

**Agilent N7744A / N7745A  
Multiport Optical Power Meter**

LXI Compliant Power Meters

**Programming Guide**



**Agilent Technologies**

# Notices

© Agilent Technologies, Inc. 2009

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 Agilent Technologies, Inc. as governed by United States and international copyright laws.

## Manual Part Number

N7744-90C01

## Edition

Second edition, February 2009

Printed in Germany

Agilent Technologies Deutschland GmbH  
Herrenberger Str. 130  
71034 Böblingen, Germany

## Warranty

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

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

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

## Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside

of the environmental specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. Agilent Technologies specifically disclaims the implied warranties of Merchantability and Fitness for a Particular Purpose.

## 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.

## Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or sub-contract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

## Exclusive Remedies

The remedies provided herein are Buyer's sole and exclusive remedies. Agilent Technologies shall not be liable for any direct, indirect, special, incidental, or consequential damages whether based on contract, tort, or any other legal theory.

## Assistance

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

## Certification

Agilent Technologies Inc. certifies that this product met its published specifications at the time of shipment from the factory. Agilent Technologies further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, NIST (formerly the United States National Bureau of Standards, NBS) to the extent allowed by the Institutes's calibration facility, and to the calibration facilities of other International Standards Organization members.

## ISO 9001 Certification

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

## Safety Notices

### CAUTION

A **CAUTION** notice 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 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.

## Warnings and Notices

### **WARNING**

To avoid the possibility of injury or death, you must observe the following precautions before switching on the instrument.

Insert the power cable plug only into a socket outlet provided with a protective earth contact. Do not negate this protective action by the using an extension cord without a protective conductor.

---

### **WARNING**

Never look directly into the end of a fiber or a connector, unless you are absolutely certain that there is no signal in the fiber.

---

## In this Manual

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

- Agilent N7744A / N7745A Multiport Optical Power Meter

These instruments can also be programmed via the IVI drivers (IVI-COM, IVI-C), available on

<http://www.agilent.com/find/ivi-com>

### Conventions used in this Manual

- All commands and typed text is written in Courier font, for example INIT[:IMM].
- SCPI commands are written in mixed case: text that you MUST print is written in capitals; text which is helpful but nor necessary is written in lower case.  
So, the command INITiate[:IMMEDIATE] can be entered either as init[:imm], or as initiate[:immediate]. It does not matter whether you enter text using capitals or lower-case letters.
- SCPI commands often contain extra arguments in square brackets. These arguments may be helpful, but they need not be entered.  
So, the command INITiate[:IMMEDIATE] can be entered as init or initiate:imm.
- A SCPI command which can be either a command or a query is appended with the text /?.  
So, DISPlay:ENABle/? refers to both the command DISPlay:ENABle and the query DISPlay:ENABle?.

### Related Manuals

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

- *Agilent N7744A / N7745A Multiport Optical Power Meter User's Guide.*

#### NOTE

Refer to the books listed in “[Introduction to Programming](#)” on page 7 for additional information about the General Purpose Interface Bus, GPIB.

# Contents

## 1 Introduction to Programming

GPIB Interface	8
Setting the GPIB Address	9
Using the Web-Enabled Instrument Interface	10
Browser Configuration	10
Message Queues	12
How the Input Queue Works	12
The Output Queue	12
The Error Queue	13
Programming and Syntax Diagram Conventions	14
Short Form and Long Form	14
Command and Query Syntax	15
Common Commands	18
Common Command Summary	18
Common Status Information	18
The Status Model	20
Status Registers	20
Status System	22
Annotations	23
Status Command Summary	25
Other Commands	25

## 2 Specific Commands

Specific Command Summary	28
--------------------------	----

## 3 Instrument Setup and Status

IEEE-Common Commands	34
Status Reporting – The STATus Subsystem	40
Interface/Instrument Behaviour Settings – The SYSTem Subsystem	47
:SYSTem:COMMunicate:ETHernet subtree	49
Handling Measurement Settings - The :CONFigure:MEASurement:SETTing subtree	56

## 4 Measurement Operations & Settings

- Root Layer Command 60
- Measurement Functions – The FETCh, INITiate, READ and SENSE Subsystems 63
  - Using data buffers for simultaneous measurement and upload 70
- Triggering - The TRIGger Subsystem 82

## 5 VISA Programming Examples

- How to Use VISA Calls 86
- How to Measure Power using FETCh and READ 88
- How to Log Results 91

## 6 The Agilent 816x VXIplug&play Instrument Driver

- Installing the Agilent 816x Instrument Driver 96
- Using Visual Programming Environments 100
  - Getting Started with Agilent VEE 100
  - Getting Started with LabView 102
  - Getting Started with LabWindows 105
- Features of the Agilent 816x Instrument Driver 106
- Directory Structure 107
- Opening an Instrument Session 108
- Closing an Instrument Session 109
- VISA Data Types and Selected Constant Definitions 110
- Error Handling 111
- Introduction to Programming 113
  - Example Programs 113
  - VISA-Specific Information 113
  - Development Environments 113
- Online Information 115
- Lambda Scan Applications 116
  - How to Perform a Multi-Frame Lambda Scan Application 118

## 7 Error Codes

- GPIB Error Strings 124



# 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 Agilent N7744A / N7745A Multiport Power Meter and assumes that you are already familiar with programming the GPIB.

GPIB Interface	8
Setting the GPIB Address	9
Message Queues	12
How the Input Queue Works	12
The Output Queue	12
The Error Queue	13
Programming and Syntax Diagram Conventions	14
Short Form and Long Form	14
Command and Query Syntax	15
Common Commands	18
Common Command Summary	18
Common Status Information	18
The Status Model	20
Status Registers	20
Status System	22
Annotations	23
Status Command Summary	25
Other Commands	25



## GPIB Interface

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

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

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

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

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

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

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

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

- The SCPI Consortium: *Standard Commands for Programmable Instruments*. To obtain a copy of this manual, contact the following address:

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

Web: <http://www.scpiconsortium.org>

The interface of the Agilent N7744A / N7745A Multiport Power Meter to the GPIB is defined by the IEEE Standards 488.1 and 488.2.



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

**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]:ADDRes”` on page 49.
- You can set the GPIB address from the web interface. 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.

## Using the Web-Enabled Instrument Interface

The LAN connection between the PC and the Agilent N7744A / N7745A provides access to the instrument through the instrument's Web-enabled interface. This is the easiest way to program the Agilent N7744A / N7745A as no additional IO drivers are required.

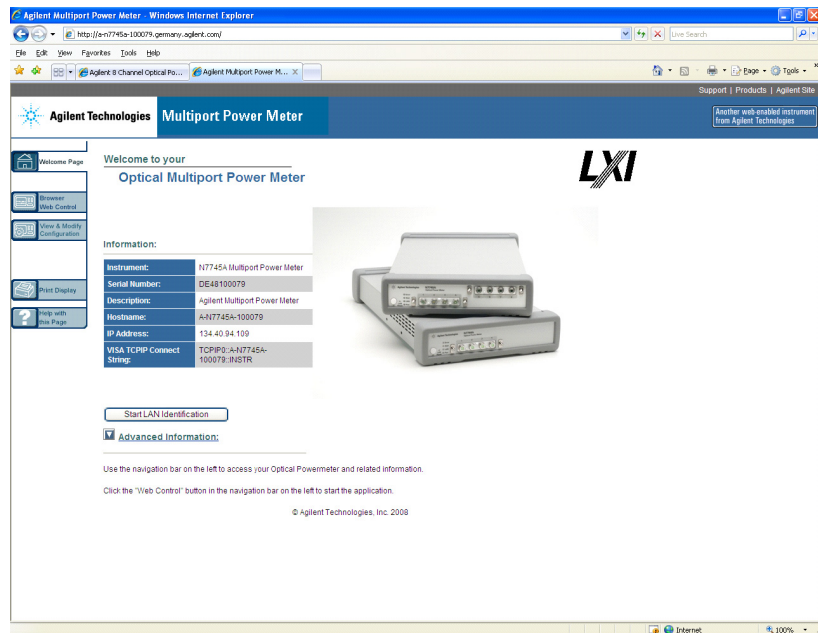
To use the Web-enabled interface:

- 1 Connect a LAN cable (yellow crossover or standard) between your PC and the Agilent N7744A / N7745A. The Web-enabled interface is not available over GPIB or USB.
- 2 If necessary, turn on the Agilent N7744A / N7745A. Following the power-on sequence note the IP address displayed on the Agilent N7744A / N7745A front panel.
- 3 Open the Internet browser on your PC. In the browser 'Address' window, type "http://" followed by the IP address of the Agilent N7744A / N7745A.

The Web interface 'Welcome Page' appears as shown in [Figure 1](#).

### Browser Configuration

The Web-enabled interface is activated from the PC's Internet browser. In some network configurations, however, a proxy server may prevent access to the instrument (i.e. "page cannot be displayed"). In this situation, the proxy must be configured from the browser such that the proxy is not used for (IP) addresses within the range of those that can be assigned to the Agilent N7744A / N7745A.



**Figure 1** The N7744A/N7745A Web-Enabled Interface 'Welcome Page'.

**NOTE**

Selecting "Help with this Page" on any Web interface window provides definitions and detailed information on the use of that window.

**NOTE**

For ease in (Internet) browser navigation when controlling multiple instruments, open a separate browser session for each Web-enabled instrument.

## 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:

<b>string</b>	is ascii data. A string is contained between double quotes ("...") or single quotes ('...').
<b>value</b>	is numeric data in integer (12), decimal (34.5) or exponential format (67.8E-9).
<b>wsp</b>	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>.

### 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 1 to 4 for the Agilent N7744A
- from 1 to 8 for the Agilent N7745A,

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

Channel numbers are not used and are optional in all commands for the Multiport Power Meter. They are accepted to ensure compatibility with the Agilent 816x power meters.

For example, if you want to query slot 1 with the command, :SENSe[n]:[CHANnel[m]]:POWER:WAVelength?, you should send the command:



:sens1: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.
  - Channel 1.
-

## Common Commands

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

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

### Common Command Summary

Table 3 gives a summary of the common commands.

**Table 3** Common Command Summary

Command	Parameter	Function	Page
*CLS		Clear Status Command	<a href="#">page 34</a>
*ESE		Standard Event Status Enable Command	<a href="#">page 34</a>
*ESE?		Standard Event Status Enable Query	<a href="#">page 35</a>
*ESR?		Standard Event Status Register Query	<a href="#">page 35</a>
*IDN?		Identification Query	<a href="#">page 35</a>
*OPC		Operation Complete Command	<a href="#">page 36</a>
*OPC?		Operation Complete Query	<a href="#">page 36</a>
*OPT?		Options Query	<a href="#">page 37</a>
*RST		Reset Command	<a href="#">page 37</a>
*STB?		Read Status Byte Query	<a href="#">page 37</a>
*TST?		Self Test Query	<a href="#">page 38</a>
*WAI		Wait Command	<a href="#">page 39</a>

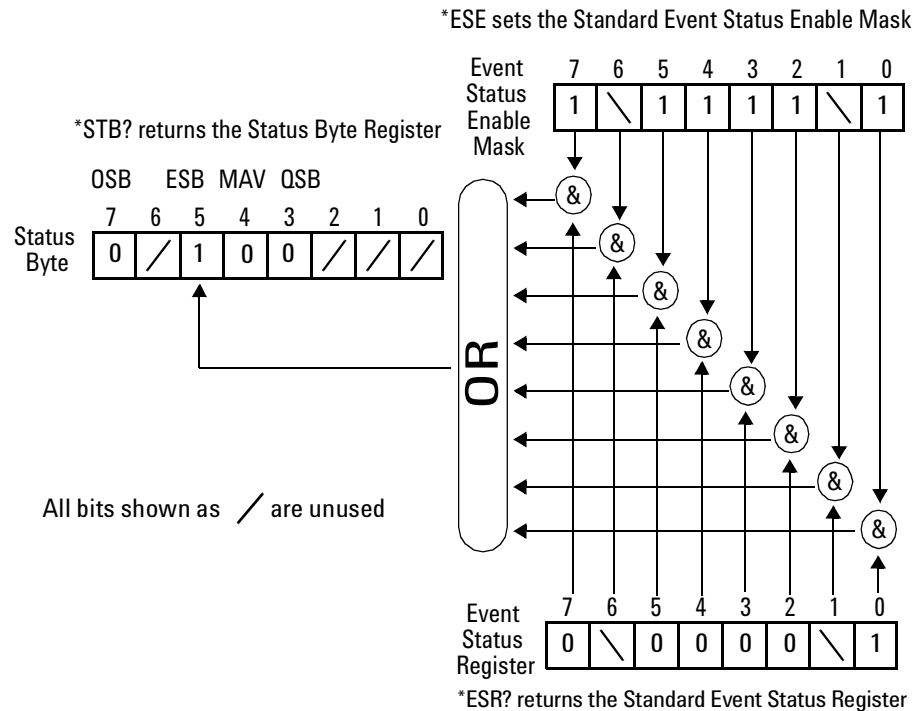
#### NOTE

These commands are described in more detail in ["IEEE-Common Commands"](#) on page 34.

### Common Status Information

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

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



**Figure 2** The Event Status Bit

The SESR contains the information about events that are not slot specific. For details of the function of each bit of the SESR, see “Standard Event Status Register” on page 23.

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 20 and “[Status Reporting – The STATUS Subsystem](#)” on page 40.

