; transition speed in dB/s. |
| example: | INP1:ATT:SPE? -> 2<END> |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:WAVelength** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:WAVelength<wsp><value>[PM \| NM \| UM\| MM \| M] \| MIN \| MAX \| DEF |
| description: | Sets the attenuator module's operating wavelength.<br>This value is used to compensate for the wavelength dependence of the filter, and to calculate a wavelegth dependent offset from the user offset table (if enabled). |
| parameters: | <value> The wavelength in meters (if you do not specify a unit).<br>MIN \| MAX \| DEF Sets the wavelength to the module limits, or the module default. |
| response: | none |
| example: | INP1:WAV +1.55000000E-006 |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:INPut[*n*][:CHANnel[*m*]]:WAVelength?** |
| syntax: | :INPut[*n*][:CHANnel[*m*]]:WAVelength?<wsp>MIN \| MAX \| DEF |
| description: | Without the optional parameter, queries the operating wavelength of the attenuator. |
| parameters: | MIN \| MAX \| DEF Queries the operating wavelength limits, or the module default. |
| response: | 4 byte Intel floating point; wavelength in m. |
| example: | INP1:WAV -> +1.55000000E-006<END> |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:APMode** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:APMode<wsp><OFF(0) \| ON(1)> |
| description: | The use of this command is optional and has no effect on operation.<br>Included for compatibility with Keysight 8156A mainframe. |

| | |
|---|---|
| parameters: | OFF or 0<br>ON or 1 |
| response: | none |
| example: | OUTP1:APMode OFF |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:APMode?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:APMode? |
| description: | Queries whether the user has amended the power value or the attenuation value.<br>This use of this command is optional.<br>Included for compatibility with Keysight 8156A mainframe. |
| parameters: | none |
| response: | *boolean* 0 User has amended the attenuation value.<br>1 User has amended the power value. |
| example: | OUTP1:APMode? -> 0<END> |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer<wsp><value>[PW \| NW \| UW \| MW \| W \| DBM ] \| MIN \| MAX \| DEF |
| description: | Sets the output power value (*P* ).<br>If your attenuator module does not include the power control feature, the new filter attenuation ($a_{\text{filter (new)}}$ ) is calculated from the reference power ($P_{\text{ref}}$ ) :<br>$P_{\text{set(new)}}$ *(dBm)* = $P_{\text{ref}}$ *(dBm)* - $a_{\text{filter (new)}}$ (dB) - $P_{\text{Offset}}$ (dB)<br>If your attenuator module includes the power control feature, the filter attenuation is changed until the set power (measured by the module's internal power meter) has been reached. $P_{\text{set(new)}}$ *(dBm)* = $P_{\text{att(new)}}$ *(dBm)* - $P_{\text{offset}}$ *(dB)*<br>If the set power cannot be achieved ExP (indicating 'Excessive Power' ) is displayed, and the appropriate GPIB status bit is set.<br>The status of these bits can be queried using :STATus:OPERation:CONDition[:LEVel0]? on page 56 |
| parameters: | <value> Desired output power (if unit not specified current unit is used).<br>MIN \| MAX \| DEF Sets the output power to the module limits, or the module default. |
| response: | none |
| example: | OUTP1:POW 12 |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer<wsp>MIN \| MAX \| DEF |
| description: | Without the optional parameter, queries the output power value. |
| parameters: | MIN \| MAX \| DEF Queries the output power module limits, or the module default. |

| | |
|---|---|
| response: | 4 byte Intel floating point; output power in current power unit. |
| example: | OUTP1:POW? -> 12<END> |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:REFerence** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:REFerence<wsp><value>[PW \| NW \| UW \| MW \| W \| DBM ] \| MIN \| MAX \| DEF |
| description: | Sets the reference power ($P_{ref}$). The reference power is used to calculate the filter attenuation ( $a_{filter}$ ) from the output power ($P$) . A change to the reference power does not affect the filter attenuation.<br>$P_{set(new)}(dBm) = P_{ref\,(new)}(dBm) - a_{filter}(dB) - P_{Offset}(dB)$ |
| parameters: | <value> Desired reference power (if unit not specified current unit is used).<br>MIN \| MAX \| DEF Sets the reference power to the module limits, or the module default. |
| response: | none |
| example: | OUTP1:POW:REF 6dBm |
| affects: | Attenuator modules without power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:REFerence?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:REFerence?<wsp>MIN \| MAX \| DEF |
| description: | Without the optional parameter, queries the reference power value. |
| parameters: | MIN\|MAX\|DEF Queries the reference power limits, or the module default. |
| response: | 4 byte Intel floating point; reference power in current power unit. |
| example: | OUTP1:POW:REF? -> 6<END> |
| affects: | Attenuator modules without power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:REFerence:POWermeter** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:REFerence:POWer<wsp><slot>,<channel> |
| description: | Copies the power value ($P_{ext}$) from an external powermeter module (hosted by the same mainframe) to the attenuator module's reference power parameter ($P_{ref}$):<br>$P_{ref}(dBm) = P_{ext}(dBm) + a_{filter}(dB)$ |
| parameters: | <slot> Slot number of the powermeter.<br><channel> Channel number of the powermeter. |
| response: | none |
| example: | OUTP1:POW:REF:POW 4,2 |
| affects: | Attenuator modules without power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:OFFSet** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:OFFSet<wsp><value>[DB] \| MIN \| MAX \| DEF |
| description: | Sets a power offset ($P_{offset}$). This factor is used to offset the power value. It does not affect the filter, nor does it change the power output at the attenuator module.<br>$P_{set(new)}$ *(dBm)* = $P_{att}$*(dBm)* - $P_{offset(new)}$ *(dB)*<br>If the wavelength offset table is enabled, the corresponding $\lambda$ offset is added to this offset. |
| parameters: | <value> The power offset required, in dB<br>MIN \| MAX \| DEF Queries the module limits, or the default. |
| response: | none |
| example: | OUTP1:POW:OFFS 2 |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:OFFSet?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:OFFSet? <wsp>MIN \| MAX\| DEF |
| description: | Without the optional parameter, queries the power offset value. |
| parameters: | MIN \| MAX \| REF Queries the power offset limits, or the module default. |
| response: | 4 byte Intel floating point; power offset in current power units. |
| example: | OUTP1:POW:OFFS? -> 2<END> |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:OFFSet:POWermeter** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:OFFSet:POWermeter<wsp><slot>,<channel> |
| description: | Calculates the power offset by subtracting the power value measured by another powermeter (hosted by the same mainframe) from the power value measured by the attenuator's integrated powermeter, and stores it as $P_{offset}$ .<br>$P_{offset(new)}$ (dBm) = $P_{att}$ (dBm) - $P_{ext}$ (dBm) + $P_{offset}$ ( $\lambda$ ) (dB) |
| parameters: | <slot> Slot number of the external powermeter.<br><channel> Channel number of the external powermeter. |

| | |
|---|---|
| response: | none |
| example: | OUTP1:POW:OFFS:POW 4,4 |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:CONTRol** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:CONTRol<wsp>OFF(0) \| ON(1) |
| description: | Sets whether the power control mode is on or off.<br>If power control is enabled, the attenuator automatically compensates for changes to input power. |
| parameters: | OFF or 0 Output power follows changes to input power.<br>ON or 1 The filter position automatically adjusts to compensate for changes to input power, so maintaining the output power set by the user. |
| response: | none |
| example: | OUTP1:POW:CONTR ON |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:CONTRol?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:CONTRol? |
| description: | Queries whether the power control mode is on or off. |
| parameters: | none |
| response: | *boolean* 0 The power control mode is off<br>1 The power control mode is on. |
| example: | OUTP1:POW:CONTR? -> 0<END> |
| affects: | Attenuator modules with power control. |

| | | |
|---|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:UNit** | |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:UNit<wsp>DBM(0) \| WATT(1) | |
| description: | Sets whether the power unit used is dBm or Watts.<br>This setting affects $P_{set}$ , $P_{ref}$ (if available), and $P_{act}$ | |
| parameters: | DBM (or 0)<br>WATT (or 1) | Sets the power unit to dBm<br>Sets the power unit to W |
| response: | none | |
| example: | OUTP1:POW:UN DBM | |
| affects: | All attenuator modules, and 81950A. | |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:POWer:UNit?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:POWer:UNit? |
| description: | Queries whether the power unit is dBm or W |
| parameters: | none |
| response: | *boolean* 0 The power unit is dBm<br>1 The power unit is W. |
| example: | OUTP1:POW:UN? -> 0<END> |
| affects: | All attenuator modules, and 81950A. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:[STATe]** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:[STATe]<wsp>OFF(0) \| ON(1) |
| description: | Sets the state of the shutter. |
| parameters: | OFF or 0 Shutter closed.<br>ON or 1 Shutter open. |
| response: | none |
| example: | OUTP1:STAT OFF |
| affects: | All attenuator modules, and 81950A. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:[STATe]?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:[STATe]? |
| description: | Queries the state of the shutter. |
| parameters: | none |
| response: | *boolean* 0 The shutter is closed.<br>1 The shutter is open. |
| example: | OUTP1:STAT? -> 0<END> |
| affects: | All attenuator modules, and 81950A. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:STATe:APOWeron** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:STATe:APOWeron<wsp>OFF(0) \| ON(1) |
| description: | Sets the state of the shutter when the mainframe is turned on. |

| | | |
|---|---|---|
| parameters: | OFF or 0<br>ON or 1 | Shutter closed after mainframe power on.<br>Shutter open after mainframe power on. |
| response: | none | |
| example: | OUTP1:APOW OFF | |
| affects: | All attenuator modules | |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:STATe:APOWeron?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:STATe:APOWeron? |
| description: | Queries the state of the shutter at power on. |
| parameters: | none |
| response: | *boolean* 0 The shutter is open after mainframe power on.<br>1 The shutter is closed after mainframe power on. |
| example: | OUTP1:APOW? -> 0<END> |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:ATIMe** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:ATIMe<wsp><value>[ NS \| US \| MS\| S ] |
| description: | Sets the powermeter averaging time, which can, if the attenuator's power contol feature is activated, affect how the attenuator compensates for changes to input power. |
| parameters: | <value> The averaging time (in seconds if no unit specified). |
| response: | none |
| example: | OUTP1:ATIM 1s |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*]:CHANnel[*m*]]:ATIMe?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:ATIMe? |
| description: | Queries the powermeter averaging time. |
| parameters: | none |

| | |
|---|---|
| response: | *4 byte Intel floating point; the averaging time in seconds* |
| example: | OUTP1:ATIM? -> 1<END> |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:CORRection:COLLection:ZERO** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:CORRection:COLLection:ZERO |
| description: | Zeros the electrical offsets of the attenuator's integrated powermeter. |
| parameters: | none |
| response: | none |
| example: | OUTP1:CORR:COLL:ZERO |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:CORRection:COLLection:ZERO:ALL** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:CORRection:COLLection:ZERO:ALL |
| description: | Zero all available powermeter channels in the mainframe. |
| parameters: | none |
| response: | none |
| example: | OUTP1:CORR:COLL:ZERO:ALL |
| affects: | Powermeter modules; attenuator modules with power control, and return loss modules. |

| | |
|---|---|
| command: | **:OUTPut*n*[:CHANnel[*m*]]:CORRection:COLLection:ZERO?** |
| syntax: | :OUTPut*n*[:CHANnel[*m*]]:CORRection:COLLection:ZERO? |
| description: | Queries the status of the last **:**OUTPutn[:CHANnel[m]]:CORRection:COLLection:ZERO operation. |
| parameters: | none |
| response: | integer 0 = OK, otherwise not OK. |
| example: | OUTP:CORR:COLL:ZERO? -> 0<END> |
| affects: | Attenuator modules with power control. |

The table of wavelength-dependent offsets

When enabled, the attenuator uses its λ offset table to compensate for wavelength dependent losses in the test set-up. This table contains, for each wavelength specified, the additional power offset to be applied.