**NOTE**

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

## The Status Model

### Status Registers

Each node of the status circuitry has three registers:

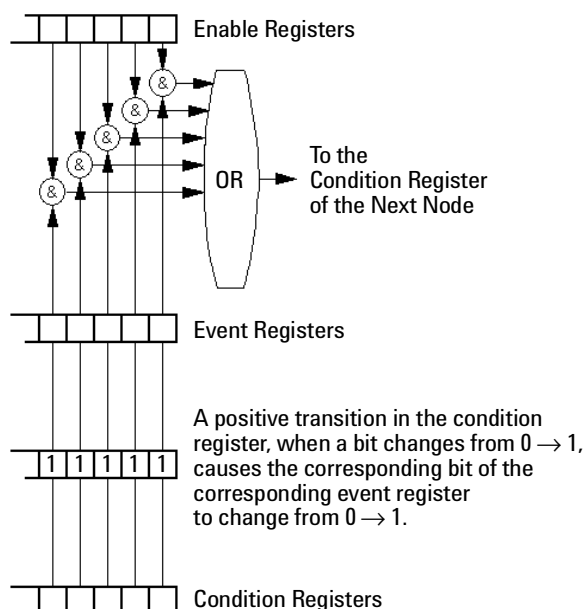
- A condition register (CONDition), which contains the current status. This register is updated continuously. It is not changed by having its contents read.
- The event register (EVENT), which contains details of any positive transitions in the corresponding condition register, that is, when a bit changes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

#### NOTE

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

---

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



**Figure 3** The Registers and Filters for a Node

The Operational/Questionable Slot Status Event Register (OSSER/QSSER) contains the status of a particular module slot. A bit changes from 0 → 1 when an event occurs, for example, when a laser is switched on. For details of the function of each bit of these registers, see “Operation/Questionable Status Summary Register” on page 23.

The Operational/Questionable Slot Enable Status Mask (OSESME/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 OSESME/QSESM to zero, the occurrence 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 OSESME/QSESM to be set to 0.

The Operational/Questionable Status Event Summary Register (OSESRE/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 QSESM is 1 at the same time, the QSESR bit representing that slot is set to 1.

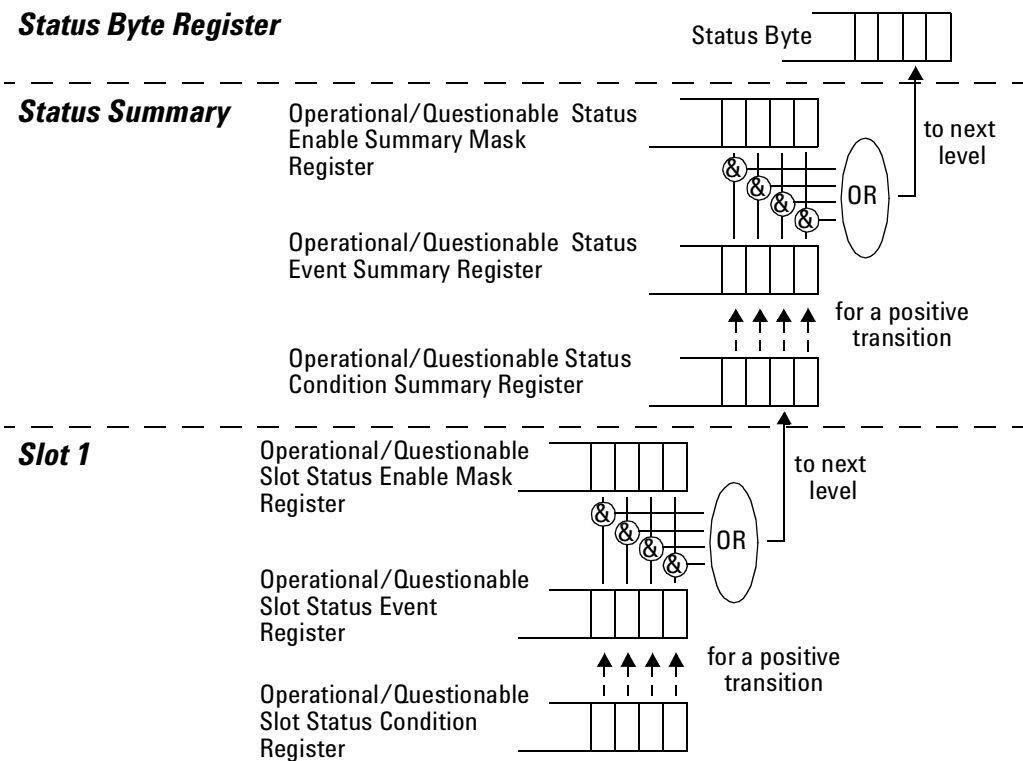
The Operational/Questionable Status Enable Summary Mask (OSESME/QSESM) allows you to choose the module slots that may affect the OSB/QSB of the Status Byte. If any bit of the QSESR goes from 0 → 1 AND the corresponding bit of the QSESM is 1 at the same time, the QSB of the Status Byte is set to

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 Agilent N7744A / N7745A Multiport Power Meter consists of one level. These are described in “Status System” on page 22.

## Status System

The status system for the Agilent N7744A / N7745A Multiport Power Meter returns the status of 4 and 8 module slots respectively. The Operational/Questionable Status Summary Registers consist of one level and are described by Figure 4. Any commands that require LEVEL1 do not apply to these mainframes.



**Figure 4** The Operational/Questionable Status System

## 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 3 is set if Power Meter zeroing.
- All other bits are unused, and therefore set to 0.

### **Questionable Slot Status Register**

- Bit 0 is set 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 5 is set if the module is out of specifications.
- All other bits are unused, and therefore set to 0.



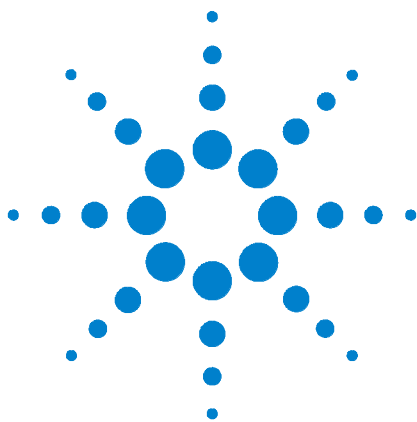
## Status Command Summary

*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. <b>NOTE:</b> *RST does NOT touch the STB or SESR. A running measurement is stopped.
*TST?	initiates an instrument selftest and returns the results as a 32 bit LONG.

## Other Commands

*OPT?	returns the installed modules and the slots these modules are installed in: For example, *OPT? → ???, , A modules is installed in slot 0. Slot 1 is 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.

## **1 Introduction to Programming**



## 2 Specific Commands

This chapter lists all the instrument specific commands relating to the Agilent N7744A / N7745A Multiport Power Meter 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 28



## 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 `:init:cont`

- `INITiate` is the subsystem containing all commands and queries concerned with setting up the power meter,
- `CONTinuous` is the command setting a continuous measurement.

### NOTE

If a command and a query are both available, the command ends `/?`. So, `:init:cont/?` means that `init:cont` and `init:cont?` are both available.

Table 1 gives an overview of the command tree. You see the nodes, the subnodes, and the included commands.

**Table 1** Specific Command Summary

Command	Description	Page
<code>:CONFigure:MEASurement:SETTing</code>		
<code>:ACTual?</code>	Get the index of the setting currently being used.	<a href="#">page 56</a>
<code>:CANCel</code>	Discard all the changes to the setting since the last save or recall	<a href="#">page 56</a>
<code>:ERASe</code>	Erase a setting from memory.	<a href="#">page 56</a>
<code>:NUMBer?</code>	Get the number of settings.	<a href="#">page 57</a>
<code>:PRESet</code>	Resets the setting values in the working memory	<a href="#">page 57</a>
<code>:RECall</code>	Recall a setting from FLASH memory.	<a href="#">page 57</a>
<code>:SAVE</code>	Save the current setting to FLASH memory.	<a href="#">page 57</a>
<code>:FETCh[n][:CHANnel[m]][:SCALar]</code>		
<code>:POWer[:DC]?</code>	Returns the most recent power value from a sensor.	<a href="#">page 63</a>
<code>:POWer[:DC]:ALL?</code>	Returns the most recent power values from all the sensors.	<a href="#">page 63</a>
<code>:POWer[:DC]:ALL:CSV?</code>	Returns a string of the most recent power values from all the sensors in comma separated format.	<a href="#">page 63</a>
<code>:INITiate[n][:CHANnel[m]]</code>		
<code>[:IMMediate]</code>	Starts a measurement.	<a href="#">page 64</a>
<code>:CONTinuous/?</code>	Starts or Queries a single/continuous measurement.	<a href="#">page 65</a>

Table 1 Specific Command Summary (continued)

Command	Description	Page
:READ[n][:CHANnel[m]][:SCALar]		
:POWer[:DC]?	Returns a power value from a sensor.	page 63
:POWer[:DC]:ALL?	Returns the power values from all the sensors.	page 65
:SENSe[n][:CHANnel[m]]:CORRection [:LOSS][:INPut][:MAGNitude]/?	Sets or returns the value of correction data for a sensor.	page 66
:COLLECT:ZERO	Executes a zero calibration of a sensor module.	page 66
:COLLECT:ZERO?	Returns the current zero state of a sensor module.	page 67
:COLLECT:ZERO:ALL/?	Executes a zero calibration of all sensor modules, or returns the status of the most recent zero calibration of all sensor modules.	page 67
:SENSe[n][:CHANnel[m]]:FUNCTion		
:LOOP/?	Sets or returns the number of logging loops.	page 72
:PARAmeter:LOGGing/?	Sets or returns the number of samples and the averaging time, $t_{avg}$ , for logging.	page 68
:PARAmeter:MINMax/?	Sets or returns the minmax mode and the window size.	page 69
:PARAmeter:STABility/?	Sets or returns the total time, delay time and the averaging time, $t_{avg}$ , for stability.	page 70
:RESult?	Returns the data array of the last function.	page 73
:RESult:BUFA?	Returns the data array of the last data acquisition function in Buffer A.	page 73
:RESult:BUFB?	Returns the data array of the last data acquisition function in Buffer B.	page 74
:RESult:INDex?	Returns the number of logging loops that have already finished.	page 73
:STATe/?	Enables/disables the function mode or returns whether the function mode is enabled.	page 74
:SENSe[n][:CHANnel[m]]:POWer		
:ATIMe/?	Sets or returns the average time of a sensor.	page 75
:GAIN:AUTO/?	Sets or returns the auto gain setting of a sensor, which optimizes the dynamic or transient response.	page 77
:RANGe:AUTO/?	Sets or returns the range of a sensor to produce the most dynamic range without overloading.	page 75
:RANGe[:UPPer]/?	Sets or returns the most positive signal entry expected for a sensor.	page 77
:REFerence/?	Sets or returns the reference level of a sensor.	page 78
:UNIT/?	Sets or returns the units used for absolute readings on a sensor.	page 80
:UNIT:ALL:CSV?	Returns the units from all the ports of the instrument.	
:WAVelength/?	Sets or returns the wavelength for a sensor.	page 80

## 2 Specific Commands

**Table 1** Specific Command Summary (continued)

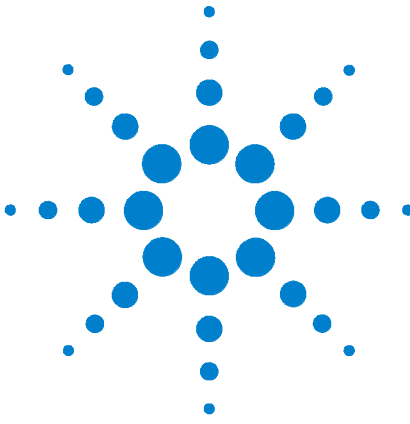
Command	Description	Page
:SENSe[n][:CHANnel[m]]:POWer:Reference		
:DISPlay	Sets the reference level for a sensor from the input power level.	<a href="#">page 78</a>
:STATe/?	Sets or returns whether sensor results are in relative or absolute units.	<a href="#">page 79</a>
:STATe:RATio/?	Sets or returns whether sensor results are displayed relative to a slot or to an absolute reference.	<a href="#">page 79</a>
:SLOT[n][:HEAD]		
:EMPTy?	Returns whether the module slot is empty.	<a href="#">page 60</a>
:IDN?	Returns information about the module.	<a href="#">page 60</a>
:OPTions?	Returns the module's options.	<a href="#">page 60</a>
:TST?	Returns the latest selftest results for a module.	<a href="#">page 60</a>
:WAVelength:RESPonse?	Returns the wavelength response from port <i>n</i> , in binary data format.	<a href="#">page 61</a>
:WAVelength:RESPonse:CSV?	Returns the wavelength response from port <i>n</i> , in .csv data format.	<a href="#">page 61</a>
:WAVelength:RESPonse:SIZE?	Returns the number of elements in the wavelength response table.	<a href="#">page 61</a>
:STATus:OPERation		
[:EVENT]?	Returns the Operational Status Event Summary Register (OESR).	<a href="#">page 42</a>
:CONDition?	Returns the Operational Status Condition Summary Register.	<a href="#">page 42</a>
:ENABle/?	Sets or queries the Operational Status Enable Summary Mask.	<a href="#">page 42</a>
:STATusn:OPERation		
[:EVENT]?	Returns the Operational Slot Status Event Register for slot <i>n</i> .	<a href="#">page 42</a>
:CONDition?	Returns the Operational Slot Status Condition Register for slot <i>n</i> .	<a href="#">page 42</a>
:ENABle/?	Sets or queries the Operation Slot Status Enable Mask for slot <i>n</i> .	<a href="#">page 42</a>
:STATus:QUEStionable		
[:EVENT]?	Returns the Questionable Status Event Summary Register.	<a href="#">page 43</a>
:CONDition?	Returns the Questionable Status Condition Summary Register.	<a href="#">page 46</a>
:ENABle/?	Sets or queries the Questionable Status Enable Summary Mask.	<a href="#">page 46</a>
:STATus:QUEStionable		
[:EVENT]?	Returns the Questionable Slot Status Event Register for slot <i>n</i> .	<a href="#">page 43</a>
:CONDition?	Returns the Questionable Slot Status Condition Register for slot <i>n</i> .	<a href="#">page 46</a>
:ENABle/?	Sets or queries the Questionable Slot Status Enable Mask for slot <i>n</i> .	<a href="#">page 46</a>