- If the attenuator module is set to a wavelength corresponding to an entry in its λ offset table, the stored offset is added to the global power offset.
- If the attenuator module is set to a wavelength between entries in its λ offset table, linear interpolation is used to calculate the appropriate offset to add to the global power offset.
- If the attenuator module is set to a wavelength beyond the range of the entries in its λ offset table, the offset stored for the nearest wavelength is added to the global power offset.
- Whether an exact, interpolated, or extrapolated offset value is applied, the algorithm applied can be queried using :STATusn:OPERation:CONDition? on page 59

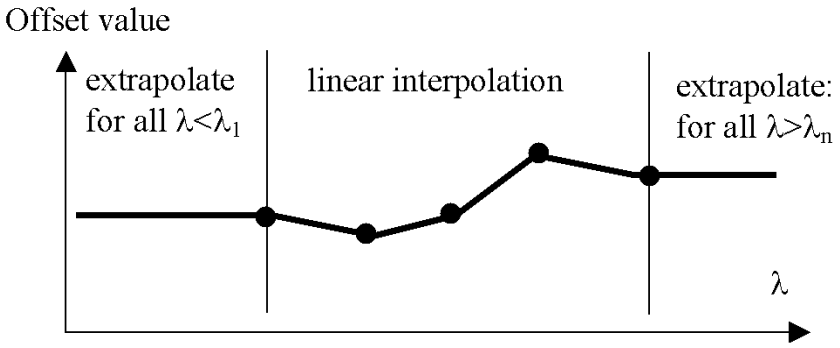



Figure 6 Extrapolation and interpolation of attenuator module λ offset table

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:STATe** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:STATe<wsp>OFF(0) \| ON(1) |
| description: | Specifies whether the attenuator uses its λ offset table to compensate for wavelength dependent losses in the the test set-up. This table contains, for each wavelength specified, the additional power offset to be applied.<br>This command does not affect the module's internal enviromental temperature and optical wavelength compensation, which remain active. |
| parameters: | OFF or 0 The offset table is not used to compensate for wavelength dependent losses.<br>ON or 1 The attenuator adds the appropriate value from its λ offset table to the global power offset. |
| response: | none |
| example: | CONF1:OFFS:WAV:STAT ON |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:STATe?** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:STATe? |
| description: | Queries whether the attenuator uses power values from its λ offset table . |
| parameters: | none |
| response: | *boolean* 0 The offset table is not used.<br>1 The attenuator uses its λ offset table. |
| example: | CONF1:OFFS:WAV:STAT? -> 0<END> |
| affects: | All attenuator modules. |

| | | |
|---|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:VALue** | |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:VALue<wsp><lambda>[PM \| NM \| UM \| MM\| M],<offset[DB]> \| TOREF | |
| description: | Adds a value pair (wavelength; offset) to the offset table, or overwrites an existing value pair.<br>The offset table entries are ordered from shortest to longest wavelength. | |
| parameters: | <lambda><br><offset><br><br><br>TOREF | The wavelength for the offset table entry, in m<br>The power offset to be applied at <lambda>.<br>To query the current power value measured by attenuator with power control see :FETCh[n][:CHANnel[m]][:SCAlar]:RETurnloss? on page 88 (Fetch) and :READ[n][:CHANnel[m]][:SCALar]:POWer[:DC]? on page 92 (Read).<br>Calculates the difference between the power measured by an external powermeter (hosted in the same mainframe) and the power measured by the attenuator module's integrated powermeter, and stores it as the offset.<br>$P_{Offset(\lambda)}(dB) = P_{att}(dBm) - P_{ext}(dBm)$<br><br>See: :CONFigure[n][:CHANnel[m]]:OFFSet:WAVelength:REFerence on page 198 (Attenuator modules with power control only). |
| response: | none | |
| example: | CONF1:OFFS:WAV:VAL +1.55000000E-006,TOREF | |
| affects: | All attenuator modules (TOREF applicable to attenuator modules with power control only). | |

| | | |
|---|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:REFerence** | |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:REFerence<wsp><slot>,<channel> | |
| description: | Specifies the slot and channel of the external powermeter (hosted in the same mainframe as the attenuator module) used by TOREF.<br>See: :CONFigure[n][:CHANnel[m]]:OFFSet:WAVelength:VALue on page 198 | |
| parameters: | <slot><br><channel> | Slot number of the powermeter.<br>Channel number of the powermeter. |
| response: | none | |
| example: | CONF1:OFFS:WAV:REF 4,2 | |
| affects: | Attenuator modules with power control. | |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:REFerence?** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:REFerence? |
| description: | Queries the currently selected slot and channel of the external powermeter (hosted in the same mainframe as the attenuator module) used by TOREF.<br>See: :CONFigure[n][:CHANnel[m]]:OFFSet:WAVelength:VALue on page 198 |
| parameters: | none |
| response: | the slot and channel of the external powermeter as *integer* values. |
| example: | CONF1:OFFS:WAV:REF? -> +2,+1<END> |
| affects: | Attenuator modules with power control. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:WAVelength?** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:WAVelength?<wsp><index> |
| description: | Queries a wavelength value from its position, or index, in the offset table. Offset table entries are ordered from shortest to longest wavelength. The first index number = 1. |
| parameters: | <index> The position of the wavelength value in the offset table. |
| response: | *4 byte Intel floating point; the wavelength in meters* |
| example: | CONF1:OFFS:WAV:VAL:WAV? 1 -> +1.55000000E-006<END> |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:OFFSet?** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:OFFSet?<wsp><index \| wavelength [PM \| NM \| UM \| MM\| M],> |
| description: | Queries an offset value from the position, or index, of the associated wavelength in the offset table. Offset table entries are ordered from shortest to longest wavelength. The first index number =1.<br>Or: Queries the offset applied for a particular wavelength. |
| parameters: | <index> The position of the value pair (wavelength; offset) in the offset table.<br><wavelength> The wavelength for the offset table entry, in m |

| | |
|---|---|
| response: | *4 byte Intel floating point; the offset* |
| example: | CONF1:OFFS:WAV:VAL:OFFS? 1 -> 2 |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:PAIR?** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:PAIR?<wsp><index \| wavelength[PM \| NM \| UM \| MM\| M],> |
| description: | Queries an offset value pair (wavelength:offset) from the position, or index, of the associated wavelength in the offset table. Offset table entries are ordered from shortest to longest wavelength.<br>Or: Queries the offset value pair (wavelength:offset) applied for the specified wavelength. |
| parameters: | <index> The position of the wavelength; offset value pair in the offset table.<br><wavelength> The wavelength for the offset table entry, in m |
| response: | *char$ in SCPI block format (Intel byte order); wavelength:offset* |
| example: | CONF1:OFFS:WAV:VAL:PAIR? 1 -> "+1.55000000E-006:2" |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:DELete** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:DELete<wsp><index \| wavelength[PM \| NM \| UM \| MM\| M],> |
| description: | Deletes an offset value pair (wavelength:offset) from the position, or index, of the associated wavelength in the offset table. Offset table entries are ordered from shortest to longest wavelength.<br>Or: Deletes the offset value pair (wavelength:offset) associated with the specified wavelength. |
| | Deleting a value pair decrements the index value of every subsequent value pair by 1. When using this command, you may prefer to work from large to small index values. |
| parameters: | <index> The position of the wavelength:offset value pair in the offset table.<br><wavelength> The wavelength for the offset table entry, in m |
| response: | *none* |
| example: | CONF1:OFFS:WAV:VAL:DEL 1 |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:DELete:ALL** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:VALue:DELete:ALL |
| CAUTION | *This command clears the offset table!* |
| description: | Deletes every value pair (wavelength:offset) from the offset table. |
| parameters: | none |
| response: | *none* |
| example: | CONF1:OFFS:WAV:VAL:DEL:ALL |
| affects: | All attenuator modules. |

| | |
|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:TABle?** |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:TABle? |
| description: | Queries the complete the offset table. |
| parameters: | none |
| response: | *SCPI binary block format format (Intel byte order); wavelength:offset pairs in ascending order.*<br>Each value pair is transferred as 12 bytes; 8 bytes represent the wavelength, 4 bytes represent the offset. |
| example: | CONF1:OFFS:WAV:TAB? -> binary block interpreted as, for example:<br>1.55e-6 \| 12 \| 1.7e-6 \|3.4 \|..... |
| affects: | All attenuator modules. |

| | | |
|---|---|---|
| command: | **:CONFigure[*n*][:CHANnel[*m*]]:OFFSet:WAVelength:TABle:SIZE?** | |
| syntax: | :CONFigure[*n*]][:CHANnel[*m*]]:OFFSet:WAVelength:TABle:SIZE?<wsp>MAX \| MIN | |
| description: | Without optional parameter, queries the size of the offset table. | |
| parameters: | MAX | Queries the maximum size of the offset table.<br>(available flash memory -> 1000 entries) |
| | MIN | Queries the minimum size of the offset table. (should -> 0) |

| | |
|---|---|
| response: | *4 byte unsigned integer; offset table size.* |
| example: | CONF1:OFFS:WAV:TAB:SIZE? -> 50 |
| affects: | All attenuator modules. |

## Compatibility of the 81560A/1A/6A/7A modular attenuator family to the 8156A attenuator

The 81560A/1A/6A/7A modular attenuator family is intended to be SCPI compatible with the 8156A attenuator but, because the modular attenuator family is part of a platform concept, there are some compatibility limitations. This section describes the differences between the SCPI syntax and the command semantic and how to deal with them.

**NOTE** The page numbers in brackets refer to pages in the 8156A Attenuator Operating and Programming Guide, Second Edition, May 2000 with part number 08156-91011:E0500.

### Slot Numbers

INPUT and OUPUT SCPI commands (page 106-114) are used to access the functionality of the 8156A Attenuator. The 816xA/B mainframes are able to host a number of modules, so a slot identifier is needed. This slot identifier was not required by the 8156A attenuator. Simply substitute INPutn for INPut, and OUTPutn for OUTPut, where n is the slot number of your attenuator module.

### Example1: Setting the attenuation

8156A:**INP:ATT 10 dB**

8156x:**INP2:ATT 10 dB**
(when the attenuator is hosted in Slot 2)

### Example2: Setting the output power

8156A:**OUTP:POW 10 dBm**

8156x:**OUTP2:POW 10 dBm**
(when the attenuator is hosted in Slot 2)

If you forget to enter the slot number, one of the following error messages is placed in the SCPI error queue:

-303,"Module slot empty or slot / channel invalid"

-301,"Module doesn't support this command (StatCmdUnknown)"

TIP: Query the SCPI error queue using SYST:ERR?

TIP: You can use INPut commands without a slot number if the 81560A/1A/6A/7A module is hosted by Slot 1. An INPut command is applied to Slot 1 by default.

### Command Semantic

All the INPut and OUTPut commands applicable to the 8156A attenuator are also supported by the 81560A/1A/6A/7A modular attenuator family. In addition, the 81560A/1A/6A/7A modular attenuator family supports new commands to access its new features. To support these new features, and improve the usability of the instrument, the meaning (the semantic) of some existing commands has changed. This section lists all commands already available to the 8156A attenuator, notes whether the semantic of the command has changed, and where applicable, suggests how to handle the change.