Table 1 Specific Command Summary (continued)

Command	Description	Page
:SYSTem		
:DATE/?	The Multiport Power Meter has no internal clock. This command is included for backwards compatibility.	page 47
:ERRor?	Returns the contents of the instrument's error queue.	page 47
:HELP:HEADers?	Returns a list of GPIB commands.	page 47
:LXI:IDN	Starts or stops the LAN LED. This can be used to identify the unit.	page 48
:PRESet	Sets all parameters to their default values.	page 48
:REBoot	Reboots the instrument	page 48
:TIME/?	The Multiport Power Meter has no internal clock. This command is included for backwards compatibility.	page 48
:VERSion?	Returns the instrument's SCPI version.	page 49
:SYSTem:COMMunicate:ETHernet		
:CANCel	Undo changes to the network parameters.	page 55
:DGATeway/?	Set/Get the default gateway.	page 54
:DGATeway:CURRent?	Get the currently used default gateway.	page 52
:DHCP:ENABle/?	Enable/Check whether DHCP is enabled or disabled.	page 50
:DOMainname/?	Set/Get the domain name	page 52
:DOMainname:CURRent?	Get the currently used domain name	page 51
:HOSTname/?	Set/Get the host name	page 52
:HOSTname:CURRent?	Get the current host name.	page 51
:IPADdress/?	Set/Get the manually set IP address of the system.	page 53
:IPADdress:CURRent?	Get the current IP address of the system.	page 51
:MACaddress?	Get the MAC address of the network adapter.	page 50
:RESet	Resets all LAN parameters to the factory default.	page 54
:REStart	Restart the system's network interface.	page 54
:SAVE	Save the system's network interface parameters.	page 55
:SMASK/?	Set/Get the subnet mask.	page 53
:SMASK:CURRent?	Get the currently used subnet mask.	page 51
:SYSTem:COMMunicate:GPIB		
[:SELF]:ADDRes/?	Sets or returns the GPIB address.	page 49
:TRIGger:CONFIguration/?	Sets or returns trigger configuration.	page 84
:TRIGger[n][CHANnel[m]]		
:INPut/?	Sets or returns the incoming trigger response .	page 82
:OUTPut/?	Sets or returns the outgoing trigger response.	page 83

## 2 Specific Commands





## 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 subsystems:

- IEEE specific commands that were introduced in “[Common Commands](#)” on page 18.
- STATus subsystem commands that relate to the status model.
- SYSTem subsystem commands that control the serial interface and internal data.

IEEE-Common Commands 34

Status Reporting – The STATus Subsystem 40

Interface/Instrument Behaviour Settings – The SYSTem Subsystem 47

Handling Measurement Settings - The :CONFigure:MEASurement:SETTING subtree 56



## IEEE-Common Commands

“Common Commands” on page 18 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:	<b>*CLS</b>
syntax:	*CLS
description:	<p>The CLear Status command *CLS clears the following:</p> <ul style="list-style-type: none"> <li>Error queue</li> <li>Standard event status register (SESR)</li> <li>Status byte register (STB)</li> </ul> <p>After the *CLS command the instrument is left waiting for the next command. The instrument setting is unaltered by the command, although *OPC/*OPC? actions are cancelled.</p>
parameters:	none
response:	none
example:	*CLS

command:	<b>*ESE</b>																											
syntax:	*ESE<wsp><value> $0 \leq \text{value} \leq 255$																											
description:	<p>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).</p> <p>The register is cleared:</p> <ul style="list-style-type: none"> <li>at power-on,</li> <li>by sending a value of zero.</li> </ul> <p>The register is not changed by the *RST and *CLS commands.</p>																											
parameters:	<p>The bit value for the register (a 16-bit signed integer value):</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Mnemonic</th> <th>Decimal Value</th> </tr> </thead> <tbody> <tr> <td>7 (MSB)</td> <td>Power On</td> <td>128</td> </tr> <tr> <td>6</td> <td>Not Used</td> <td>0</td> </tr> <tr> <td>5</td> <td>Command Error</td> <td>32</td> </tr> <tr> <td>4</td> <td>Execution Error</td> <td>16</td> </tr> <tr> <td>3</td> <td>Device Dependent Error</td> <td>8</td> </tr> <tr> <td>2</td> <td>Query Error</td> <td>4</td> </tr> <tr> <td>1</td> <td>Not Used</td> <td>0</td> </tr> <tr> <td>0 (LSB)</td> <td>Operation Complete</td> <td>1</td> </tr> </tbody> </table>	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
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:	<b>*ESE?</b>
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:	<b>*ESR?</b>																											
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):																											
	<table border="1"> <thead> <tr> <th>Bit</th> <th>Mnemonic</th> <th>Decimal Value</th> </tr> </thead> <tbody> <tr> <td>7 (MSB)</td> <td>Power On</td> <td>128</td> </tr> <tr> <td>6</td> <td>Not used</td> <td>0</td> </tr> <tr> <td>5</td> <td>Command Error</td> <td>32</td> </tr> <tr> <td>4</td> <td>Execution Error</td> <td>16</td> </tr> <tr> <td>3</td> <td>Device Dependent Error</td> <td>8</td> </tr> <tr> <td>2</td> <td>Query Error</td> <td>4</td> </tr> <tr> <td>1</td> <td>Not used</td> <td>0</td> </tr> <tr> <td>0 (LSB)</td> <td>Operation Complete</td> <td>1</td> </tr> </tbody> </table>	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
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:	<b>*IDN?</b>
syntax:	*IDN?
description:	The IDeNtification query *IDN? gets the instrument identification over the interface.
parameters:	none



parameters:	none
response:	1<END> is returned if all modules are ready to execute a new operation. 0<END> is returned if any module is busy.
example:	*OPC? → 1<END>

command:	<b>*OPT?</b>
syntax:	*OPT?
description:	The OPTions query *OPT? returns the modules installed in your instrument.
parameters:	none
response:	Returns the part number of all installed modules, separated by commas. Slots are listed starting with the lowest slot number, that is, slot 0.  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? → N7744A , , <END>

command:	<b>*RST</b>
syntax:	*RST
description:	The ReSeT command *RST sets the mainframe and all modules to the reset setting (standard setting) stored internally. Pending *OPC? actions are cancelled. The instrument is placed in the idle state awaiting a command. The *RST command clears the error queue. The *RST command is equivalent to the *CLS command AND the syst:preset command. The following are not changed: <ul style="list-style-type: none"> <li> GPIB (interface) state</li> <li> Instrument interface address</li> <li> Output queue</li> <li> Service request enable register (SRE)</li> <li> Standard Event Status Enable Mask (SESEM)</li> </ul>
parameters:	none
response:	none
example:	*RST

command:	<b>*STB?</b>
syntax:	*STB?
description:	The SStatus Byte query *STB? returns the contents of the Status Byte register.
parameters:	none

### 3 Instrument Setup and Status

response:	The bit value for the register (a <i>16-bit signed integer</i> value):		
	<b>Bit</b>	<b>Mnemonic</b>	<b>Decimal Value</b>
	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:	<b>*TST?</b>
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 $0 \leq \text{value} \leq 4294967296$ ):		
	<b>Bits</b>	<b>Mnemonic</b>	<b>Decimal Value</b>
	31	Selftest failed on Mainframe	A negative value
	18 - 30	Not used	0
	17	Selftest failed on Slot 17	131072
	16	Selftest failed on Slot 16	65536
	15	Selftest failed on Slot 15	32768
	14	Selftest failed on Slot 14	16384
	13	Selftest failed on Slot 13	8192
	12	Selftest failed on Slot 12	4096
	11	Selftest failed on Slot 11	2048
	10	Selftest failed on Slot 10	1024
	9	Selftest failed on Slot 9	512
	8	Selftest failed on Slot 8	256
	7	Selftest failed on Slot 7	128
	6	Selftest failed on Slot 6	64
	5	Selftest failed on Slot 5	32
	4	Selftest failed on Slot 4	16
	3	Selftest failed on Slot 3	8
	2	Selftest failed on Slot 2	4
	1	Selftest failed on Slot 1	2
	0	Selftest failed on Slot 0	1
	If 16 is returned, the module in slot 4 has failed.		
	If 18 is returned, the modules in slots 1 and 4 have failed.		
	A value of zero indicates no errors.		
example:	*TST? → 0<END>		

command:	<b>*WAI</b>
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 20.

command:	<b>:STATus:OPERation[:EVENT][:LEVel]?</b>		
syntax:	:STATus:OPERation[:EVENT][:LEVel]?		
description:	Returns the Operational Status Event Summary Register (OSESr).		
parameters:	none		
response:	The sum of the results for the slots (a <i>16-bit signed integer</i> value, where $0 \leq \text{value} \leq 32767$ ):		
	<b>Bits</b>	<b>Mnemonics</b>	<b>Decimal Value</b>
	15	Not used	0
	14	Not used	16384
	13	Not used	8192
	12	Not used	4096
	11	Not used	2048
	10	Not used	1024
	9	Not used	512
	8	Not used	256
	7	Slot 8 Summary	128
	6	Slot 7 Summary	64
	5	Slot 6 Summary	32
	4	Slot 5 Summary	16
	3	Slot 4 Summary	8
	2	Slot 3 Summary	4
	1	Slot 2 Summary	2
	0	Slot 1 Summary	1
example:	stat:oper? → +0<END>		

command:	<b>:STATus:OPERation:CONDition[:LEVel]?</b>		
syntax:	:STATus:OPERation:CONDition[:LEVel]?		
description:	Reads the Operational Status Condition Summary Register.		
parameters:	none		



response:	The sum of the results for the individual slots (a <i>16-bit signed integer</i> value, where $0 \leq \text{value} \leq 32767$ ):		
	<b>Bits</b>	<b>Mnemonics</b>	<b>Decimal Value</b>
	15	Not used	0
	14	Not used	16384
	13	Not used	8192
	12	Not used	4096
	11	Not used	2048
	10	Not used	1024
	9	Not used	512
	8	Not used	256
	7	Slot 8 Summary	128
	6	Slot 7 Summary	64
	5	Slot 6 Summary	32
	4	Slot 5 Summary	16
	3	Slot 4 Summary	8
	2	Slot 3 Summary	4
	1	Slot 2 Summary	2
	0	Slot 1 Summary	1
example:	stat:oper:cond? → +0<END>		

command:	<b>:STATus:OPERation:ENABLE[:LEVel]</b>
syntax:	:STATus:OPERation:ENABLE[:LEVel]<wsp><value>
description:	Sets the bits in the Operational Status Enable Summary Mask (OSESMS) that enable the contents of the OSESR to affect the Status Byte (STB). Setting a bit in this register to 1 enables the corresponding bit in the OSESR to affect bit 7 of the Status Byte.
parameters:	The bit value for the OSESMS as a <i>16-bit signed integer</i> value (0 .. +32767) The default value is 0.
response:	none
example:	stat:oper:enab 128

command:	<b>:STATus:OPERation:ENABLE[:LEVel]?</b>
syntax:	:STATus:OPERation:ENABLE[:LEVel]?
description:	Returns the OSESMS for the OSESR
parameters:	none
response:	The bit value for the operation enable mask as a <i>16-bit signed integer</i> value (0 .. +32767)
example:	stat:oper:enab? → +128<END>

### 3 Instrument Setup and Status

command:	<b>:STATus<math>n</math>:OPERation[:EVENT]?</b>		
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 <i>16-bit signed integer</i> value, where $0 \leq \text{value} \leq 32767$ ):		
	<b>Bit</b>	<b>Mnemonic</b>	<b>Decimal Value</b>
	8-15	Not used	0
	7	Not used	128
	6	Not used	64
	5	Not used	32
	4	Not used	16
	3	Slot $n$ : Zeroing ongoing	8
	2	Not used	0
	1	Not used	2
	0	Not used	1
example:	stat1:oper? → +0<END>		

command:	<b>:STATus<math>n</math>:OPERation:CONDition?</b>		
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 <i>16-bit signed integer</i> value, where $0 \leq \text{value} \leq 32767$ ):		
	<b>Bit</b>	<b>Mnemonic</b>	<b>Decimal Value</b>
	8-15	Not used	0
	7	Not used	128
	6	Not used	64
	5	Not used	32
	4	Not used	16
	3	Not used	8
	2	Slot $n$ : Zeroing ongoing	0
	1	Not used	2
	0	Not used	1
		Not used	
example:	stat1:oper:cond? → +0<END>		

command:	<b>:STATus<math>n</math>:OPERation:ENABle</b>
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. 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 <i>16-bit signed integer</i> value (0 .. +32767)
response:	none
example:	stat:oper:enab 128

command:	<b>:STATus:n:OPERation:ENABLE?</b>
syntax:	:STATus:n:OPERation:ENABLE?
description:	Returns the OSSEM of slot $n$
parameters:	none
response:	The bit value for the OSSEM as a <i>16-bit signed integer</i> value (0 .. +32767)
example:	stat:oper:enab? → +128<END>

command:	<b>:STATus:PRESet</b>
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:	<b>:STATus:QUEStionable[:EVENT][:LEVel]?</b>
syntax:	:STATus:QUEStionable[:EVENT][:LEVel]?
description:	Returns the Questionable Status Event Summary Register (QSESR).
parameters:	none