**Table 7 Comparison of command semantics beween 8156A attenuator and 8156xA modular attenuator family.**

| Command | Comment |
|---|---|
| INPut:ATTenuation | No change. |
| INPut:ATTenuation? | No change. |
| INPut:LCMode | No longer supported. Use the wavelength dependent offset command. |
| INPut:LCMode? | No longer supported. Use the wavelength dependent offset command. |
| INPut:OFFSet | No change. |
| INPut:OFFSet? | No change. |
| INPut:OFFSet:DISPlay | No change. |
| INPut:WAVelength | No change. |

| Command | Comment |
|---|---|
| OUTPut:APMode | The 8156A uses this command to calculate a base power level while the instrument switches to another mode. This behavior is replaced by a mechanism that is easier to use.<br>To calculate a power level at the device under test, formerly known as *through* power, the 81560A and 81561A attenuator modules use a reference power. This reference power can be modified either via the user interface or by using the SCPI command OUTPut:POWer:REFerence. The power is calculated from the attenuation and the reference power using this formula:<br>$P_{set}$ (dBm) = $P_{ref}$ (dBm) - $\alpha$ (dB)<br>The 81566A and 81567A attenuator modules do not need a base power level because they are able to measure the output power directly.<br>Despite these new features, the 8156x modular attenuator family supports this command, but only to address compatibility issues. The command only sets an internal flag, which can be read using OUTput:APMode? You are free to choose between adjusting the output power or adjusting the attenuation factor. |
| OUTPut:APMode? | It is now possible to adjust both power and attenuation without changing the mode, so this command is supported only to address compatibility issues. This query returns whether power (1) or attenuation (0) was changed last. All other actions have no effect on this internal flag. |
| OUTPut:POWer | Except that the base power level is determined in another way (see OUTPut:APMode), there is no change to the semantic of this command. |
| OUTPut:POWer? | No change. |
| OUTPut:STATe | No change. |
| OUTPut:STATe? | No change. |
| OUTPut:STATe:APOWeron | No change. |
| OUTPut:STATe:APOWeron? | No change. |

## Display and System Commands

The commands to adjust the instrument display (page 104ff) and query the error queue (page 122) also work with the 816xA/B platform:

DISPlay:BRIGhtness

DISPlay:ENABle

SYSTem:ERRor?

### IEEE Commands

Every SCPI compatible measurement instrument implements a subset of the IEEE SCPI command set. The 8156A attenuator and the 81560A/1A/6A/7A attenuator family use almost the same subset. The following IEEE commands are available when using the 8156A but not available when using the 81560A/1A/6A/7A modules (page 93ff):

*RCLRecover parameter setup

*SAVSave parameter setup

*SRE and *SRE?Status request register

### Status Commands

The instrument status model can be controlled, and its current state queried, using commands from the SCPI STATus subtree. All the STATus commands available for the 8156A attenuator are supported by the 81560A/1A/6A/7A modular attenuator familiy except:

STATus:OPERation:PTRansition

STATus:OPERation:NTRansition

STATus:QUEStionable:PTRansition

STATus:QUEStionable:NTRansition

There are new status bits available to query the current modular attenuator state.

### User Calibration Data

The user calibration mode of the 8156A overrides the attenuator's built-in wavelength calibration table, so allowing user defined wavelength compensation. Since the 81560A/1A/6A/7A modular attenuator family features an improved factory calibration process, so this user calibration feature (page 123) is not supported.

The 81560A/1A/6A/7A modular attenuator family includes a user configurable offset function. If you enable this feature, the module's internal wavelength compensation remains active and you are able to compensate for additional external wavelength-dependent losses within the measurement setup by creating a wavelength/offset table. For additional information, refer to our Application Note "Variable Optical Attenuator in BER Test Applications", part number 5988-3159EN.

# Signal Routing

The commands in this section allow you to control Keysight 8159x Optical Switch modules

| | |
|---|---|
| command: | **:ROUTe[*n*][:CHANnel[*m*]]** |
| syntax: | :ROUTe[*n*]:[CHANnel[*m*]]<wsp><channel_list> |
| description: | Sets the channel route between two ports. |
| | When you use switches with dependent connections (e.g. the 2x2 switch), it is possible that one route configuration automatically changes another connection! |
| parameters: | n: the slot number of the switch module<br>m: the switch channel within the selected switch module.<br>e.g. for dual 1 x 2 module m = 1 for switch 1; m = 2 for switch 2<br>channel_list the route between left and right ports.<br>channel_list format: [A....Z],[1....n] |
| response: | If an invalid route is selected the following error message is returned - "StatParamError" |
| example: | rout3:chan1 A,2 (module in slot 3,channel 1, connect port A with port 2) |
| affects: | All switch modules |

| | |
|---|---|
| command: | **:ROUTe[*n*][:CHANnel[*m*]]?** |
| syntax: | :ROUTe[*n*]:[CHANnel[*m*]]<wsp><channel_list> |
| description: | Queries the current channel route of the switch for a specific module and switch channel. |
| parameters: | n: the slot number of the switch module<br>m: the switch channel within the selected switch module. Default value is 1. |
| response: | [A.....Z],[1.....n];[A.....Z],[1.....n] as a text string.<br>"," separates input and output ports of a specific connection.<br>";" separates parallel connections (as used in 2x2 switch). |
| example: | rout3:chan1? -> A,1 simple 1xN switch<br>rout2:chan1? -> A,2;B,1 (2x2 crossover switch in crossover config). |
| affects: | All switch modules |

| | |
|---|---|
| command: | **:ROUTe[*n*][:CHANnel[*m*]]:CONFig?** |
| syntax: | :ROUTe[*n*]:[CHANnel[*m*]]:CONFig? |
| description: | Queries the switch configuration of the instrument. For each channel, the minimum and maximum channel number of each port is given. |
| | |
| parameters: | none |
| response: | <j>,<k>;<l>,<m> as text string where:<br><j> is the first port character on the left<br><k> is the last port character on the left<br><l> is the minmimum port number on the right<br><m> is the maximum port number on the right |
| example: | rout2:conf? -> A,B;1,2 (2 left and 2 right ports) |
| affects: | All switch modules |

| | |
|---|---|
| command: | **:ROUTe[*n*][:CHANnel[*m*]]:CONFig:ROUTe?** |
| syntax: | :ROUTe[*n*]:[CHANnel[*m*]]:CONFig:ROUTe? |
| description: | Queries the allowed switch routes of an instrument. |
| | |
| parameters: | none |
| response: | [A.....Z],[1.....n];[A.....Z],[1.....n].[A.....Z],[1.....n] as a text string.<br>"," separates input and output ports of a single connection.<br>";" separates parallel connections<br>"." separates possible switch states |
| example: | rout2:conf:rout? -> A,1;B,2.A,2;B,1<br>2x2 x-over switcch:<br>state 1: When A to 1 then B to 2nd connection (straight)<br>state 2: When A to 2 then B to 1st connection (cross-over) |
| affects: | All switch modules |

# Triggering - The TRIGger Subsystem

The TRIGger Subsystem allows you to configure how the instrument reacts to incoming or outgoing triggers.

**Table 8 Triggering and Power Measurements**

| Hardware Triggering | 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 9 Generating Output Triggers from Power Measurements**

| Hardware Triggering | 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. |

| | |
|---|---|
| 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.<br>2 or NODEB: Generates trigger at the Output Trigger Connector. |
| | A hardware trigger cannot be effective in the DISabled triggering mode but can be effective in DEFault, PASSthrough or LOOPback triggering modes, see :TRIGger:CONF on page 214 for information on triggering modes. |

| | |
|---|---|
| | :TRIGger on page 217 describes the :TRIGger command for advanced users using :TRIGger:CONF:EXTended on page 217. |
| 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:<br>SMEasure:<br><br><br>CMEasure:<br><br><br>NEXTstep:<br>SWStart: | Ignore incoming trigger.<br>Start a single measurement. If a measurement function is active, see :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104, one sample is performed and the result is stored in the data array, see :SENSe[n][:CHANnel[m]]:FUNCtion:RESult? on page 100.<br>Start a complete measurement. If a measurement function is active, see:SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104, a complete measurement function is performed.<br>Perform next step of a stepped sweep.<br>Start a sweep cycle. |
| | You must prearm a wavelength sweep or a measurement function before an action can be triggered:<br>First, set the incoming trigger response.<br>Then:<br>prearm a wavelength sweep using [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:[STATe] on page 176. The wavelength of the tunable laser module is set to the start wavelength of the sweep.<br>or prearm a measurement function using :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104.<br>NOTE: If a trigger signal arrives at the Input Trigger Connector at the same time that the :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104 command is executed, the first measurement value is invalid. You should always discard the first measurement value in this case.<br>The module performs the appropriate action when it is triggered. | |
| response: | none | |
| example: | trig1:inp ign | |
| affects: | All tunable laser modules except 81950A, power meters, and return loss modules, and attenuators with power control. | |
| | If you use the Keysight 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 104, one sample is performed and the result is stored in the data array, see :SENSe[n][:CHANnel[m]]:FUNCtion:RESult? on page 100. |
| | CMEasure: | Start a complete measurement. If a measurement function is active, see :SENSe[n][:CHANnel[m]]:FUNCtion:STATe on page 104, 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 (except 81950A), power meters, and return loss modules, and attenuators with power control. | |
| 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. | |
| | See Table 8 on page -208, for information on how this command affects triggering power measurements. | |
| parameters: | A *boolean* value: | OFF or 0: trigger rearming disabled<br>ON or 1: trigger rearming enabled (default) |
| | If you return to Local control, all modules return to the default setting. | |
| response: | none | |
| example: | trig1:inp:rearm 0 | |
| affects: | All 8163A/B series power meter modules, and 8161x series return loss 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<br>1: trigger rearming enabled (default) |
| example: | trig1:inp:rearm? -> 0<END> |
| affects: | All 8163A/B series power meter modules, and 8161x series return loss modules. |
| dual sensors: | Can only be sent to master channel, slave channel parameters are identical. |

| | |
|---|---|
| command: | **:TRIGger[*n*][:CHANnel[*m*]]:OFFSet** |
| syntax: | :TRIGger[*n*][:CHANnel[*m*]]:OFFSet <value> |
| description: | Sets the number of incoming triggers received before data logging begins. |
| parameters: | <value> - an **integer** value. (maximum possible value is 1e+9) |
| response: | none |
| example: | trig1:offs 5 |
| affects: | All 81636B and 81637B series power meter modules. |

| | |
|---|---|
| command: | **:TRIGger[*n*][:CHANnel[*m*]]:OFFSet?** |
| syntax: | :TRIGger[*n*][:CHANnel[*m*]]:OFFSet? |
| description: | Returns the number of incoming triggers received before data logging begins. |
| parameters: | none |
| response: | an **integer** value. |
| example: | trig1:offs? -> 5<END> |
| affects: | All 81636B and 81637B series power meter modules. |

| | | |
|---|---|---|
| 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:<br>AVGover:<br>MEASure:<br>MODulation:<br>STFinished:<br>SWFinished:<br>SWSTarted: | Never.<br>When averaging time period finishes.<br>When averaging time period begins.<br>For every leading edge of a digitally-modulated (TTL) signal<br>When a sweep step finishes.<br>When sweep cycle finishes.<br>When a sweep cycle starts. |
| response: | none | |
| example: | trig1:outp dis | |
| affects: | All tunable laser modules (except 81950A), series power meters, and return loss modules. | |
| dual sensors: | Can only be sent to master channel, slave channel is also affected. | |
| | In continuous mode, wav:swe:step:[widt] is used for triggering, see [:SOURce[n]][:CHANnel[m]]:WAVelength:SWEep:STEP:[WIDTh] on page 179. | |