### 3 Instrument Setup and Status

response:	The sum of the results for the QSESR as a <i>16-bit signed integer</i> value (0 .. +32767)		
	<b>Bits</b>	<b>Mnemonics</b>	<b>Decimal Value</b>
	15	Not used	0
	14	Not used	16384
	13	Not used	8192
	12	Not used	4096
	11	Not used	2048
	10	Not used	1024
	9	Not used	512
	8	Not used	256
	7	Not used	128
	6	Not used	64
	5	Not used	32
	4	Not used	16
	3	Not used	8
	2	Not used	4
	1	Slot 2 Summary	2
	0	Slot 1 Summary	1
example:	stat:ques? → +0<END>		

command:	<b>:STATus:QUEStionable:CONDition[:LEVel]?</b>
syntax:	:STATus:QUEStionable:CONDition[:LEVel]?
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 <i>16-bit signed integer</i> value (0 .. +32767)																																																			
	<table border="1"> <thead> <tr> <th>Bits</th> <th>Mnemonics</th> <th>Decimal Value</th> </tr> </thead> <tbody> <tr><td>15</td><td>Not used</td><td>0</td></tr> <tr><td>14</td><td>Not used</td><td>16384</td></tr> <tr><td>13</td><td>Not used</td><td>8192</td></tr> <tr><td>12</td><td>Not used</td><td>4096</td></tr> <tr><td>11</td><td>Not used</td><td>2048</td></tr> <tr><td>10</td><td>Not used</td><td>1024</td></tr> <tr><td>9</td><td>Not used</td><td>512</td></tr> <tr><td>8</td><td>Not used</td><td>256</td></tr> <tr><td>7</td><td>Not used</td><td>128</td></tr> <tr><td>6</td><td>Not used</td><td>64</td></tr> <tr><td>5</td><td>Not used</td><td>32</td></tr> <tr><td>4</td><td>Not used</td><td>16</td></tr> <tr><td>3</td><td>Not used</td><td>8</td></tr> <tr><td>2</td><td>Not used</td><td>4</td></tr> <tr><td>1</td><td>Slot 2 Summary</td><td>2</td></tr> <tr><td>0</td><td>Slot 1 Summary</td><td>1</td></tr> </tbody> </table>	Bits	Mnemonics	Decimal Value	15	Not used	0	14	Not used	16384	13	Not used	8192	12	Not used	4096	11	Not used	2048	10	Not used	1024	9	Not used	512	8	Not used	256	7	Not used	128	6	Not used	64	5	Not used	32	4	Not used	16	3	Not used	8	2	Not used	4	1	Slot 2 Summary	2	0	Slot 1 Summary	1
Bits	Mnemonics	Decimal Value																																																		
15	Not used	0																																																		
14	Not used	16384																																																		
13	Not used	8192																																																		
12	Not used	4096																																																		
11	Not used	2048																																																		
10	Not used	1024																																																		
9	Not used	512																																																		
8	Not used	256																																																		
7	Not used	128																																																		
6	Not used	64																																																		
5	Not used	32																																																		
4	Not used	16																																																		
3	Not used	8																																																		
2	Not used	4																																																		
1	Slot 2 Summary	2																																																		
0	Slot 1 Summary	1																																																		
example:	stat:ques:cond? → +0<END>																																																			

command:	<b>:STATus:QUESTionable:ENABLE[:LEVel]</b>
syntax:	:STATus:QUESTionable:ENABLE[:LEVel0]<wsp><value>
description:	Sets the bits in the Questionable Status Enable Summary Mask (QSESM) that enable the contents of the QSESR to affect the Status Byte (STB). Setting a bit in this register to 1 enables the corresponding bit in the QSESR to affect bit 3 of the Status Byte.
parameters:	The bit value for the questionable enable mask as a <i>16-bit signed integer</i> value (0 .. +32767) The default value is 0.
response:	none
example:	stat:ques:enab 128

command:	<b>:STATus:QUESTionable:ENABLE[:LEVel]?</b>
syntax:	:STATus:QUESTionable:ENABLE[:LEVel0]?
description:	Returns the QSESM for the event register
parameters:	none
response:	The bit value for the QSEM as a <i>16-bit signed integer</i> value (0 .. +32767)
example:	stat:ques:enab? → +128<END>

### 3 Instrument Setup and Status

command:	<b>:STATus<math>n</math>:QUESTionable:CONDition?</b>																																							
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 <i>16-bit signed integer</i> value, where $0 \leq \text{value} \leq 32767$ ):																																							
	<table border="1"> <thead> <tr> <th>Bit</th> <th>Mnemonic</th> <th>Decimal Value</th> </tr> </thead> <tbody> <tr> <td>11 - 15</td> <td>Not Used</td> <td></td> </tr> <tr> <td>10</td> <td>Slot <math>n</math>: Lambda zeroing is recommended</td> <td>1024</td> </tr> <tr> <td>9</td> <td>Slot <math>n</math>: Beam Path Protection on (shutter off)</td> <td>512</td> </tr> <tr> <td>8</td> <td>Slot <math>n</math>: Coherence control is uncalibrated</td> <td>256</td> </tr> <tr> <td>7</td> <td>Slot <math>n</math>: Duty cycle is out of range</td> <td>128</td> </tr> <tr> <td>6</td> <td>Slot <math>n</math>: ARA recommended</td> <td>64</td> </tr> <tr> <td>5</td> <td>Slot <math>n</math>: Module is out of specification</td> <td>32</td> </tr> <tr> <td>4</td> <td>Slot <math>n</math>: Module has not settled</td> <td>16</td> </tr> <tr> <td>3</td> <td>Slot <math>n</math>: Laser protection on</td> <td>8</td> </tr> <tr> <td>2</td> <td>Slot <math>n</math>: Temperature out of range</td> <td>4</td> </tr> <tr> <td>1</td> <td>Slot <math>n</math>: Zeroing failed</td> <td>2</td> </tr> <tr> <td>0</td> <td>Slot <math>n</math>: Excessive Value</td> <td>1</td> </tr> </tbody> </table>	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
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:	<b>:STATus<math>n</math>:QUESTionable:ENABle</b>
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. 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 <i>16-bit signed integer</i> value (0 .. +32767)
response:	none
example:	stat:ques:enab 128

command:	<b>:STATus<math>n</math>:QUESTionable:ENABle?</b>
syntax:	:STATus $n$ :QUESTionable:ENABle?
description:	Returns the QSSEM for slot $n$
parameters:	none
response:	The bit value for the QSSEM as a <i>16-bit signed integer</i> 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:	<b>:SYSTem:DATE</b>
syntax:	:SYSTem:DATE<wsp><year>,<month>,<day>
description:	The Multiport Power Meter has no internal clock. This command is included for backwards compatibility.

command:	<b>:SYSTem:DATE?</b>
syntax:	:SYSTem:DATE?
description:	The Multiport Power Meter has no internal clock. This query is included for backwards compatibility.
parameters:	none
response:	The date in the format year, month, day ( <i>16-bit signed integer</i> values)
example:	sys:date? → +0000,+0,+00<END>

command:	<b>:SYSTem:ERRor?</b>
syntax:	:SYSTem:ERRor?
description:	Returns the next error from the error queue (see " <a href="#">The Error Queue</a> " on page 13). Each error has the error code and a <i>short</i> description of the error, separated by a comma, for example 0, "No error". Error codes are numbers in the range -32768 and +32767. Negative error numbers are defined by the SCPI standard. Positive error numbers are device dependent.
parameters:	none
response:	The number of the latest error, and its meaning.
example:	sys:err? → -113,"Undefined header"<END>

command:	<b>:SYSTem:HELP:HEADers?</b>
syntax:	:SYSTem:HELP:HEADers?
description:	Returns a list of all GPIB commands.
parameters:	none
response:	Returns a list of all GPIB commands
example:	sys:help:head? → <i>Returns a list of all GPIB commands</i>

### 3 Instrument Setup and Status

command:	<b>:SYSTem:LXI:IDN</b>
syntax:	:SYSTem:LXI:IDN<wsp><boolean>
description:	This command starts or stops the LAN LED on the front panel blinking. This makes it easy to identify the unit associated with the address.
parameters:	boolean (0   1   off   on)
response:	none
example:	SYST:LXI:IDN ON

command:	<b>:SYSTem:PRESet</b>
syntax:	:SYSTem:PRESet
description:	Sets the mainframe and all installed modules to their standard settings. The following are not affected by this command: the GPIB, USB and LAN (interface) state, the interface addresses, the output and error queues, the Service Request Enable register (SRE), the Status Byte (STB), the Standard Event Status Enable Mask (SESEM), and the Standard Event Status Register (SESR).
parameters:	none
response:	none
example:	SYST:PRES

command:	<b>:SYSTem:REBoot</b>
syntax:	:SYSTem:REBoot
description:	Reboots the instrument
parameters:	none
response:	none
example:	SYST:REB

command:	<b>:SYSTem:TIME</b>
syntax:	:SYSTem:TIME<wsp><hour>,<minute>,<second>
description:	The Multiport Power Meter has no internal clock. This command is included for backwards compatibility.

command:	<b>:SYSTem:TIME?</b>
syntax:	:SYSTem:TIME?
description:	The Multiport Power Meter has no internal clock. This query is included for backwards compatibility.



parameters:	none
response:	The time in the format hour, minute, second. Hours are counted 0...23 ( <i>16-bit signed integer values</i> ).
example:	syst:time? → +00,+00,+00<END>

command:	<b>:SYSTem:VERSion?</b>
syntax:	:SYSTem:VERSion?
description:	Returns the SCPI revision to which the instrument complies.
parameters:	none
response:	The revision year and number.
example:	syst:vers? → 1990.0<END>

command:	<b>:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess</b>
syntax:	:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess<wsp><GPIB Address>
description:	Sets the GPIB address.
parameters:	The GPIB Address <span style="float: right;">Values allowed 0-30 21 is often reserved by the GPIB Controller.</span>
response:	none
example:	SYST:COMM:GPIB:ADDR 20

command:	<b>:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?</b>
syntax:	:SYSTem:COMMunicate:GPIB[:SELF]:ADDRess?
description:	Returns the GPIB address.
parameters:	none
response:	The GPIB Address
example:	SYST:COMM:GPIB:ADDR? → +20<END>

### **:SYSTem:COMMunicate:ETHernet subtree**

The optical power meter supports USB and LAN interfaces. The :SYSTem:COMMunicate:ETHernet subtree is only necessary for setting up the LAN (ETHernet).

When first delivered, DHCP is enabled.

If you do not want to use DHCP, or if it is not supported by your network, configure the network settings first over USB.

Although later changes can be made using the LAN interface, we recommend always changing ethernet parameters via USB connection, otherwise you may lose your connection.

The default host name is of the format

*A-PPPPPP-SSSSSS*

where, *PPPPPP* is the product number (for example, N7745A), and *SSSSSS* is the last six digits of the serial number. For example, A-N7745A-123456.

**Some notes on DHCP/AutoIP/DNS**

If DHCP is enabled but no DHCP server is found, the power meter tries to use AutoIP as a fallback. AutoIP can take some time (depending on timeout settings).

Depending on the available network capabilities the power meter tries to tell the DNS server its host name, or read the host name it has been assigned.

**MAC address**

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

command:	<b>:SYSTem:COMMunicate:ETHernet:MACaddress?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:MACaddress?
description:	Get the MAC address of the network adapter.
parameters:	none
response:	response string (hex value without a prefix or separators).
example:	:syst:comm:eth:mac? → "0007E014AE08"<END>

**Automatically set Ethernet parameters**

If DHCP/AutoIP is enabled, the optical power meter may use other parameters than specified explicitly, that is, it will use the parameters provided by the DHCP server. It tries to use its configured hostname (which may fail, depending on the network setup).

There will be an error if you try to query these parameters if the network is not connected, or before they have been set by the DHCP server.

command:	<b>:SYSTem:COMMunicate:ETHernet:DHCP:ENABle</b>
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:	<b>:SYSTem:COMMunicate:ETHernet:DHCP:ENABLE?</b>
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:	<b>:SYSTem:COMMunicate:ETHernet:IPAdDress:CURRent?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:IPAdDress:CURRent?
description:	Get the current IP address of the system.
parameters:	none
response:	string
example:	:syst:comm:eth:ipad:curr? → "192.132.13.2"<END>

command:	<b>:SYSTem:COMMunicate:ETHernet:SMASk:CURRent?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:SMASk:CURRent?
description:	Get the currently used subnet mask.
parameters:	none
response:	string
example:	example :syst:comm:eth:smas:curr? → "255.255.255.0"<END>

command:	<b>:SYSTem:COMMunicate:ETHernet:HOSTname:CURRent?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:HOSTname:CURRent?
description:	Get the current host name. The default host name is A-P..P-S..S; where A is for Agilent, P..P is the Product Number, and S..S is as many of the last digits of the Serial Number to get a 15 character hostname. For example: A-N7745A-012345
parameters:	none
response:	string
example:	:syst:comm:eth:host:curr? → "A-N7745A-123456"<END>

command:	<b>:SYSTem:COMMunicate:ETHernet:DOMainname:CURRent?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:DOMainname:CURRent?
description:	Get the currently used domain name

### 3 Instrument Setup and Status

parameters:	none
response:	string
example:	:syst:comm:eth:dom:curr? → “.companyame.com” <END>

command:	<b>:SYSTem:COMMunicate:ETHernet:DGATeway:CURRent?</b>
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>

#### Explicitly set Ethernet parameters

You must reboot the instrument or send a `SYST:COMM:ETH:REStart` command before any alterations to the Ethernet parameters become effective.

If you query one of the alterable parameters, you always get the most recently set value, even if you have not yet activated it.