| | | |
|---|---|---|
| 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:<br>AVGover:<br>MEASure:<br>MODulation:<br>STFinished:<br>SWFinished:<br>SWSTarted: | Never.<br>When averaging time period finishes.<br>When averaging time period begins.<br>For every leading edge of a digitally-modulated (TTL) signal<br>When a sweep step finishes.<br>When sweep cycle finishes.<br>When a sweep cycle starts. |
| example: | trig1:outp? -> dis<END> | |
| affects: | All tunable laser modules (except 81950A), power meters, and return loss modules. | |
| 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. |
| | See Table 9 on page -209, 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<br>ON or 1: trigger rearming enabled (default) |
| | If you return to Local control, all modules return to the default setting. |
| response: | none |
| example: | trig1:outp:rearm 1 |
| affects: | All power meters, and return loss modules. |
| 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)<br>1: trigger rearming enabled. |
| example: | trig1:outp:rearm? -> 0<END> |
| affects: | All power meters, and return loss modules. |
| dual sensors: | Can only be sent to master channel, slave channel parameters are identical. |

| | |
|---|---|
| command: | **:TRIGger:CONF** |
| syntax: | :TRIGger:CONF<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 210 determines how each slot responds to an incoming trigger, all slot events (see :TRIGger[n][:CHANnel[m]]:OUTPut on page 213) 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:CONF?** | |
| syntax: | :TRIGger:CONF? | |
| 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 210 determines how each slot responds to an incoming trigger, all slot events (see :TRIGger[n][:CHANnel[m]]:OUTPut on page 213) 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:CONF:EXTended on page 217 or the Keysight 816x VXI*plug&play* Instrument Driver. |
| example: | trig:conf? -> DEF<END> | |

| | |
|---|---|
| command: | **:TRIGger:CONF:FPEDal** |
| syntax: | :TRIGger:CONF: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)<br>ON or 1: foot pedal enabled |
| response: | none | |
| example: | trig:conf? -> DEF<END> | |

| | | |
|---|---|---|
| command: | **:TRIGger:CONF:FPEDal?** | |
| syntax: | :TRIGger:CONF: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<br>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.

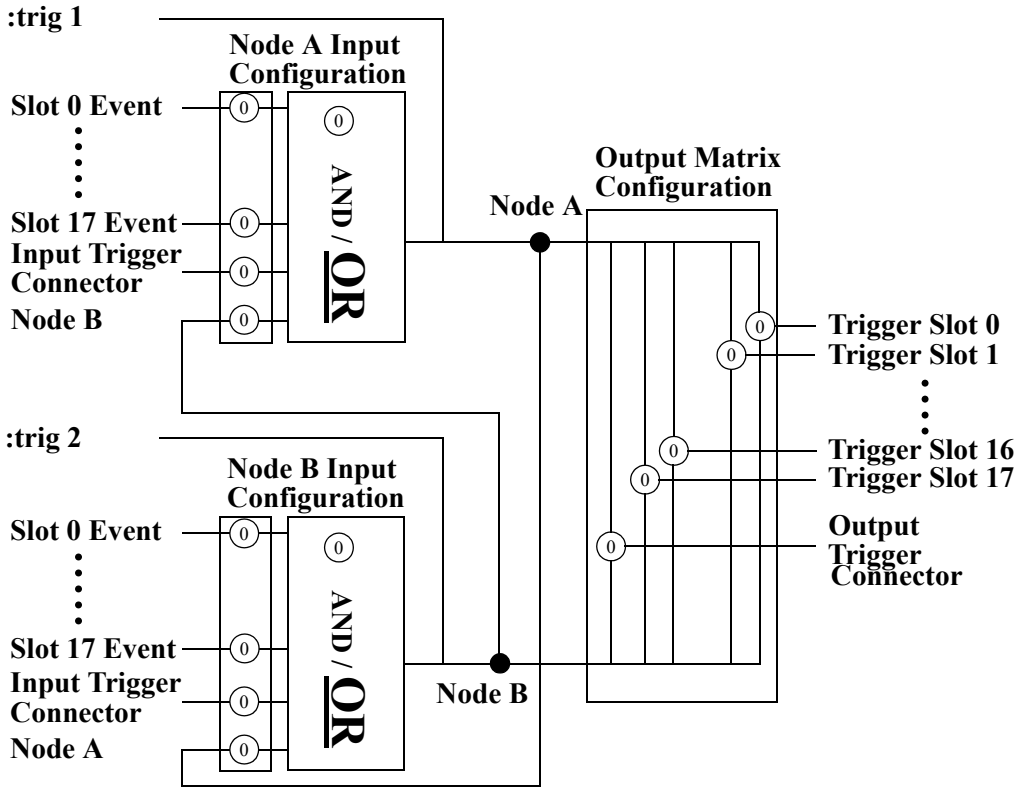
You can configure the ouputs and inputs from two nodes, Node A and Node B. See <Cross Reference Color>Figure 7on page 219 for more information on Node A and Node B. You can configure these nodes to be triggered by certain events and for these nodes to trigger particular actions.

| | | |
|---|---|---|
| command: | **:TRIGger** | |
| syntax: | :TRIGger<wsp>NODEA\|1\|NODEB\|2 | |
| description: | Generates a hardware trigger. | |
| parameters: | 1 or NODEA:<br>2 or NODEB: | Generates trigger at Node A.<br>Generates trigger at Node B. |
| | Use :TRIGger:CONF:EXTended on page 217 to configure Node A and Node B. | |
| | :TRIGger on page 209 describes the :TRIGger command for basic users. | |
| response: | none | |
| example: | trig 1 | |

| | | |
|---|---|---|
| command: | **:TRIGger:CONF:EXTended** | |
| syntax: | :TRIGger:CONF:EXTended<wsp><Node A Input Config.>,<br><Node B Input Config.>,<Output Matrix Config.> | |
| description: | Sets the extended hardware trigger configuration. | |
| parameters: | Node A Input Configuration:<br>Node B Input Configuration:<br>Output Matrix Configuration: | A **32-bit unsigned integer**, see below.<br>A **32-bit unsigned integer**, see below.<br>A **32-bit unsigned integer**, see below. |
| response: | none | |
| example: | trig:conf:ext 0,0,0 | |

| | |
|---|---|
| command: | **:TRIGger:CONF:EXTended?** |
| syntax: | :TRIGger:CONF:EXTended? |
| description: | Returns the extended hardware trigger configuration. |

| | | |
|---|---|---|
| parameters: | none | |
| response: | Node A Input Configuration:<br>Node B Input Configuration:<br>Output Matrix Configuration: | A 32-bit signed integer, see below.<br>A 32-bit signed integer, see below.<br>A 32-bit signed 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 7 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 | #**H**80000000 |
| 30 | Input Trigger Connector: 0 - Inactive, 1 - Trigger at Input Trigger Connector can trigger Node A<br>Node B: 0 - Inactive, 1 - Trigger at Node B can trigger Node A | #H40000000 |
| 29 | Not used. | #H20000000 |
| 18-2 | Slot 17: 0 - Inactive, 1 - Event at slot 17 can trigger Node A | 0 |
| 8 | Slot 16: 0 - Inactive, 1 - Event at slot 16 can trigger Node A | #H20000 |
| 17 | • | #H10000 |
| 16 | • | |
| | • | |
| | • | |
| | Slot 2: 0 - Inactive, 1 - Event at slot 2 can trigger Node A | #H4 |
| 2 | Slot 1: 0 - Inactive, 1 - Event at slot 1 can trigger Node A | #H2 |
| 1 | Slot 0: 0 - Inactive, 1 - Event at slot 0 can trigger Node A | #H1 |
| 0 | :TRIGger[n][:CHANnel[m]]:OUTPut on page 213 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 | #**H**80000000 |
| 30 | Input Trigger Connector: 0 - Inactive, 1 - Trigger at Input Trigger Connector can trigger Node B<br>Node A: 0 - Inactive, 1 - Trigger at Node A can trigger Node B | #H40000000 |
| 29 | Not used. | #H20000000 |
| 18-2 | Slot 17: 0 - Inactive, 1 - Event at slot 17 can trigger Node B | 0 |
| 8 | Slot 16: 0 - Inactive, 1 - Event at slot 16 can trigger Node B | #H20000 |
| 17 | | #H10000 |
| 16 | | |
| | Slot 2: 0 - Inactive, 1 - Event at slot 2 can trigger Node B • | |
| | Slot 1: 0 - Inactive, 1 - Event at slot 1 can trigger Node B • | #H4 |
| 2 | Slot 0: 0 - Inactive, 1 - Event at slot 0 can trigger Node B • | #H2 |
| 1 | :TRIGger[n][:CHANnel[m]]:OUTPut on page 213 explains how slot events can generate triggers. | #H1 |
| 0 | | |

| | Output Matrix Configuration<br>This **32-bit unsigned integer** lets you choose Node A OR Node B to trigger each of the following:<br>the Output Trigger Connector or<br>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 |
| • | | |
| • | | |
| • | Slot 2: 0 - Node A triggers slot 2, 1 - Node B triggers slot 2 | |
| 2 | Slot 1: 0 - Node A triggers slot 1, 1 - Node B triggers slot 1 | #H4 |
| 1 | Slot 0: 0 - Node A triggers slot 0, 1 - Node B triggers slot 0 | #H2 |
| 0 | :TRIGger[n][:CHANnel[m]]:INPut on page 210 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 Keysight 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*.