To undo any changes before they become active, send `SYST:COMM:ETH:CANcel`.

command:	<b>:SYSTem:COMMunicate:ETHernet:HOSTname</b>
syntax:	:SYSTem:COMMunicate:ETHernet:HOSTname
description:	Set the host name
parameters:	string (maximum 79 characters, though not all characters can be used)
response:	none
example:	:syst:comm:eth:host “powermeter1”

command:	<b>:SYSTem:COMMunicate:ETHernet:HOSTname?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:HOSTname?
description:	Get the host name
parameters:	none
response:	string
example:	:syst:comm:eth:host? → “powermeter1” <END>

command:	<b>SYSTem:COMMunicate:ETHernet:DOMainname</b>
syntax:	:SYSTem:COMMunicate:ETHernet:DOMainname

description:	Set the domain name (used if DHCP is disabled)
parameters:	string (maximum 79 characters, though not all characters can be used)
response:	none
example:	:syst:comm:eth:dom ".companyname.com"

command:	<b>:SYSTem:COMMunicate:ETHernet:DOMainname?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:DOMainname?
description:	Get the domain name
parameters:	none
response:	string
example:	:syst:comm:eth:dom? → ".companyname.com" <END>

command:	<b>:SYSTem:COMMunicate:ETHernet:IPADdress</b>
syntax:	:SYSTem:COMMunicate:ETHernet:IPADdress
description:	Set the IP address of the system manually (used if DHCP is disabled).
parameters:	string (maximum 79 characters, though only integers and the period, ".", can be used)
response:	none
example:	:syst:comm:eth:ipad "192.132.13.2"

command:	<b>:SYSTem:COMMunicate:ETHernet:IPADdress?</b>
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:	<b>:SYSTem:COMMunicate:ETHernet:SMASk</b>
syntax:	:SYSTem:COMMunicate:ETHernet:SMASk
description:	Set the subnet mask.
parameters:	string (maximum 79 characters, though only integers and the period, ".", can be used)
response:	none
example:	:syst:comm:eth:smas "255.255.255.0"

command:	<b>:SYSTem:COMMunicate:ETHernet:SMASk?</b>
syntax:	:SYSTem:COMMunicate:ETHernet:SMASk?
description:	Get the subnet mask.
parameters:	none

### 3 Instrument Setup and Status

response:	string
example:	:syst:comm:eth:smas? → "255.255.255.0"<END>

command:	<b>:SYSTem:COMMunicate:ETHernet:DGATeway</b>
syntax:	:SYSTem:COMMunicate:ETHernet:DGATeway
description:	Set the default gateway.
parameters:	string (maximum 79 characters, though only integers and the period, ".", can be used)
response:	none
example:	:syst:comm:eth:dgat "192.168.101.11"

command:	<b>:SYSTem:COMMunicate:ETHernet:DGATeway?</b>
syntax:	SYSTem:COMMunicate:ETHernet:DGATeway?
description:	Get the default gateway.
parameters:	none
response:	string (maximum 79 characters)
example:	:syst:comm:eth:dgat? → "192.168.101.11"<END>

#### Changing the Ethernet parameters

In most cases, instead of using :SYSTem:COMMunicate:ETHernet:REStart, it is better to save the new parameters (:SYSTem:COMMunicate:SAVE) and reboot the instrument (:SYSTem:REBoot).

command:	<b>:SYSTem:COMMunicate:ETHernet:RESet</b>
syntax:	:SYSTem:COMMunicate:ETHernet:RESet
description:	Resets all the LAN parameters to the factory default. <ul style="list-style-type: none"><li>• DHCP On</li><li>• AutoIP On</li><li>• Hostname is a concatenation of product number and serial number.</li><li>• The password for the web based LAN configuration interface is reset to 'agilent'.</li></ul> This command is also triggered when the reset button on the front panel is pressed longer than 3 seconds.
parameters:	none
response:	none
example:	:syst:comm:eth:res

command:	<b>:SYSTem:COMMunicate:ETHernet:REStart</b>
syntax:	:SYSTem:COMMunicate:ETHernet:REStart

description:	Restart the system's network interface with the new parameters. This command only works if the instrument has a working network connection at the time the command is issued. If you are connected by USB, use :SYSTem:COMMunicate:ETHernet:SAVE followed by :SYSTem:REBoot.
parameters:	none
response:	none
example:	:syst:comm:eth:rest

command:	<b>:SYSTem:COMMunicate:ETHernet:SAVE</b>
syntax:	:SYSTem:COMMunicate:ETHernet:SAVE
description:	Save the system's network interface parameters.
parameters:	none
response:	none
example:	:syst:comm:eth:save

command:	<b>:SYSTem:COMMunicate:ETHernet:CANCel</b>
syntax:	:SYSTem:COMMunicate:ETHernet:CANCel
description:	Undo all changes to the network parameters that have been made since the last save, reboot or ":syst:comm:eth:restart" command.
parameters:	none
response:	none
example:	:syst:comm:eth:canc

## Handling Measurement Settings - The :CONFigure:MEASurement:SETting subtree

The instrument can store a number of settings in FLASH memory. The number of memory places can be queried with CONF:MEAS:SETT:NUMBer?.

A 'measurement setting consists of all parameters which can be set with the :SENSe:\* commands.

After each parameter change, you can use :SYSTem:ERRor? to check if it is OK.

command:	<b>:CONFigure:MEASurement:SETting:ACTual?</b>
syntax:	:CONFigure:MEASurement:SETting:ACTual?
description:	Get the index of the setting currently being used.
parameters:	none
response:	int A value >0 is returned if the setting has been stored in FLASH memory (using :CONFigure:MEASurement:SETting:SAVE), or has been recalled from FLASH memory (using :CONFigure:MEASurement:SETting:RECall), and has not been changed since. 0 is returned if the FLASH setting has been deleted (using:CONFigure:MEASurement:SETting:ERASe) since the last recall or store. 0 is also returned if the setting has not yet been stored. -1 is returned if the setting was changed but has not saved yet.
example:	:conf:meas:sett:act? → +2<END>

command:	<b>:CONFigure:MEASurement:SETting:NUMBer?</b>
syntax:	:CONFigure:MEASurement:SETting:NUMBer?
description:	Get the number of settings. In addition to the settings spaces in FLASH memory, the working memory can hold a setting.
parameters:	none
response:	int
example:	:conf:meas:sett:numb? → +1<END>

command:	<b>:CONFigure:MEASurement:SETting:PRESet</b>
syntax:	:CONFigure:MEASurement:SETting:PRESet
description:	Resets the setting values in the working memory
parameters:	none
response:	none
example:	:conf:meas:sett:pres



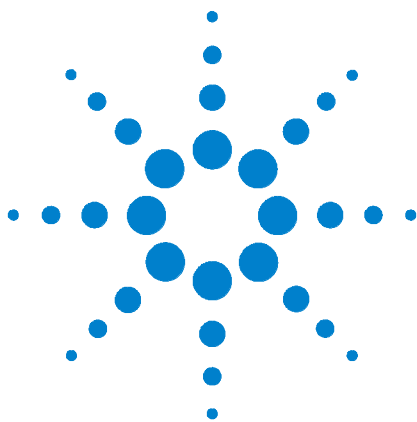
command:	<b>:CONFigure:MEASurement:SETTing:CANCEl</b>
syntax:	:CONFigure:MEASurement:SETTing:CANCEl
description:	Discard all the changes to the setting since the last save or recall
parameters:	none
response:	none
example:	:conf:meas:sett:canc

command:	<b>:CONFigure:MEASurement:SETTing:RECall</b>
syntax:	:CONFigure:MEASurement:SETTing:RECall
description:	Recall a setting from FLASH memory.
parameters:	int (setting index)
response:	none
example:	:conf:meas:sett:rec 1

command:	<b>:CONFigure:MEASurement:SETTing:SAVE</b>
syntax:	:CONFigure:MEASurement:SETTing:SAVE
description:	Save the current setting to FLASH memory.
parameters:	int (setting index)
response:	none
example:	:conf:meas:sett:sav 1

command:	<b>:CONFigure:MEASurement:SETTing:ERASe</b>
syntax:	:CONFigure:MEASurement:SETTing:ERASe
description:	Erase a setting from memory.
parameters:	int (setting index)
response:	none
example:	:conf:meas:sett:eras 1

### **3 Instrument Setup and Status**



## 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.
- TRIGGER subsystem commands that control triggering.

Root Layer Command 60

Measurement Functions – The FETCH, INITiate, READ and SENSE  
Subsystems 63

Triggering - The TRIGGER Subsystem 82



## Root Layer Command

The commands in the Slot subsystem allow you to query a particular port *n*, identified here as slot, for compatibility with the 816x platform.

For the same reason the command may optionally begin with `:SLOT[n][:HEAD[m]]`.

command:	<b>:SLOT[n]:EMPTY?</b>	
syntax:	:SLOT[n]:EMPTY?	
description:	Queries whether the module slot is empty.	
parameters:	none	
response:	A <i>boolean</i> value:	0: there is a module in the slot 1: the module slot is empty
examples:	slot1:empt? → 0<END>	There is a module in slot1
affects:	Independent of module type	

command:	<b>:SLOT[n]:IDN?</b>	
syntax:	:SLOT[n]:IDN?	
description:	Returns information about the module.	
parameters:	none	
response:	AGILENT:	manufacturer
	<i>mmmm</i> :	instrument model number (for example 81533B)
	<i>sssssss</i> :	serial number
	<i>rrrrrrrr</i> :	date of firmware revision
example:	slot1:idn? → AGILENT, N7744A,3411G06054,07-Jan-08<END>	
affects:	Independent of module type	

command:	<b>:SLOT[n]:OPTions?</b>	
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:	<b>:SLOT[n]:TST?</b>	
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:	<b>:SLOT[n]:HEAD[m]:WAVelength:RESPonse?</b>
syntax:	:SLOT[n]:HEAD[m]:WAVelength:RESPonse?
description:	Returns the wavelength response from a wavelength calibrated module in binary format.
parameters:	none
response:	Wavelength Response table as a binary block.  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.  For more information on binary block formats see "Data Types" on page 16
example:	slot1:head1:wav:resp? → #536570.....
affects:	Independent of module type

command:	<b>:SLOT[n]:HEAD[m]:WAVelength:RESPonse:CSV?</b>
syntax:	:SLOT[n]:HEAD[m]:WAVelength:RESPonse:CSV?
description:	Returns the wavelength response from the attenuator module in CSV format.
parameters:	none
response:	Wavelength Response table as a string  The string is a comma separated value (CSV) list and can be written to a file and be processed with a spreadsheet program.  List format: λ1, c1\n λ2, c2\n ..... λn, cn\n ", " separates wavelength and response factor "\\n" (= ASCII code 10) separates value pairs
example:	slot1:head1:wav:resp:csv? → 1200e-6,2.019\n 1210e-6, 1.956\n...
affects:	Independent of module type

command:	<b>:SLOT[n]:HEAD[m]:WAVelength:RESPonse:SIZE?</b>
syntax:	:SLOT[n]:HEAD[m]:WAVelength:RESPonse:SIZE?

## 4 Measurement Operations & Settings

description:	Returns the number of elements in the wavelength response table.
parameters:	none
response:	Number of elements in the wavelength table as an integer value
example:	slot2:head1:wav:resp:size? → 50<END>
affects:	Independent of module type

## Measurement Functions – The FETCh, INITiate, READ and SENSE Subsystems

These subsystems let you control measurement parameters for the power meter. Generally the parameter *n* refers to the optical port number. The Channel parameter *m* is not needed and only included for backward compatibility.

command:	<b>:FETCh[<i>n</i>][:CHANnel[<i>m</i>]][:SCALar]:POWer[:DC]?</b>
syntax:	:FETCh[ <i>n</i> ][:CHANnel[ <i>m</i> ]][:SCALar]:POWer[:DC]?
description:	<p>Reads the current power meter value. It does not provide its own triggering and so must be used with either continuous software triggering (see “:INITiate[<i>n</i>][:CHANnel[<i>m</i>]]:CONTinuous?” on page 65) or a directly preceding immediate software trigger (see “:INITiate[<i>n</i>][:CHANnel[<i>m</i>]][:IMMediate]” on page 64).</p> <p>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.</p>
parameters:	none
response:	<p>The current value as a <b>float</b> value in dBm,W or dB.</p> <p>If the reference state is absolute, units are dBm or W.</p> <p>If the reference state is relative, units are dB.</p>
example:	fetc1:pow? → +6.73370400E-04<END>

command:	<b>:FETCh[<i>n</i>][:CHANnel[<i>m</i>]][:SCALar]:POWer:ALL?</b>
syntax:	:FETCh[ <i>n</i> ][:CHANnel[ <i>m</i> ]][:SCALar]:POWer[:DC]:ALL?
description:	<p>Reads all current power meter values. It does not provide its own triggering and so must be used with either continuous software triggering (see “:INITiate[<i>n</i>][:CHANnel[<i>m</i>]]:CONTinuous?” on page 65) or a directly preceding immediate software trigger (see “:INITiate[<i>n</i>][:CHANnel[<i>m</i>]][:IMMediate]” on page 64).</p> <p>It returns the value the previous software trigger measured. Any subsequent FETCh command will return the same values, if there is no subsequent software trigger.</p> <p>The power meters must be running for this command to be effective.</p>
parameters:	none
response:	<p>4-byte Intel <i>float</i> values in a binary block in Intel byte order. The values are ordered by slot. See “Data Types” on page 16 for more information on Binary Blocks.</p> <p>Data values are always in Watt.</p>
example:	fetc:pow:all? → interpreted as +1.33555600E-006   +1.34789100E-006   +1.37456900E-006<END>

command:	<b>:FETCh[<i>n</i>][:CHANnel[<i>m</i>]][:SCALar]:POWer:ALL:CSV?</b>
syntax:	:FETCh[ <i>n</i> ][:CHANnel[ <i>m</i> ]][:SCALar]:POWer[:DC]:ALL:CSV?

## 4 Measurement Operations & Settings

description:	<p>Reads all current power meter values. 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 65) or a directly preceding immediate software trigger (see “:INITiate[n]:CHANnel[m]][:IMMediate]” on page 64).</p> <p>It returns the value the previous software trigger measured. Any subsequent FETCh command will return the same values, if there is no subsequent software trigger.</p> <p>The power meters must be running for this command to be effective.</p>
parameters:	none
response:	string containing the power values from each available channel in a comma separated format.
example:	Data values are always in Watt. fetc:pow:all:CSV? → “+1.33555600E-06, +1.34789100E-06, +1.37456900E-06” <END>

command:	<b>:INITiate[n]:CHANnel[m]][:IMMediate]</b>
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

command:	<b>:INITiate[n]:CHANnel[m]:CONTinuous</b>
syntax:	:INITiate[n]:CHANnel[m]:CONTinuous<wsp><boolean>
description:	Sets the software trigger system to continuous measurement mode.
parameters:	A <i>boolean</i> value:           0 or OFF: do not measure continuously 1 or ON: measure continuously
response:	none
example:	init2:cont 1



command:	<b>:INITiate[n]:[CHANnel[m]]:CONTInuous?</b>
syntax:	:INITiate[n]:[CHANnel[m]]:CONTInuous?
description:	Queries whether the software trigger system operates continuously or not
parameters:	none
response:	A <i>boolean</i> value:            0 or OFF: measurement is not continuous 1 or ON: measurement is continuous
example:	init2:cont? → 1<END>

command:	<b>:READ[n]:[CHANnel[m]][:SCALar]:POWer:ALL?</b>
syntax:	:READ[n]:[CHANnel[m]][:SCALar]:POWer[:DC]:ALL?
description:	Reads all available power channels. It provides its own software triggering and does not need a triggering command.
parameters:	none
response:	4-byte Intel <i>float</i> values in a binary block in Intel byte order. The values are ordered by slot. See "Data Types" on page 16 for more information on Binary Blocks.  Data values are always in Watt.
example:	read:pow:all? → interpreted as +1.33555600E-006   +1.34789100E-006   +1.37456900E-006<END>

command:	<b>:READ[n]:[CHANnel[m]]:POWer:ALL:CONFig?</b>
syntax:	:READ[n]:[CHANnel[m]]:POWer[:DC]:ALL:CONFig?
description:	Returns the slot numbers for all available power meters. Use this command to match returned power values to the appropriate slot.
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 slot number, the second member of the pair represents the channel number. The channel number is always 1.
example:	read1:pow:all:conf? → interpreted as 1 1 2 1 3 1 4 1<END>  This 16-byte block means that there are four powermeters present

command:	<b>:READ[n]:[CHANnel[m]][:SCALar]:POWer[:DC]?</b>
syntax:	:READ[n]:[CHANnel[m]][:SCALar]:POWer[:DC]?

## 4 Measurement Operations & Settings

description:	<p>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.</p> <p>If the software trigger system operates continuously (see “:INITiate[n]:[CHANnel[m]]:CONTinuous?” on page 65), this command is identical to “:FETCh[n]:[CHANnel[m]][:SCAlar]:POWer[:DC]?” on page 63.</p> <p>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 64) and then reading the power meter value.</p> <p>The power meter must be running for this command to be effective.</p>
parameters:	none
response:	<p>The current power meter reading as a <i>float</i> value in dBm, W or dB.</p> <p>If the reference state is absolute, units are dBm or W.</p> <p>If the reference state is relative, units are dB.</p>
example:	read1:pow? → +1.33555600E-006<END>

command:	<b>:SENSe[n]:[CHANnel[m]]:CORRection[:LOSS][:INPut][:MAGNitude]</b>
syntax:	:SENSe[n]:[CHANnel[m]]:CORRection[:LOSS][:INPut][:MAGNitude]<wsp><value>[DB   MDB]
description:	Enters a calibration value for a module.
parameters:	<p>The calibration factor as a <i>float</i> value</p> <p>If no unit type is specified, decibels (dB) is implied.</p>
response:	none
example:	sens1:corr 10DB

command:	<b>:SENSe[n]:[CHANnel[m]]:CORRection[:LOSS][:INPut][:MAGNitude]?</b>
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 <i>float</i> value. Units are in dB, although no units are returned in the response message.
example:	sens1:corr? → +1.00000000E+000<END>

command:	<b>:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO</b>
syntax:	:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO
description:	Zeros the electrical offsets for a power meter.
parameters:	none
response:	none
example:	sens1:corr:coll:zero

command:	<b>:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO?</b>
syntax:	:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO?
description:	Returns the status of the most recent zero command.
parameters:	none
response:	0: zero succeeded without errors. any other number: remote zeroing failed (the number is the error code returned from the operation).
example:	sens1:corr:coll:zero? → 0<END>

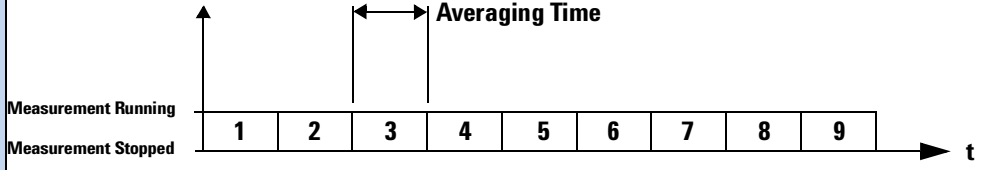
command:	<b>:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO:ALL</b>
syntax:	SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO:ALL
description:	Zeros the electrical offsets for all installed power meters.
parameters:	none
response:	none
example:	sens:chan:corr:coll:zero:all

command:	<b>:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO:ALL?</b>
syntax:	:SENSe[n]:[CHANnel[m]]:CORRection:COLLect:ZERO:ALL?
description:	Returns the status of the most recent zero all command. The result is backed up in the nonvolatile RAM. Note: If a channel fails to zero, it continues to use the result of the last successful zeroing.
parameters:	none
response:	A hexadecimal integer value which represents the result for all channels. Each hexadecimal digit represents one channel . <ul style="list-style-type: none"> <li>• 0: zero succeeded without errors.</li> <li>• Any other number: remote zeroing failed (the number is the error code returned from the operation).</li> </ul>
example:	sens:chan:corr:coll:zero:all? → 272 272 decimal = 0x110. This means zeroing failed on channels 2 and 3. All other channels were successful.  Alternatively the result can be found with the individual commands: sens1:chan:corr:coll:zero? → 0 sens2:chan:corr:coll:zero? → 1 sens3:chan:corr:coll:zero? → 1

**NOTE**

Setting parameters for the logging function sets some parameters, including hidden parameters, for the stability and MinMax functions and vice versa. You must use the :SENSe[n]:[CHANnel[m]]:FUNctioN:PARAmeter:LOGGing command to set parameters before you start a logging function using the :SENSe[n]:[CHANnel[m]]:FUNctioN:STATe command.

## 4 Measurement Operations & Settings

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:LOGGing</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:LOGGing<wsp><data points>, <averaging time>[NS US MS S]
description:	Sets the number of data points and the averaging time for the logging data acquisition function.
parameters:	<p>Data Points: Data Points is the number of samples that are recorded before the logging mode is completed. Data Points is an <b>integer</b> value.</p> <p>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 70 if you want to use delayed measurement.</p> 
response:	none
example:	sens1:func:par:logg 64,1ms

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:LOGGing?</b>
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 <b>integer</b> value and the averaging time, $t_{avg}$ , as a <b>float</b> value in seconds.
example:	sens1:func:par:logg? → +64,+1.00000000E-001<END>

### NOTE

Setting parameters for the MinMax function sets some parameters, including hidden parameters, for the stability and logging functions and vice versa. You must use the :SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:MINMax command to set parameters before you start a MinMax function using the :SENSe[n][:CHANnel[m]]:FUNction:STABility command.

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:MINMax</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:MINMax<wsp> CONTinuous   WINDow   REFResh,<data points>
description:	Sets the MinMax mode and the number of data points for the MinMax data acquisition function.
parameters:	<p>CONTinuous: continuous MinMax mode</p> <p>WINDow: window MinMax mode</p> <p>WINDow mode has the same function as REFResh mode. It is included to ensure compatibility.</p> <p>REFResh: refresh MinMax mode</p> <p>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 <b>integer</b> value.</p> <p>See Chapter 3 of the Agilent N7744A / N7745A Multiport Power Meter User's Guide, for more information on MinMax mode.</p> <p>See ":SENSe[n][:CHANnel[m]]:FUNction:STATe" on page 74 for information on starting/stopping a data acquisition function.</p> <p>See ":SENSe[n][:CHANnel[m]]:FUNction:RESult?" on page 73 for information on accessing the results of a data acquisition function.</p> <p>See "Triggering and Power Measurements" on page 82 for information on how triggering affects data acquisition functions.</p>
response:	none
example:	sens1:func:par:minm WIND,10

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:MINMax?</b>
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:	<p>CONT: continuous MinMax mode</p> <p>WIND: window MinMax mode</p> <p>WINDow mode has the same function as REFResh mode. It is included to ensure compatibility.</p> <p>REFR: refresh MinMax mode</p> <p>The number of data points is returned as an <b>integer</b> value.</p>
example:	sens1:func:par:minm? → WIND,+10<END>

**NOTE**

Setting parameters for the stability function sets some parameters, including hidden parameters, for the logging and MinMax functions and vice versa. You must use the :SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:STABility command to set parameters before you start a stability function using the :SENSe[n][:CHANnel[m]]:FUNction:STATe command.

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:STABility</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:STABility<wsp> <total time>[NS US MS S],<period time>[NS US MS S],<averaging time>[NS US MS S]
description:	Sets the total time, period time, and averaging time for the stability data acquisition function.
parameters:	<p>Total time: The total time from the start of stability mode until it is completed.</p> <p>Period time: A new measurement is started after the completion of every period time.</p> <p>Averaging time: A measurement is averaged over the averaging time.</p>
	<p>The total time should be longer than the period time.          The period time should be longer than the averaging time.          The number of data points is equal to the total time divided by the period time.          Total time, period time, and averaging time are time values in seconds.          If you specify no units in your command, seconds are used as the default.</p> <p>See “:SENSe[n][:CHANnel[m]]:FUNction:STATe” on page 74 for information on starting/stopping a data acquisition function.</p> <p>See “:SENSe[n][:CHANnel[m]]:FUNction:RESult?” on page 73 for information on accessing the results of a data acquisition function.</p> <p>See “<a href="#">Triggering and Power Measurements</a>” on page 82 for information on how triggering affects data acquisition functions.</p>
response:	none
example:	sens1:func:par:stab 1s,0.1s,0.1s

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:STABility?</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:STABility?
description:	Returns the total time, period time, and averaging time for the stability data acquisition function.
parameters:	none
response:	Total time, delay time, and averaging time are float values in seconds.
example:	sens1:func:par:stab? → +1.00000000E+000, +1.00000000E-001,+1.00000000E-001<END>

### Using data buffers for simultaneous measurement and upload

Enhanced logging functionality is available using two data buffers for each port. The MPPM ensures there is no time lag between logging and data availability by using a data buffer with a capacity of 1M samples for each channel. You can read the most recent results out of one buffer while the next logging

measurement is filling the other buffer.

You can use this to speed up applications with repeated logging operations. It is especially valuable for monitoring signals over extended periods to detect transient events.

To use the data buffers, set the logging function to perform a fixed or indefinite number of "LOOP"s (available from firmware versions 1.11 and later).

- When LOOP = 1, default logging behavior is selected, that is measurements are made and the results are written to the first buffer.
- If LOOP = 0, logging continues, writing the results into alternate buffers, until you stop it.
- When LOOP =  $n > 1$ , then upon completion of a logging measurement, another is started and the results are written in rotation to the two buffers (first to buffer A, then buffer B, then buffer A, and so on). This is repeated until  $n$  logging measurements have been performed. For example, if  $n=2$ , both buffers are filled, and you can upload a total of 2M samples for the channel.

You can also use input triggers to begin individual loops.

The "Index" value is updated to indicate that data from each LOOP is available for readout.

See also:

:SENSe[n][:CHANnel[m]]:FUNCtion:RESult:INDEX?

:SENSe[n][:CHANnel[m]]:FUNCtion:RESult:BUFA?

:SENSe[n][:CHANnel[m]]:FUNCtion:RESult:BUFB?