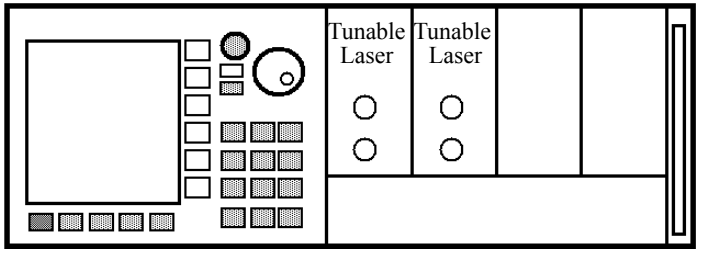



Figure 8 Setup for Extended Trigger Configuration Example

```
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.

# 5 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.

### :DISPlay:CONTrast

| | |
|---|---|
| command: | **:DISPlay:CONTrast** |
| syntax: | :DISPlay:CONTrast<wsp><value> |
| description: | Controls the contrast of the display. |
| parameters: | An **integer** value in the range 0 to 100 |
| response: | none |
| example: | disp:cont 50 |
| affects: | 8163B Lightwave Multimeter and 8166B Lightwave Multichannel System |

### :DISPlay:CONTrast?

| | |
|---|---|
| command: | **:DISPlay:CONTrast?** |
| syntax: | :DISPlay:CONTrast? |
| description: | Queries the contrast of the display. |
| parameters: | none |
| response: | An **integer** value in the range 0 to 100 |
| example: | disp:cont? -> +50<END> |
| affects: | 8163B Lightwave Multimeter and 8166B Lightwave Multichannel System |

## :DISPlay:BRIGhtness

| | |
|---|---|
| command: | **:DISPlay:BRIGhtness** |
| syntax: | :DISPlay:BRIGhtness<wsp><value> |
| description: | Controls the brightness of the display. |
| parameters: | An **integer** value in the range 0 to 100 |
| response: | none |
| example: | disp:brig 75 |
| affects: | 8163B Lightwave Multimeter and 8166B Lightwave Multichannel System -<br>8164A Lightwave Measurement System: only checks if the value equals 0. (0 -> display off, other values: display on) |

## :DISPlay:BRIGhtness?

| | |
|---|---|
| command: | **:DISPlay:BRIGhtness?** |
| syntax: | :DISPlay:BRIGhtness? |
| description: | Queries the brightness of the display. |
| parameters: | none |
| response: | An **integer** value in the range 0 to 100 |
| example: | disp:brig? -> +75<END> |
| affects: | 8163B Lightwave Multimeter and 8166B Lightwave Multichannel System -<br>8164A Lightwave Measurement System: only checks if the value equals 0. (0 -> display off, other values: display on) |

### :DISPlay:ENABle

| | | |
|---|---|---|
| command: | **:DISPlay:ENABle** | |
| syntax: | :DISPlay:ENABle<wsp>ON\|OFF\|1\|0 | |
| description: | Enables or disables the display.<br>The display is cleared, and an appropriate message displayed. This setting may improve sweep performance. | |
| parameters: | A *boolean* value: | OFF or boolean 0 – switch off the display<br>ON or boolean 1 – switch on the display |
| | If you press [LOCAL] softkey, the display is enabled automatically. | |
| response: | none | |
| example: | disp:enab 1 | |

### :DISPlay:ENABle?

| | | |
|---|---|---|
| command: | **:DISPlay:ENABle?** | |
| syntax: | :DISPlay:ENABle? | |
| description: | Queries the state of the display. | |
| parameters: | none | |
| response: | A *boolean* value: | 0 – the display is turned off<br>1 – the display is turned on |
| example: | disp:enab? -> 1<END> | |

### :DISPlay:LOCKout

| | |
|---|---|
| command: | **:DISPlay:LOCKout** |
| syntax: | :DISPlay:LOCKout<wsp>ON\|OFF\|1\|0 |
| description: | Enables or Disables local operation. |

| | | |
|---|---|---|
| parameters: | A *boolean* value: | OFF or boolean 0 – local operation is disabled<br>ON or boolean 1 – local operation is enabled. |
| response: | none | |
| example: | disp:lock 1<END> | |

### :DISPlay:LOCKout?

| | | |
|---|---|---|
| command: | **:DISPlay:LOCKout?** | |
| syntax: | :DISPlay:LOCKout? | |
| description: | Queries whether local operation is locked out. | |
| parameters: | none | |
| response: | A *boolean* value: | 0 – local operation is disabled<br>1 – local operation is enabled. |
| example: | disp:lock -> 1<END> | |

# 6 VISA 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 header files visa.h and visatype.h are included.
- Make sure the library path includes visa32.lib
- Ensure 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.

**NOTE** **Never use VISA calls and the Keysight 816x VXI*plug&play* Instrument Driver in the same program.**

**TIP:** Additional programming examples are provided on the Support Disk CD-ROM 08164-90BC4

## 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 GPIB 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 GPIB 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 GPIB 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 GPIB 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 able 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 from
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,replyBuf);
        /* 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 binary data */
    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(ViReal32));
      }

    /* stop logging */
    viPrintf(vi,"SENS%1d:CHAN%1d:FUNC:STAT LOGG,STOP\
n",slot,chan);
    checkError(vi,errStatus);

    /* display the values using %g, a float format specifier,
you may also use %e or %f */
    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) ;
          if
          ((!strncmp(errMsg,
         "-303,\"Module slot empty or slot / channel invalid\
"",
          strlen("-303,\"Module slot empty or slot / channel
invalid\"")))
          ||
```

```
          (!strncmp(errMsg,
          "-301,\"Module doesn't support this command
(StatCmdUnknown)\"",
          strlen(
          "-301,\"Module doesn't support this command
(StatCmdUnknown)\"")))
           {
             printf("No power meter in slot 1 so exiting\n\
n");
                           exit(1);
            }
          }
         }
  }

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 GPIB 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;
  }
```

# 7 The Keysight 816x VXIplug&play Instrument Driver



This chapter gives you extra information about installing and getting started with the Keysight 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 Keysight 816x Instrument Driver

The Keysight 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. The archive can be downloaded and installed using the N7700A Package Manager, or downloaded directly from the Keysight website or CD.

Note that VISA should first be installed on the PC, for example using the Keysight IO Libraries Suite.

1 Run hp816x.exe,
The welcome screen for the InstallShield Wizard used to install the Keysight 816x VXI*plug&play* Instrument Driver is displayed.

2 Press **Next**> to continue.
Specify the folder to which files will be saved.

3 Press **Next**> to continue.
Files are copied and extracted.
If necessary, a dialog requests your premission to overwrite existing files.

## Using Visual Programming Environments

### Getting Started with Keysight VEE

Keysight Technologies Visual Engineering Environment (Keysight VEE) is a visual programming language optimized for instrument control applications. To develop programs in Keysight VEE, you connect graphical 'objects' instead of writing lines of code. These programs resemble easy-to-understand block diagrams with lines.

Keysight VEE allows you to leverage your investment in textual languages by integrating with languages such as C, C++, Visual Basic, FORTRAN, and Pascal.

Keysight VEE controls GPIB, VXI, Serial, PC Plug-in, USB, and LAN instruments directly over the interfaces or by using instrument drivers.

**NOTE** Keysight VEE automatically calls the *initialize* and *close* functions to perform automatic error checking.

### GPIB Interfacing in Keysight VEE

Keysight 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 I/O menu.

2 Double-click on the Add button to bring up the Device Configuration screen, see Figure 9 on page -258.

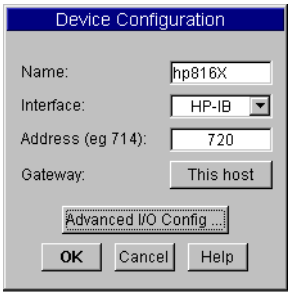



Figure 9 Device Configuration

3 Enter the following information:

- **Name**: enter hp816X.
- **Interface**: GPIB
- **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 10 on page -259 appears.

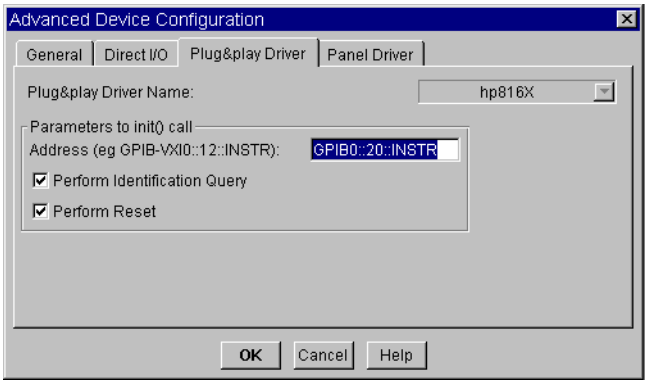



Figure 10 Advanced Device Configuration - Plug&play Driver

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.**

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 Keysight VEE opens the instrument for interaction.

8 Confirm the selections pressing the OK button.

9 Return to the Instrument Manager screen and press the Save Config to save the configuration.

## Getting Started with LabView

The 32-bit Keysight 816x driver can be used with LabView 5.0 and above.

Besides installing the Keysight 816x instrument driver, wrapper files need to be generated or installed for the driver. LabVIEW supports use of the VXI Plug&Play style drivers with wrapper files that can be generated for the corresponding driver and LabVIEW versions with the Instrument Driver Import Wizard utility, available from National Instruments.

It may be necessary to choose options the same as displayed in the figure below:

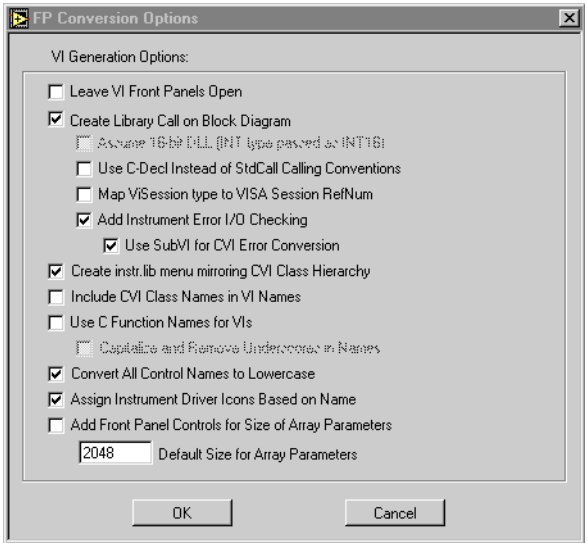



Figure 11 FP Conversion Options Box

**NOTE** You must check the **Add Front Panel Controls for Size of Array Parameters** box. There will be a front panel control created for each VI that requires you to assign the array size.

**NOTE** You must use the 32-bit version of Labview with the Keysight 816x VXI*plug&play* Instrument Driver.

**NOTE** LabView is a trademark of National Instruments Corporation.

## Getting Started with LabWindows

The 32-bit Keysight 816x VXI*plug&play* Instrument 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 Keysight 816x VXI*plug&play* Instrument 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 Keysight 816x Instrument Driver

The Keysight 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 Keysight 816x VXI*plug&play* Instrument Driver conforms with the VXI*plug&play* standard.
- There is one exception as the Keysight 816x driver does not have a soft front panel or a knowledge-based file.
- The Keysight 816x VXI*plug&play* Instrument Driver is built on top of VISA, and uses the services provided.
- VISA supports GPIB and VXI protocols. The driver can be used with any GPIB card for which the manufacturer has provided a VISA DLL.
- The Keysight 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 Keysight VEE, LabWindows, and LabView.
- The Keysight 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 Keysight 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 Keysight 816x instrument driver usually uses the directory C:\Program Files (x86)\IVI Foundation , which is used by the VISA installation. The structures for the IVI Foundation subdirectory tree are shown in Figure 12 on page -264.

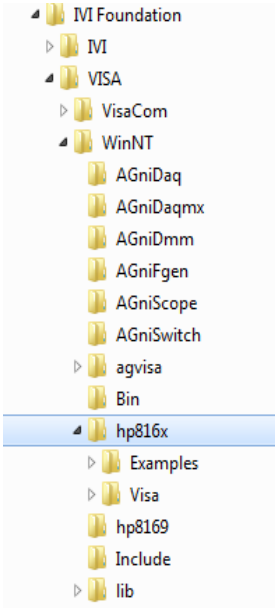



Figure 12 Windows VXIPNP Directory Structure

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 Keysight 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 Figure 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_mes-
sage);

    /* 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_mes-
sage);

    }
    /* 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 VISA Programming Examples on page 229.

## 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 Keysight 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 C:\Program Files (x86)\IVI Foundation\VISA\WinNT\Include.

You may also need to include the file visa.bas. visa.bas is provided with your VISA DLL.

### Keysight VEE

The online Help installed with VEE contains detailed information for using VXI Plug&Play drivers.

### LabWindows CVI/ (R) 4.0 (or higher)

The Keysight 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 Keysight Technologies,
- 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 C:\Program Files (x86)\IVI Foundation\VISA\WinNT\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 C:\Program Files (x86)\IVI Foundation\VISA\WinNT\Include is added to the include paths (CVI Project Option menu).

## Online Information

The latest copy of this driver can be downloaded via:

www.keysight.com/find/octfirmware,
or using the N7700A Package Manager from www.keysight.com/find/n7700

If you do not have web access, the 816x driver package and N7700A Package Manager are available on your OCT Support CD, or contact your Keysight Technologies supplier.

## Lambda Scan Applications

These functions combine multiple SCPI commands into a single, functional operation. They are designed to allow quick and easy access to common instrument command sequences.