### Example: Programming Streaming Data

#### 1 Set Loop Parameter

SENS1:FUNC:LOOP 0 - For unlimited loops

#### 2 Configuration: for all ports n

SENSn:FUNC:STAT LOGG,STOP - Cleaning up, in case the last run didn't finish

INITn:CONT 0 - Disable continuous triggering

SENSn:POW:RANG xDBM - Choose range

SENS:POW:UNIT 1 - Fastest upload with units Watts

SENSn:FUNC:PAR:LOGG x,y us - Enter logging samples and avg. time

## 4 Measurement Operations & Settings

- 3 Enable Logging: for all ports n  
 SENSn:FUNC:STAT LOGG,STAR - Starts logging, synchronized at start of final port
  
- 4 Check for Measurement Finished  
 SENSn:FUNC:RES:INDEX? - Index incremented when loop finishes
  
- 5 Get Logging results for all ports  
 SENSn:FUNC:RES? - Data in binary block form, "LSB" byte ordering

Stop Loops when finished  
 SENS1:FUNC:LOOP 1

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:LOOP</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:LOOP<wsp><value>
description:	Sets the number of logging loops
parameters:	Number of Loops, an integer value. <ul style="list-style-type: none"> <li>• 0 = Endless Streaming</li> <li>• 1 = 1 (Default)</li> <li>• 2 = 2 (For 2 Million Points with Buffer A and B) ....</li> <li>• n</li> </ul>
response:	none
example:	SENS1:FUNC:LOOP 0

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:LOOP?</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:LOOP?
description:	Gets the number of logging loops. For details look at description and example in <a href="#">"Using data buffers for simultaneous measurement and upload"</a> on page 70.
parameters:	none
response:	Number of loops: Number of loops is an integer value. <ul style="list-style-type: none"> <li>• 0 = Endless Streaming</li> <li>• 1 = 1 (Default)</li> <li>• 2 = 2 (For 2 Million Points with Buffer A and B) ....</li> <li>• n</li> </ul>
example:	:SENS1:FUNC:LOOP? → 0



command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:RESult:INDEX?</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:RESult:INDEX?
description:	Gets the number of already finished logging loops. For details look at description and example in <a href="#">"Using data buffers for simultaneous measurement and upload"</a> on page 70.
parameters:	none
response:	Number of loops: Number of loops is an integer value.
example:	sens1:func:res:index? → 1

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:RESult?</b>
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. For Logging and Stability Data Acquisition functions, one measurement value is a 4-byte-long <b>float</b> in Intel byte order. For the MinMax Data Acquisition function, the query returns the minimum, maximum and current power values. See <a href="#">"Data Types"</a> on page 16 for more information on Binary Blocks. See <a href="#">"How to Log Results"</a> on page 91 for information on logging using VISA calls. There are some tips about how to use float format specifiers to convert the binary blocks into <b>float</b> values. If you use LabView or Agilent VEE, we recommend using the Agilent N7744A / N7745A <i>VXIplug&amp;play</i> Instrument Driver to perform the Logging and Stability Data Acquisition functions.
example:	sens1:func:res? → returns a data array for Logging and Stability Data Acquisition functions sens1:func:res? → #255 Min: 7.24079E-04, Max: 7.24252E-04, Act: 7.24155E-04 returns the minimum, maximum and current power values for the MinMax Data Acquisition function

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:RESult:BUFA?</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:RESult:BUFA?
description:	Returns the data array of the last data acquisition function in Buffer A. This works only for Logging and Stability Data Acquisition in the loop mode, not for the MinMax Data Acquisition. For details look at description and example in <a href="#">"Using data buffers for simultaneous measurement and upload"</a> on page 70
parameters:	none

## 4 Measurement Operations & Settings

response:	<p>The last data acquisition function's data array as a binary block.</p> <p>For Logging and Stability Data Acquisition functions, one measurement value is a 4-byte-long float in Intel byte order.</p> <p>See <a href="#">"Data Types"</a> on page 16 for more information on Binary Blocks.</p> <p>See <a href="#">"How to Log Results"</a> on page 91 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.</p>
example:	sens1:func:res:bufa? → #255

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:RESult:BUFB?</b>
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:RESult:BUFB?
description:	<p>Returns the data array of the last data acquisition function in Buffer B.</p> <p>This works only for Logging and Stability Data Acquisition in the loop mode, not for the MinMax Data Acquisition.</p> <p>For details look at description and example in <a href="#">"Using data buffers for simultaneous measurement and upload"</a> on page 70</p>
parameters:	none
response:	<p>The last data acquisition function's data array as a binary block.</p> <p>For Logging and Stability Data Acquisition functions, one measurement value is a 4-byte-long float in Intel byte order.</p> <p>See <a href="#">"Data Types"</a> on page 16 for more information on Binary Blocks.</p> <p>See <a href="#">"How to Log Results"</a> on page 91 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.</p>
example:	sens1:func:res:bufb? → #255

command:	<b>:SENSe[n][:CHANnel[m]]:FUNction:STATe</b>										
syntax:	:SENSe[n][:CHANnel[m]]:FUNction:STATe<wsp> LOGGing STABility MINMax,STOP STARt										
description:	Enables/Disables the logging, MinMax, or stability data acquisition function mode.										
parameters:	<table> <tr> <td>LOGGing:</td> <td>Logging data acquisition function</td> </tr> <tr> <td>STABility:</td> <td>Stability data acquisition function</td> </tr> <tr> <td>MINMax:</td> <td>MinMax data acquisition function</td> </tr> <tr> <td>STOP:</td> <td>Stop data acquisition function</td> </tr> <tr> <td>STARt:</td> <td>Start data acquisition function</td> </tr> </table> <p>See <a href="#">":SENSe[n][:CHANnel[m]]:FUNction:PARAmeter:LOGGing"</a> on page 68 for more information on the logging data acquisition function.</p> <p>Stop any functions on all channels before you try to set up a new function. Some parameters cannot be set until you stop the function.</p>	LOGGing:	Logging data acquisition function	STABility:	Stability data acquisition function	MINMax:	MinMax data acquisition function	STOP:	Stop data acquisition function	STARt:	Start data acquisition function
LOGGing:	Logging data acquisition function										
STABility:	Stability data acquisition function										
MINMax:	MinMax data acquisition function										
STOP:	Stop data acquisition function										
STARt:	Start data acquisition function										
response:	none										
example:	sens1:func:stat logg,star										

command:	<b>:SENSe[n]:CHANnel[m]:FUNction:STATe?</b>
syntax:	:SENSe[n]:CHANnel[m]:FUNction:STATe?
description:	Returns the function mode and the status of the data acquisition function.
parameters:	none
response:	NONE                                   No function mode selected LOGGING_STABILITY                    Logging or stability data acquisition function MINMAX                                   MinMax data acquisition function PROGRESS                                Data acquisition function is in progress COMPLETE                                Data acquisition function is complete
example:	sens1:func:stat? → LOGGING_STABILITY,COMPLETE<END>

command:	<b>:SENSe[n]:CHANnel[m]:POWer:ATIMe</b>
syntax:	:SENSe[n]:CHANnel[m]:POWer:ATIMe<wsp><averaging time>[NS US MS S]
description:	Sets the averaging time for the module.
parameters:	The averaging time as a float value in seconds. If you specify no units in your command, seconds are used as the default.
response:	none
example:	sens1:pow:atim 1s

command:	<b>:SENSe[n]:CHANnel[m]:POWer:ATIMe?</b>
syntax:	:SENSe[n]:CHANnel[m]:POWer:ATIMe?
description:	Returns the averaging time for the module.
parameters:	none
response:	The averaging time as a <i>float</i> value in seconds.
example:	sens1:pow:atim? → +1.00000000E+000<END>

command:	<b>:SENSe[n]:CHANnel[m]:POWer:RANGe:AUTO</b>
syntax:	SENSe[n]:CHANnel[m]:POWer:RANGe:AUTO <wsp><boolean>
description:	Enables or disables automatic power ranging for the module. If automatic power ranging is enabled, ranging is automatically determined by the instrument. Otherwise, it must be set by the sensn:pow:rang command.
parameters:	A <i>boolean</i> value:                    0 or OFF: automatic ranging disabled 1 or ON: automatic ranging enabled
response:	none
example:	sens1:pow:rang:auto 1

command:	<b>:SENSe[n]:CHANnel[m]:POWer:RANGe:AUTO?</b>
syntax:	:SENSe[n]:CHANnel[m]:POWer:RANGe:AUTO?
description:	Returns whether automatic power ranging is being used by the module.
parameters:	none

## 4 Measurement Operations & Settings

response:	A <i>boolean</i> value:	0: automatic ranging is not being used. 1: automatic ranging is being used.
example:	sens1:pow:rang:auto? → 1<END>	

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:RANGe[:UPPer]</b>												
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:RANGe[:UPPer]<wsp><value>[DBM]												
description:	<p>Sets the power range for the module.</p> <p>The range changes at 10 dBm intervals. The corresponding ranges for linear measurements (measurements in Watts) is given below:</p> <table border="1"> <thead> <tr> <th>Range</th> <th>Upper Linear Power Limit</th> </tr> </thead> <tbody> <tr> <td>+10 dBm</td> <td>19.999 mW</td> </tr> <tr> <td>0 dBm</td> <td>1999.9 <math>\mu</math>W</td> </tr> <tr> <td>-10 dBm</td> <td>199.99 <math>\mu</math>W</td> </tr> <tr> <td>-20 dBm</td> <td>19.999 <math>\mu</math>W</td> </tr> <tr> <td>-30 dBm</td> <td>1999.9 nW</td> </tr> </tbody> </table>	Range	Upper Linear Power Limit	+10 dBm	19.999 mW	0 dBm	1999.9 $\mu$ W	-10 dBm	199.99 $\mu$ W	-20 dBm	19.999 $\mu$ W	-30 dBm	1999.9 nW
Range	Upper Linear Power Limit												
+10 dBm	19.999 mW												
0 dBm	1999.9 $\mu$ W												
-10 dBm	199.99 $\mu$ W												
-20 dBm	19.999 $\mu$ W												
-30 dBm	1999.9 nW												
parameters:	The range as a <b>float</b> 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												

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:RANGe[:UPPer]?</b>
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:RANGe[:UPPer]?
description:	Returns the range setting for the module.
parameters:	none
response:	The range setting as a <i>float</i> value in dBm
example:	sens1:pow:rang? $\rightarrow$ -2.00000000E+001<END>

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:GAIN:AUTO</b>
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:GAIN:AUTO<wsp><value>
description:	Set the Auto Gain.
parameters:	<ul style="list-style-type: none"> <li>0 = Auto Gain Off. This is the position for best transient response.</li> <li>1 = Auto Gain On (Default) This is the Position for best dynamic.</li> </ul> <p>Auto gain only works for averaging times &gt; 10 <math>\mu</math>s. For shorter averaging times, the auto gain is always disabled.</p> <p>The Auto Gain setting works also in the logging and stability modes, where it also increases dynamic or enhances transient response.</p>
response:	none
example:	sens:pow:gain:auto 1

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:GAIN:AUTO?</b>
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:GAIN:AUTO?
description:	Get the Auto Gain status.

## 4 Measurement Operations & Settings

parameters:	none
response:	<ul style="list-style-type: none"> <li>• 0 = Auto Gain Off. This is the position for best transient response.</li> <li>• 1 = Auto Gain On (Default) This is the Position for best dynamic.</li> </ul>
example:	sens:pow:gain:auto? → 1

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:REFerence</b>
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:	<p>TOMODule:            Sets the reference value in dB used if you choose measurement relative to another slot</p> <p>TOREF:                Sets the reference value in Watts or dBm if you choose measurement relative to a constant reference value</p> <p>The reference as a <b>float</b> value.</p> <p>You must append a unit type</p> <ul style="list-style-type: none"> <li>• dB if you use TOMODule or</li> <li>• Watts or dBm if you use TOREF.</li> </ul> <p>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 79, the instrument uses the last reference value entered for the selected reference mode.</p>
response:	none
example:	sens1:pow:ref tomod,-40DB

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:REFerence?</b>
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:REFerence?<wsp>TOMODule   TOREF
description:	Returns the sensor reference value.
parameters:	<p>TOMODule:            Returns the reference value in dB used if you choose measurement relative to another slot</p> <p>TOREF:                Returns the reference value in Watts or dBm if you choose measurement relative to a constant reference value</p>
response:	The reference as a <b>float</b> value.
example:	sens1:pow:ref? toref → +1.00000000E-006<END>

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:REFerence:DISPlay</b>
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:REFerence:DISPlay
description:	Takes the input power level value as the reference value.
parameters:	none



## 4 Measurement Operations & Settings

response:	results are displayed relative to an absolute reference or to the current power reading from another slot.	
examples:	sens1:pow:ref:stat:rat? → +255,+0<END>	results are displayed relative to an absolute reference
	sens1:pow:ref:stat:rat? → +2,+1<END>	results are displayed relative to slot 2

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:UNIT</b>	
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:UNIT<wsp>DBM 0 Watt 1	
description:	Sets the sensor power unit	
parameters:	An <i>integer</i> value:           0: dBm 1: Watt	
	or DBM or Watt	
response:	none	
example:	sens1:pow:unit 1	

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:UNIT?</b>	
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:UNIT?	
description:	Queries the current sensor power unit	
parameters:	none	
response:	An <i>integer</i> value:           0: Current power units are dBm. 1: Current power units are Watts.	
example:	sens1:pow:unit? → +1<END>	

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:UNIT:ALL:CSV?</b>	
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:UNIT:ALL:CSV?	
description:	Queries the power unit for all ports of the instrument, returned in csv data format.	
parameters:	none	
response:	<i>Integer</i> values, separated by commas: <ul style="list-style-type: none"> <li>• 0: Current power units are dBm.</li> <li>• 1: Current power units are Watts.</li> </ul>	
example:	sens1:pow:unit:all:csv? → 1,1,1,1,1,1,1,1,<END>	

command:	<b>:SENSe[n]:[CHANnel[m]]:POWer:WAVelength</b>	
syntax:	:SENSe[n]:[CHANnel[m]]:POWer:WAVelength<wsp><value> MIN MAX DEF [PM NM UM MM M]	
description:	Sets the sensor wavelength.	



parameters:	The wavelength as a <i>float</i> value in meters. Also allowed are: MIN: minimum programmable value MAX: maximum programmable value DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable value and the maximum programmable value
response:	none
example:	sens1:pow:wav 1550nm
command:	<b>:SENSe[<i>n</i>]:[CHANnel[<i>m</i>]]:POWer:WAVelength?</b>
syntax:	:SENSe[ <i>n</i> ]:[CHANnel[ <i>m</i> ]]:POWer:WAVelength?[<wsp>MIN   MAX   DEF]
description:	Inquires the current sensor wavelength.
parameters:	none Also allowed are: MIN: minimum programmable value MAX: maximum programmable value DEF: This is not the preset (*RST) default value but is half the sum of, the minimum programmable value and the maximum programmable value
response:	The wavelength as a <i>float</i> value in meters.
example	sens1:pow:wav? → +1.55000000E-006<END>

## Triggering - The TRIGger Subsystem

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

**Table 1** Triggering and Power Measurements

Hardware Triggering trig:inp	Software Triggering		Data Acquisition Functions sens:func:stat	
	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	Every hardware trigger starts a new power measurement.			Every hardware trigger starts a new power measurement until the function is finished.
CMEasure				The first hardware trigger starts the function. Subsequent power measurements are automatically performed until the function is finished.