These application functions allow you to perform one of the following applications:

- A Lambda Scan - a measurement operation where a Keysight Lightwave mainframe with a continuous sweep tunable laser module and up to four power meters installed, performs a wavelength sweep while the power meters are synchronized to record optical power.
- A Multi Frame Lambda Scan - a measurement operation where a Keysight Lightwave mainframe with a continuous-sweep tunable laser module performs a wavelength sweep in coordination with power meters that are installed in the same mainframe or in other instruments. These additional instruments must be connected to the PC and have their input rigger connector connected to the output trigger connector of the laser mainframe. The Multiframe LambdaScan also provides more flexibility in configuring the measurement parameters. The N774x-series multi-port power meters are also supported..

The following two functions apply to both Lambda Scan and Multi Frame Lambda Scan applications:

- The Set Lambda Scan Wavelength (hp816x_set_LambdaScan_wavelength) function allows you to use a different wavelength than 1550 nm during a Lambda Scan operation. All Power Meters taking part in the Lambda Scan operation will be set to the chosen wavelength.
- The Enable High Sweep Speed (hp816x_enableHighSweepSpeed) function was used with older model lasers and enables/disables the highest available sweep speed (40 nanometers per second) for Lambda Scan operations. The Lambda Scan operation chooses the highest possible sweep speed for the chosen step size.
  - If you choose Enable, the highest sweep speed possible will be used. This may lead to less accurate measurements.
  - If you choose Disable, the highest sweep speed will not be used.

### Equally Spaced Datapoints

A linear interpolation is performed on all wavelength and power data for the Lambda Scan Application and is optional for the Multi Frame Lambda Scan Application.

The advantage of spacing all measurements equally is that presenting results through use of a spreadsheet is greatly simplified. The operation returns one wavelength array and a power array for each power meter channel.

The disadvantage of using equally spaced datapoints is that the linear interpolation is analogous to the use of a low pass filter. Figure 13 on page -274shows the original curve as measured directly by a Power Meter and the interpolated curve.

Interpolation will always tend to produce a smoother curve by rounding off any peaks in the curve.


Figure 13 Equally Spaced Datapoints

How to Perform a Lambda Scan Application

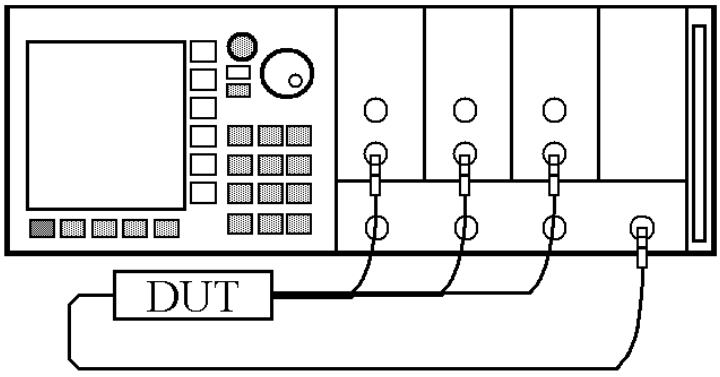



Figure 14 Lambda Scan Operation Setup

### The Prepare Lambda Scan Function

The Prepare Lambda Scan (hp816x_prepareLambdaScan) function prepares a Lambda Scan operation.

The Prepare Lambda Scan (hp816x_prepareLambdaScan) function must be called before a Lambda Scan operation is executed. Use the return values of this function (Number of Datapoints and Number of Power Arrays) to allocate arrays for the Execute Lambda Scan (hp816x_executeLambdaScan) function.

To obtain a higher precision, the Tunable Laser Source is set 1 nm before the Start Wavelength, this means, you have to choose a Start Wavelength 1 nm greater than the minimum possible wavelength. Also, the wavelength sweep is actually started 90 pm before the Start Wavelength and ends 90 pm after the Stop Wavelength, this means, you have to choose a Stop Wavelength 90 pm less than the maximum possible wavelength.

Triggers coordinate the Tunable Laser module with all Power Meters. The function sets for the lowest possible averaging time available for the installed Power Meters and, then, sets the highest possible sweep speed for the selected Tunable Laser module sweep.

If one of the following circumstances occurs, the "parameter mismatch" error will be returned:

1 If one Power Meter is out of the specification at 1550 nm, the error "powermeter wavelength does not span 1550nm" will be returned. For example, the HP 81530A Power Sensor and the HP 81520A Optical Head are out of specification at 1550 nm. Remove the Power Meter that is out of specification at 1550 nm from the mainframe.

2 If the Step Size is too small and results in a trigger frequency that is to high for the installed Power Meters, the error "could not calculate a sweep speed!" will be returned. Increase the Step Size.

3 If the chosen wavelength range is too large and Step Size is too small, the error "too many datapoints to log!" will be returned. In this case, reduce the wavelength range and/or increase the Step Size.

### The Get Lambda Scan Parameters Function

The Get Lambda Scan Parameters (hp816x_getLambdaScanParameters_Q) function returns all parameters that the Prepare Lambda Scan (hp816x_prepareLambdaScan) function adjusts or automatically calculates.

### The Execute Lambda Scan Function

The Execute Lambda Scan (hp816x_executeLambdaScan) function runs and returns the results of a Lambda Scan operation.

That is, it executes an operation where a Keysight Lightwave mainframe with a continuous-sweep tunable laser module and up to four power sensors installed, performs a wavelength sweep.

The Prepare Lambda Scan (hp816x_prepareLambdaScan) function must be called before a Lambda Scan operation is executed. Use the return values of this function (Number of Datapoints and Number of Power Arrays) to allocate arrays for the Execute Lambda Scan (hp816x_executeLambdaScan) function.

Equally Spaced Datapoints is enabled as part of this function and cannot be disabled. Use Multi Frame Lambda Scan if you need to have inequally spaced datapoints. See Equally Spaced Datapoints on page 273 for more details.

How to Perform a Multi-Frame Lambda Scan Application

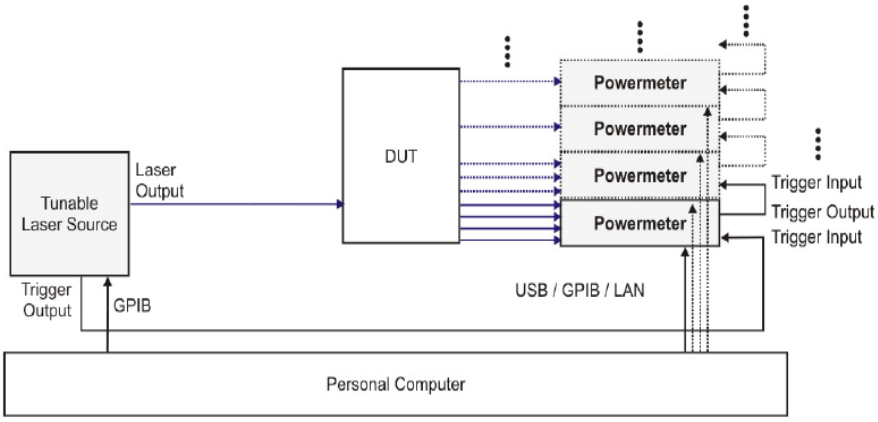



Figure 15 Multi Frame Lambda Scan Operation Setup

### The Equally Spaced Datapoints Function

The Equally Spaced Datapoints (hp816x_returnEquidistantData) function allows you to select whether you the results will be equally spaced by performing a linear interpolation on the wavelength point and power measurement data, see Equally Spaced Datapoints on page 273 for more details.

This function is used because Lambda Scan functions make use of Lambda Logging to log the exact wavelength that measurements were triggered at. This results in Lambda Array wavelength points that are not equally spaced.

**NOTE** **Lambda Logging is not available if your Tunable Laser module firmware revision is lower than 2.0.**

Equally Spaced Datapoints is enabled as a default.

### The Register Mainframe Function

Use the Register Mainframe (hp816x_registerMainframe) function to register your mainframe as a participant in a Multi Frame Lambda Scan operation. The mainframe must be connected to the GPIB bus and have their Input Trigger Connector connected to the Output Trigger Connector of the Keysight Lightwave mainframe that the tunable laser module is installed in the N774x-series multiport power meters are also supported.

### The Unregister Mainframe Function

Use the Unregister Mainframe function (hp816x_unregisterMainframe) to remove a mainframe from a Multi Frame Lambda Scan operation and clear the driver's internal data structures.

If you use LabView the following items should be noted:

- All multi frame functions are not re-entrant, if the driver is running and initialized more than once, results may be unpredictable.
- To avoid wrong results, call the Unregister Mainframe function prior to the Initialize function (hp816x_init). This is especially necessary during program debugging, if the Close function (hp816x_close) is not called.

### The Prepare Multi Frame Lambda Scan Function

The Prepare Multi Frame Lambda Scan (hp816x_prepareMfLambdaScan) function prepares a Lambda Scan operation for multiple Mainframes.

That is, it prepares an operation where a Keysight Lightwave mainframe with a continuous-sweep tunable laser module and up to 100 power meter channels located in different instruments are installed.

The Prepare Multi Frame Lambda Scan (hp816x_prepareMfLambdaScan) function must be called before a Multi Frame Lambda Scan is executed. Use the return values of this function (Number of Datapoints and Number of Power Arrays) to allocate arrays for the Execute Multi Frame Lambda Scan (hp816x_executeMfLambdaScan) function.

The function scans all mainframes to find laser sources. The function scans each mainframe in the order that they were originally registered by the Register Mainframe function (hp816x_registerMainframe). The first continuous-sweep tunable laser found will perform the sweep operation.If more than one tunable laser is installed, only the laser with the lower slot number will be used.

The wavelength sweep is actually started 50 pm before the Start Wavelength and ends 50 pm after the Stop Wavelength, this means, you have to choose a Stop Wavelength to assure that the full chosen wavelength range is covered. If the step is larger than 50 pm, then this larger margin is used instead.

Triggers coordinate the Tunable Laser module with all Power Meters. The function sets for the lowest possible averaging time available for the installed Power Meters and, then, sets the highest possible sweep speed for the selected Tunable Laser module sweep. All mainframes must be connected to the GPIB bus and have their Input Trigger Connector connected to the Output Trigger Connector of the mainframe that the Tunable Laser module is installed in.

If one of the following circumstances occurs, the "parameter mismatch" error will be returned:

1 If one Power Meter is out of the specification at 1550 nm, the error "powermeter wavelength does not span 1550nm" will be returned. For example, the HP 81530A Power Sensor and the HP 81520A Optical Head are out of specification at 1550 nm. Remove the Power Meter that is out of specification at 1550 nm from the mainframe.

2 If the Step Size is too small and results in a trigger frequency that is to high for the installed Power Meters, the error "could not calculate a sweep speed!" will be returned. Increase the Step Size.

3 If the chosen wavelength range is too large and Step Size is too small, the error "too many datapoints to log!" will be returned. In this case, reduce the wavelength range and/or increase the Step Size.

### The Get MF Lambda Scan Parameters Function

The Get MF Lambda Scan Parameters (hp816x_getMFLambdaScanParameters_Q) function returns all parameters that the Prepare Multi Frame Lambda Scan (hp816x_prepareMfLambdaScan) function adjusts or automatically calculates.

### The Execute Multi Frame Lambda Scan Function

The Execute Multi Frame Lambda Scan (hp816x_executeMfLambdaScan) function runs a Lambda Scan operation and returns an array that contains the wavelength values at which power measurements are made.

Use the values returned from the Prepare Multi Frame Lambda Scan (hp816x_prepareMfLambdaScan) function to set the parameters of the Execute Multi Frame Lambda Scan (hp816x_executeMfLambdaScan) function.

### The Get Lambda Scan Result Function

The Get Lambda Scan Result (hp816x_getLambdaScanResult) function returns for a given Power Meter channel a power value array and a wavelength value array.

These arrays contains the results of the last Multi Frame Lambda Scan operation.

### The Get Number of PWM Channels Function

The Get Number of PWM Channels (hp816x_getNoOfRegPWMChannnels_Q) function returns the number of Power Meter channels in a test setup.

Only Power Meters whose mainframe was registered using the Register Mainframe (hp816x_registerMainframe) function are counted.

### The Get Channel Location Function

The Get Channel Location function (hp816x_getChannelLocation_Q) returns the location of the chosen Power Meter channel as used in a Multi Frame Lambda Scan operation.

The maximum number of channels that may be specified is 1000.

# 8 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.

**Table 10 Incompatible GPIB Bus Command**

| Command | Change | Affects |
|---|---|---|
| LLO - local lockout | | Both |
| DCL - device clear | | Both |
| GET - group execute trigger | | Both |

S

## 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 11 on page -283 contains details of commands that have been removed without replacement.

**Table 11 Removed Commands**

| 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:<br>NTRansition/?<br>STATus:OPERation:<br>PTRansition/?<br>STATus:QUEStionable:<br>NTRansition/?<br>STATus:QUEStionable:<br>PTRansition/? | These status model features are not supported. | 8153 |
| SYSTem:BEEPer:STATe/? | Beeper access is not supplied. | 8153 |
| *SAV<br>*RCL | User interface or GPIB settings cannot be stored or recalled. | 8167/8 |
| BDATA?<br>DOSMODE/? | Memory card access is not provided. | 8167/8 |
| TRACe:CATalog?<br>TRACE:DATA?<br>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 |

## Obsolete Commands

Table 12 on page -284 contains details of commands that have been directly replaced.

**Table 12 Obsolete Commands**

| Old Command | New Command | Affects |
|---|---|---|
| DISPlay:STATe/? | DISPlay:ENABle/? | 8153 |
| PROGram command tree | SENSe:FUNCtion command tree.<br>Some commands from the PROGram command tree have not been replaced.<br>The HP 8153A application interface on the GPIB is not supported.<br>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 |

## Changed Parameter Syntax and Semantics

Table 13 on page -285 details commands whose parameter syntax or semantics have changed.

**Table 13 Commands with Different Parameters or Syntax**

| 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.<br>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 |

## Changed Query Result Values

Table 14 on page -286 details queries that respond with different return codes than the old instruments.

**Table 14 Queries with Different Result Values**

| Command | Change | Affects |
|---|---|---|
| *IDN? | Returns new instrument and module identifiers. | Both |
| *OPT? | Returns new module options. | Both |
| *TST | Selftest result codes are completely new.<br>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 |

## Timing Behavior

Table 11 on page -283 details the ways in which timing behavior is different.

**Table 15 Timing Behavior Changes**

| Change | Affects |
|---|---|
| Command execution may be different. | Both |
| In all mainframes and modules firmware revisions before 3.x 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.<br>In later firmware revisions GPIB only blocks if a command can't be processed because of a pending command. While a command is pending *OPC? returns 0 now. This is the behaviour of the 8153. | Both |
| 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 |

## 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 8163A/B and 8164A/B; instead of timing out for every error, special values are returned for erroneous queries. Table 16 on page -288 and Table 17 on page -288 detail the new errors

The error queue is written to as before.

**Table 16 Error Handling Changes**

| Expected Return Value | Returned Value | Affects |
|---|---|---|
| FLOAT(32/64) | FLT/DBL_MAX | 8153 |
| (U)INT(16/32) | (U)INT(16/32)_MAX | 8153 |
| Block | " " | 8153 |
| Boolean Value | 0 | 8153 |
| Enum | Time out | 8153 |

**Table 17 Specific Errors**

| Command | Change | Affects |
|---|---|---|
| FETCh:POWer? - without using a preceding trigger | Returns the last valid value instead of timing out. No error is generated. | 8153 |

## Command Order

It is not yet known if there are any changes in the command order behavior.

# 9 Error Codes



This chapter gives information about error codes used with the 8163A/B Lightwave Multimeter, the 8164A/B Lightwave Measurement System, and the 8166A/B Lightwave Multichannel System.

# GPIB Error Strings

Error strings in the range -100 to -183 are defined by the SCPI standard, downloadable from: http://www.ivifoundation.org/docs/scpi-99.pdf

String descriptions taken from this standard (VERSION 1999.0 May, 1999), whether in whole or in part, are enclosed by [ ].

**Table 18 Overview for Supported Strings**

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Note: | Error strings in the range -100 to -183 are defined by the SCPI standard, downloadable from: http://www.ivifoundation.org/docs/scpi-99.pdf<br>String descriptions taken from this standard (VERSION 1999.0 May, 1999), whether in whole or in part, are enclosed by [ ] in this table. | |
| -100 to -199 Command Errors | | |
| Standard | -100 | "Command Error"<br>[This is the generic syntax error used when a more specific error cannot be detected. This code indicates only that a Command Error as defined in *IEEE 488.*2,11.5.1.1.4 has occurred.] |
| Standard | -101 | "Invalid character"<br>[A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of error -114 and perhaps some others.] |
| Standard | -102 | "Syntax error"<br>[An unrecognized command or data type was encountered; for example, a string was received when the **device** does not accept strings.] |
| Standard | -103 | "Invalid separator"<br>[The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit] |
| Standard | -104 | "Data type error"<br>[The parser recognized a data element different than one allowed; for example,numeric or string data was expected but block data was encountered.] |
| Standard | -105 | "GET not allowed"<br>[A Group Execute Trigger was received within a program message (see *IEEE488.*2, 7.7).] |
| Standard | -108 | "Parameter not allowed"<br>[More parameters were received than expected for the header] |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Standard | -109 | "Missing parameter"<br>[Fewer parameters were recieved than required for the header] |
| Standard | -112 | "Program mnemonic too long"<br>[The header contains more than twelve characters (see *IEEE 488.*2, 7.6.1.4.1).] |
| Standard | -113 | "Undefined header"<br>[The header is syntactically correct, but it is undefined for this specific **devic**e; for example, *XYZ is not defined for any device.] |
| Standard | -120 | "Numeric data error"<br>[This error, as well as errors -121 through -129, are generated when parsing a data element which appears to be numeric, including the nondecimal numeric types. This error message is used if the **device** cannot detect a more specific error.] |
| Standard | -121 | "Invalid character in number"<br>[An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric] |
| Standard | -123 | "Exponent too large"<br>[The magnitude of the exponent was larger than 32000 (see *IEEE 488.*2,7.7.2.4.1).] |
| Standard | -124 | "Too many digits"<br>[The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see *IEEE 488.*2, 7.7.2.4.1).] |
| Standard | -128 | "Numeric data not allowed"<br>[A legal numeric data element was received, but the **device** does not accept one in this position for the header.] |
| Standard | -131 | "Invalid suffix"<br>[The suffix does not follow the syntax described in *IEEE 488.*2, 7.7.3.2, or thesuffix is inappropriate for this **devic**e.] |
| Standard | -134 | “Suffix too long”<br>[The suffix contained more than 12 characters (see *IEEE 488.*2, 7.7.3.4).] |
| Standard | -138 | “Suffix not allowed”<br>[A suffix was encountered after a numeric element which does not allow suffixes.] |
| Standard | -141 | “Invalid character data”<br>[Either the character data element contains an invalid character or the particular element received is not valid for the header.] |
| Standard | -148 | “Character data not allowed”<br>[A legal character data element was encountered where prohibited by the **devic**e.] |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Standard | -150 | "String data error"<br>[This error, as well as errors -151 through -159, are generated when parsing a string data element. This error message is used when the **device** cannot detect a more specific error.] |
| Standard | -151 | "Invalid string data"<br>[A string data element was expected, but was invalid for some reason (see *IEEE 488.*2, 7.7.5.2); for example, an END message was received before the terminal quote character.] |
| Standard | -158 | "String data not allowed"<br>[A string data element was encountered but was not allowed by the **device** at this point in parsing.] |
| Standard | -161 | "Invalid block data"<br>[A block data element was expected, but was invalid for some reason (see *IEEE 488.*2, 7.7.6.2); for example, an END message was received before the length was satisfied.] |
| Standard | -168 | "Block data not allowed"<br>[A legal block data element was encountered but was not allowed by the **device** at this point in parsing.] |
| Standard | -170 | "Expression error"<br>[This error, as well as errors -171 through -179, are generated when parsing an expression data element. This particular error message is used when the **device** cannot detect a more specific error.] |
| Standard | -171 | "Invalid expression"<br>[The expression data element was invalid (see *IEEE 488.*2, 7.7.7.2); for example, unmatched parentheses or an illegal character.] |
| Standard | -178 | "Expression data not allowed"<br>[A legal expression data was encountered but was not allowed by the **device** at this point in parsing.] |
| Standard | -181 | "Invalid outside macro definition"<br>[Indicates that a macro parameter placeholder ($<number) was encountered outside of a macro definition.] |
| Standard | -183 | "Invalid inside macro definition"<br>[Indicates that the program message unit sequence, sent with a *DDT or *DMC command, is syntactically invalid (see *IEEE 488.*2, 10.7.6.3).] |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| New | -185 | “Subop out of range”<br>*Description:*<br>Suboperations are parameters that are passed to refine the destination of a command. They are used to address slots, channels, laser selections and GPIB/SCPI register levels. This error is generated if the parameter is not valid in the current context or system configuration.<br>*Example:*<br>This error occurs if the user queries the status of a summary register and passes an invalid status level (also see "Status for 816x" on page 28 programmer's guide).<br>*Note:*<br>Incorrect slots and channels addresses are handled by error code -301 |
| -200 to -299 Execution Errors | | |
| Standard | -200 | “Execution error (StatExecError)"<br>*Description:*<br>This error occurs when the current function, instrument or module state (or status) prevents the execution of a command. This is a generic error which can occur for a number of reasons.<br>*Example:*<br>When a powermeter has finished a logging application and data is available, the user is not able to reconfigure the logging application parameters. First, the user must stop the logging application. |
| New | -201 | "Please be patient - GPIB currently locked out"<br>*Description:*<br>Some operations block the complete system. Since no sensible measurements are possible while this is true, the GPIB is locked out.<br>*Example:*<br>When ARA, Lambda zeroing or zeroing is executing on a TLS module, the GPIB is not accessible. |
| New | -205 | “Powermeter not running (StatMeterNotRunning)"<br>*Description:*<br>Some command and actions may stop the data aquisition unit of a powermeter. If a command fetches data, there may be no measurement values and this error is generated. Please check module state and repeat operation. |
| Old | -211 | “Trigger ignored”<br>*Description:*<br>A trigger has been detected but ignored because of timing contraints. (For Example: average time to large). |
| Old | -212 | “Arm ignored”<br>*Description:*<br>The user can set the automatic re-arming option for input and output trigger events (see GPIB Error Strings on page 290). When this error occurs, the device ignores the setting because the current module status does not allow the change of trigger settings. |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Old | -213 | “Init ignored”<br>*Description:*<br>The INIT:IMM command (page 89) initiates a trigger and completes a full measurement cycle. The continuous measurement must be DISABLED. This error code is generated if the powermeter is still in cont. measurement mode. |
| Old | -220 | "Parameter error (StatParmError)”<br>*Description:*<br>The user has passed a parameter that cannot be changed in this way. The device cannot detect one of the following more specific errors: |
| Old | -220 | -220, "Parameter error (StatParmOutOfRange)"<br>*Description:*<br>The user has passed a parameter that exceeds the valid range for this parameter. |
| Old | -220 | "Parameter error (StatParmIllegalVal)"<br>*Description:*<br>The user has passed a parameter that does not match a value in a list of possible values. |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Old | -221 | "Settings conflict (StatParmInconsistent)"<br>*Description:*<br>The user has passed a parameter that conflicts with other already configured parameters.<br>*Example:*<br>There are constrains for TLS sweep parameters: this error is generated when lambda step size exceeds the difference between start and stop wavelength.<br>If error -221 is returned after you try to start a wavelength sweep, one of the following cases of sweep parameter inconsistency has occurred:<br>Continuous Sweep mode AND l Start is less than l Stop.<br>Continuous Sweep mode AND Sweep Time is too short. Adjust Sweep Speed, l Start, or l Stop.<br>Continuous Sweep mode AND Sweep Time is too long. Adjust Sweep Speed, l Start, or l Stop.<br>Continuous Sweep mode AND Trigger Frequency is too high. Adjust Step Size. Trigger Frequency is the Sweep Speed divided by the Step Size.<br>Stepped Sweep mode AND Lambda Logging Enabled.<br>Continuous Sweep mode AND Lambda Logging Enabled AND Output trigger mode not set to STFinished (Step finished).<br>Continuous Sweep mode AND Lambda Logging is Enabled AND Modulation Source is not set to OFF.<br>Continuous Sweep mode AND Lambda Logging is Enabled AND Sweep Cycles is not set to 1.<br>Continuous Sweep mode AND Coherence Control is Enabled.<br>"Not allowed while laser is on" -<br>*Description:*<br>The user tried to set grid spacing or reference frequency without switching off the laser.<br>"Not allowed while frequency auto mode is on"<br>*Description:*<br>The user tried to issue a command which is only supported in grid mode.<br>"Not allowed while frequency auto mode is off"<br>*Description:*<br>The user tried to issue a command which is only supported in auto mode.<br>"Settings conflict"<br>*Description:*<br>Returned if the laser module reports an invalid configuration |
| Standard | -222 | "Data out of range (StatParmTooLarge)"<br>*Description:*<br>The user has passed a continuous parameter that is too large.<br>*Example:*<br>Wavelength 1800nm when maximum wavelength is 1700nm. |
| Standard | -222 | "Data out of range (StatParmTooSmall)"<br>*Description:*<br>The user has passed a continuous parameter that is too small.<br>*Example:*<br>Wavelength 700nm when minimum wavelength is 800nm. |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Standard | -223 | “Too much data”<br>*Description:*<br>A function returns more data or the user requests more data than the application is able to handle.<br>*Example:*<br>A tunable laser source produces more data when lambda values of a sweep are stored than the 816x instrument is able to handle. Use the new SENSE:FUNC:RES:BLOCK? command to split the data aquisition into multiple parts. |
| Standard | -224 | “Illegal parameter value”<br>[Used where exact value, from a list of possibles, was expected.] |
| New | -225 | "Out of memory"<br>*Description:*<br>The request application or function cannot be executed because the instrument runs out of memory. |
| Old | -231 | "Data questionable (StatValNYetAcc)"<br>*Description:*<br>The data that is retured is not accurate or reliable. The user should repeat the operation. The reason for this error is unspecific.<br>*Example:*<br>A powermeter configured a long average time has not completed its current measurement cycle when the user queries the current power. |
| Old | -231 | "Data questionable (StatRangeTooLow)"<br>*Description:*<br>As -231 (StatValNYetAcc) but for a more specific reason: The powermeter readout data is not reliable because the currently set (manual) range does not correspond with the input power. |
| Old | -261 | "Math error in expression (StatUnitCalculationError)"<br>*Description:*<br>This may occur when the user attempts to transform data in a way that is currently not possible.<br>*Example:*<br>When a powermeter is measuring very small power values in dBm (such as noise power), negative power values in Watt may also be present (such as when the powermeter calibration wavelength does not correspond to the wavelength of input signal). The instrument cannot transform negative Watt values to dBm because the logarithm of a negative value is not defined. |
| Standard | -272 | “Macro execution error”<br>[Indicates that a syntactically legal macro program data sequence could not beexecuted due to some error in the macro definition (see IEEE 488.2, 10.7.6.3.)] |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Standard | -273 | “Illegal macro label”<br>[Indicates that the macro label defined in the *DMC command was a legal string syntax, but could not be accepted by the **device** (see *IEEE 488.*2, 10.7.3 and 10.7.6.2); for example, the label was too long, the same as a common command header, or contained invalid header syntax.] |
| Standard | -276 | “Macro recursion error”<br>[Indicates that a syntactically legal macro program data sequence could not be executed because the device found it to be recursive (see *IEEE 488.*2, 10.7.6.6).] |
| Standard | -277 | “Macro redefinition not allowed”<br>[Indicates that a syntactically legal macro label in the *DMC command could not be executed because the macro label was already defined (see *IEEE 488.*2,10.7.6.4).] |
| Standard | -278 | “Macro header not found”<br>[Indicates that a syntactically legal macro label in the *GMC? query could not be executed because the header was not previously defined.] |
| Old | -284 | "Function currently running (StatModuleBusy)"<br>*Description:*<br>This error is generated when a function is currently running on a module so that it cannot process another command.<br>*Example:*<br>When a powermeter is running a logging application, you are not able to configure the logging application parameters (also see -200). |
| Old | -286 | "No function currently running"<br>*Description:*<br>This error is generated when a user tries to execute a command which requires a particular set of data that is not available.<br>*Example:*<br>Application data is necessary to execute SENSE:FUNC:RES?. If no suitable function has completed, there is no data and this error is generated. (also see -200). |
| New | -290 | “Application currently running - no GPIB support”<br>*Description:*<br>The instrument has built-in applications that have no GPIB support ( such as Logging,Stability,PACT).<br>*Example*<br>When an application is running error -290 will be returned if any command other than one the following is sent:<br>*WAI<br>*OPC?<br>:SPECial:REBoot<br>:SYSTem:ERRor?<br>:SYSTem:VERSion? |
| -300 to -399 or between 1 and 32767 Device-Specific Errors (Module) | | |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Old | -300 | “Internal error (StatVals Lost)”<br>“Internal error (StatInternalError)”<br>*Description*<br>These are generic device-dependent errors used when the instrument cannot detect more specific errors. |
| New | -301 | "Module doesn't support this command (StatCmdUnknown)"<br>*Description:*<br>The addressed module does not support the SCPI command.<br>*Example:*<br>When a command from the SENSe SCPI tree is sent to a fixed or tunable laser source. |
| New | -302 | "Internal timeout error (StatTimedOut)"<br>*Description:*<br>A command has not returned in the expected time. |
| New | -303 | "Module slot empty or slot / channel invalid"<br>*Description:*<br>The user has send a command to an empty slot. |
| New | -304 | "Command was aborted (StatAborted)"<br>*Description:*<br>The command has been interrupted by another event. |
| New | -305 | "Internal messaging error (StatCmdError)"<br>"Internal messaging error (StatCmdNotAllowed)"<br>"Internal messaging error (StatWrongLength)"<br>"Internal messaging error (StatWrongReceiver)"<br>"Internal messaging error (StatBufAllocError)"<br>"Internal messaging error (StatDPRamFull)"; }<br>"Internal messaging error (StatSemError)"<br>*Description:*<br>An error has occured in the instrument communication system. Please report this error with a description of the circumstances that generated the error and the configuration of the system. |
| New | -306 | "Channel doesn't support this command (StatCmdUnknownForSlave)"<br>*Description:*<br>Slave channels have limited functionality. The module supports this command, but the command must be sent to the master channel. |
| New | -307 | "Channel without head connection (StatHeadless)"<br>*Description:*<br>The channel supports this command, but it cannot be executed because the optical measurement head is not plugged into the interface module. |
| Standard | -310 | “System error”<br>[Indicates that some error, termed “system error” by the device, has occurred. This code is device-dependent.] |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Standard | -321 | “Out of memory”<br>[An internal operation needed more memory than was available.] |
| New | -322 | "Flash programming error (StatFlashEraseFailed)"<br>"Flash programming error (StatFlashWriteFailed)"<br>"Flash programming error (StatFlashDataCntError)"<br>"Flash programming error (StatFlashDPAlgoFailed)"<br>*Description:*<br>An error has occured in a module. Please report this error with a description of the circumstances that generated the error and the configuration of the system. |
| New | -323 | "Flash programming error (StatUserCalTable Empty)"<br>It is not possible to activate the offset (λ) functionality when the offset table is empty<br>"Flash programming error (StatUserCalTable Full)"<br>The offset (λ) table is full and no more ? can be stored<br>"Flash programming error (StatUserCalActive)"<br>It is not possible to program the offset (λ) table when the offset (λ) feature is activated. Deactivate first. |
| Old | -330 | “Self-test failed”<br>*Description:*<br>You have started the self test, but the module has detected an error while executing it |
| New | -340 | "Printing error (StatPrintError)"<br>*Description:*<br>An unspecified problem occurred while communicating with the printer. |
| New | -341 | "Printing error - paper out (StatPaperOut)"<br>*Description:*<br>The instrument cannot print because there is no paper in the connected printer. |
| New | -342 | "Printing error - offline (StatOffline)"<br>*Description:*<br>The instrument cannot print because the connected printer is offline. |
| Standard | -350 | “Queue overflow”<br>[A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.] |
| -400 to -499 Query Errors | | |
| Standard | -400 | “Query error”<br>[This is the generic query error for **devices** that cannot detect more specific errors. This code indicates only that a Query Error as defined in *IEEE 488.*2, 11.5.1.1.7 and 6.3 has occurred.] |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Standard | -410 | "Query INTERRUPTED"<br>[Indicates that a condition causing an INTERRUPTED Query error occurred (see *IEEE 488.2*, 6.3.2.3); for example, a query followed by DAB or GET before a response was completely sent.] |
| Standard | -420 | "Query UNTERMINATED"<br>[Indicates that a condition causing an UNTERMINATED Query error occurred (see *IEEE 488.2*, 6.3.2.2); for example, the **device** was addressed to talk and an incomplete program message was received.] |
| Standard | -430 | "Query DEADLOCKED"<br>[Indicates that a condition causing an DEADLOCKED Query error occurred (see *IEEE 488.2*, 6.3.1.7); for example, both input buffer and output buffer are full and the device cannot continue.] |
| Standard | -440 | "Query UNTERMINATED after indef resp"<br>[Indicates that a query was received in the same program message after an query requesting an indefinite response was executed (see *IEEE 488.2*, 6.5.7.5).] |

**Table 19 Overview for Unsupported Strings**

| New/Old/Standard | Error | |
|---|---|---|
| | 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 ???" |

| New/Old/Standard | Error | |
|---|---|---|
| | Number | String |
| Old | -210 | “Trigger error” |
| Old | -214 | “Trigger deadlock” |
| Old | -215 | “Arm deadlock” |
| Old | -230 | “Data corrupt or stale” |
| Old | -240 | “Hard ware error” |
| Old | -241 | “Hard ware 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” |

# Index

## R



## S



## T



## U



## V



## W

This information is subject to
change without notice.
© Keysight Technologies 2016
Edition 2.0, June 2016


www.keysight.com

KEYSIGHT
TECHNOLOGIES