**Table 2** Generating Output Triggers from Power Measurements

Hardware Triggering trig:outp	Software Triggering		Data Acquisition Functions sens:func:stat	
	init:imm	init:cont	MINMax	LOGGing   STABility
DISabled	An output trigger will never be generated.			
AVGover	An output trigger is generated for every new power measurement when the averaging time period finishes.			Applies for all subsequent data acquisition functions.
MEASure	An output trigger is generated for every new power measurement when the averaging time period begins.			Applies for all subsequent data acquisition functions.

command:	<b>:TRIGger[n][:CHANnel[m]]:INPut</b>
syntax:	:TRIGger[n][:CHANnel[m]]:INPut<wsp><trigger response>
description:	Sets the incoming trigger response and arms the module.

parameters:	<p>IGNore: Ignore incoming trigger.</p> <p>SMEasure: Start a single measurement. If a measurement function is active, see “:SENSE[n]:CHANnel[m]:FUNCtion:STATe” on page 74, one sample is performed and the result is stored in the data array, see “:SENSE[n]:CHANnel[m]:FUNCtion:RESult?” on page 73.</p> <p>CMEasure: Start a complete measurement. If a measurement function is active, see “:SENSE[n]:CHANnel[m]:FUNCtion:STATe” on page 74, a complete measurement function is performed.</p> <p>You must prearm a measurement function before an action can be triggered: First, set the incoming trigger response. Then: prearm a measurement function using “:SENSE[n]:CHANnel[m]:FUNCtion:STATe” on page 74. NOTE: If a trigger signal arrives at the Input Trigger Connector at the same time that the :SENSE[n]:CHANnel[m]:FUNCtion:STATe command is executed, the first measurement value is invalid. You should always discard the first measurement value in this case. The module performs the appropriate action when it is triggered.</p>
response:	none
example:	<p>trig1:inp ign</p> <p>If you use the VXIplug&amp;play Instrument Driver, you can trigger power measurements.</p>

command:	<b>:TRIGger[n]:CHANnel[m]:INPut?</b>
syntax:	:TRIGger[n]:CHANnel[m]:INPut?
description:	Returns the incoming trigger response.
parameters:	none
response:	<p>IGNore: Ignore incoming trigger.</p> <p>SMEasure: Start a single measurement. If a measurement function is active, see “:SENSE[n]:CHANnel[m]:FUNCtion:STATe” on page 74, one sample is performed and the result is stored in the data array, see “:SENSE[n]:CHANnel[m]:FUNCtion:RESult?” on page 73.</p> <p>CMEasure: Start a complete measurement. If a measurement function is active, see “:SENSE[n]:CHANnel[m]:FUNCtion:STATe” on page 74, a complete measurement function is performed.</p>
example:	trig1:inp? → ign<END>

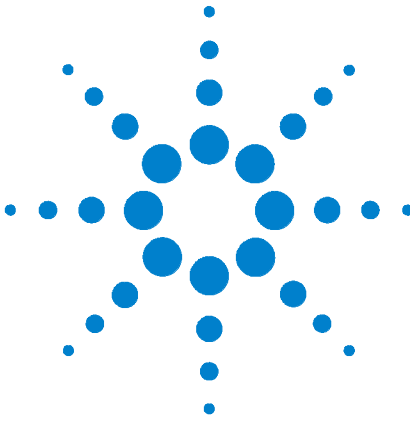
command:	<b>:TRIGger[n]:CHANnel[m]:OUTPut</b>
syntax:	:TRIGger[n]:CHANnel[m]:OUTPut
description:	Specifies when an output trigger is generated and arms the module.
parameters:	<p>DISabled: Never.</p> <p>AVGOver: When averaging time period finishes.</p> <p>MEASure: When averaging time period begins.</p>
response:	none
example:	trig1:outp dis

## 4 Measurement Operations & Settings

command:	<b>:TRIGger[n][:CHANnel[m]]:OUTPut?</b>
syntax:	:TRIGger[n][:CHANnel[m]]:OUTPut?
description:	Returns the condition that causes an output trigger.
parameters:	none
response:	DISabled: Never. AVGover: When averaging time period finishes. MEASure: When averaging time period begins.
example:	trig1:outp? → dis<END>

command:	<b>:TRIGger:CONFiguration</b>
syntax:	:TRIGger:CONFiguration<wsp><triggering mode>
description:	Sets the hardware trigger configuration with regard to Output and Input Trigger Connectors.
parameters:	0 or DISabled: Trigger connectors are disabled. 1 or DEFault: The Input Trigger Connector is activated, the incoming trigger response for each slot “:TRIGger[n][:CHANnel[m]]:INPut” on page 82 determines how each slot responds to an incoming trigger, all slot events (see “:TRIGger[n][:CHANnel[m]]:OUTPut” on page 83) 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 PASSthrough. This is included for compatibility reasons.
response:	none
example:	trig:conf dis

command:	<b>:TRIGger:CONFiguration?</b>
syntax:	:TRIGger:CONFiguration?
description:	Returns the hardware trigger configuration.
parameters:	none
response:	DIS: Trigger connectors are disabled. DEF: The Input Trigger Connector is activated, the incoming trigger response for each slot “:TRIGger[n][:CHANnel[m]]:INPut” on page 82 determines how each slot responds to an incoming trigger, all slot events (see “:TRIGger[n][:CHANnel[m]]:OUTPut” on page 83) 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 PASSthrough. This is included for compatibility reasons.
example:	trig:conf? → DEF<END>



## 5 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 Agilent N7744A / N7745A *VXIplug&play* Instrument Driver in the same program.

**TIP:** Additional programming examples are provided on the Support Disk CD-ROM 00000-00000

How to Use VISA Calls 86

How to Measure Power using FETCh and READ 88

How to Log Results 91



## 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 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 comparison */
    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);

```

## 5 VISA Programming Examples

```
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","%",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 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, specifying 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);
    }
}

```

## 5 VISA Programming Examples

```
/* 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%d:CHAN%d:FUNC:PAR:LOGG %d,%f\n",
    slot,
    chan,
    noOfValues,
    averagingTime);
    checkError(vi,errStatus);

/* start logging */
viPrintf(vi,"SENS%d:CHAN%d: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%d:CHAN%d: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(!strcmp(replySubStr,"COMPLETE",strlen("COMPLETE"))) break;
        replySubStr ++;
    }
}while (!*replySubStr); /*substring "COMPLETE" not found */
    /*continue polling */

/* The instrument returns the logging result in the following format: #xyyffff...; 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%d:CHAN%d: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\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

```

## 5 VISA Programming Examples

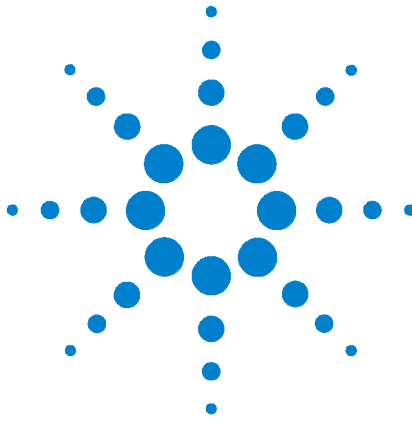
```
(!strcmp(errMsg,
"-303,\"Module slot empty or slot / channel invalid\"",
strlen("-303,\"Module slot empty or slot / channel invalid\"")))
||
(!strcmp(errMsg,
"-301,\"Module doesn't support this command (StatCmdUnknown)\"",
strlen(
"-301,\"Module doesn't support this command (StatCmdUnknown)\"")))
{
    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;
}
```



## 6 The Agilent 816x VXI*plug&play* Instrument Driver

### NOTE

The N7744A/45A is supported by the standard 816x Plug & Play Instrument Driver version 4.2.1 and higher.

IVI-COM and IVI-C drivers are also available for download at <http://www.agilent.com/find/ivi-com>

This chapter gives you extra information about installing and getting started with the Agilent 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 Agilent 816x Instrument Driver 96

Using Visual Programming Environments 100

    Getting Started with Agilent VEE 100

    Getting Started with LabView 102

    Getting Started with LabWindows 105

Features of the Agilent 816x Instrument Driver 106

Directory Structure 107

Opening an Instrument Session 108

Closing an Instrument Session 109

VISA Data Types and Selected Constant Definitions 110

Error Handling 111

Introduction to Programming 113

    Example Programs 113

    VISA-Specific Information 113

    Development Environments 113

Online Information 115

Lambda Scan Applications 116

    How to Perform a Multi-Frame Lambda Scan Application 118



## Installing the Agilent 816x Instrument Driver

The Agilent 816x VXI*plug&play* Instrument Driver comes as a self-extracting archive with an installation wizard. The installation wizard extracts all the files to preset destinations, asking you appropriate questions as it does so.

You install the driver by running the executable hp816x.exe.

- 1 Run hp816x.exe,

The welcome screen for the InstallShield Wizard used to install the Agilent 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 permission to overwrite existing files.

The version number of the instrument driver is displayed.

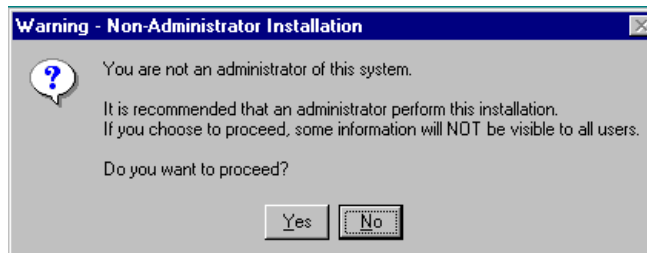
You may now elect to skip installation at this PC. Copy the extracted disk images to floppy, and use them to install the instrument driver at another PC.

- 4 Press OK> to continue.

If you are not an administrator, you see a VXI*plug&play* window, and a message telling you that if you proceed with the installation, some information will NOT be visible to all users. This means that any program menu options will only be available to the user that performed the installation. If you are the administrator all program menu options will be visible for all users.

If you see the message in Figure 1, press Yes to install the driver or press No and contact your administrator.





**Figure 1** Non-Administrator Installation Pop-Up Box

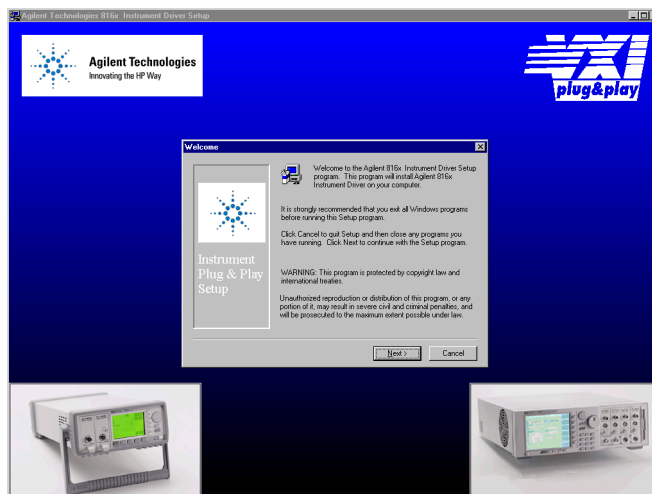
### NOTE

If Agilent 816x VXIplug&play Instrument Driver is already installed on your system, you see a message asking you if you want to uninstall the old version.

Press Yes, if required, then wait until you see a message telling you that the uninstall has been successful. You may be asked for permission to remove shared files.

Then press OK to continue.

- 5 You see a message, as shown in Figure 2, advising you to close the programs that you have running.



**Figure 2** Welcome Screen

- 6 Close these programs and press Next> to continue. Then, you see a message informing you if VISA is installed on your PC.

### NOTE

If you do not have VISA installed, press Cancel to temporarily exit this installation procedure; install VISA on your PC, then run hp816x.exe again.

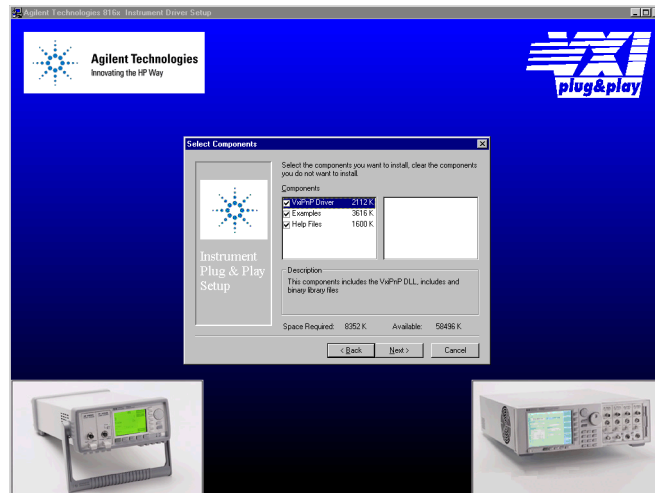
If you have VISA installed, press Next> to continue. You see a window that requests you to choose your Setup.

- 7 You can choose a Typical, Compact, or Custom Setup. Choose a setup option and press Next> to continue.

**NOTE**

If you choose the Custom Setup, you may choose the options you want to install from the screen in Figure 3. These options are:

- VxiPnP Driver, you may choose to install the Agilent 816x VXIplug&play instrument driver.
- Examples, you may choose to install Visual Basic, Visual C, LabView, Agilent VEE and VISA programming examples.
- Help Files, you may choose to install the help file.



**Figure 3** Customizing Your Setup

Select the components you want to install.

- 8 Press Next> to continue.

Specify the program folder required; the default choice is VXIPNP.

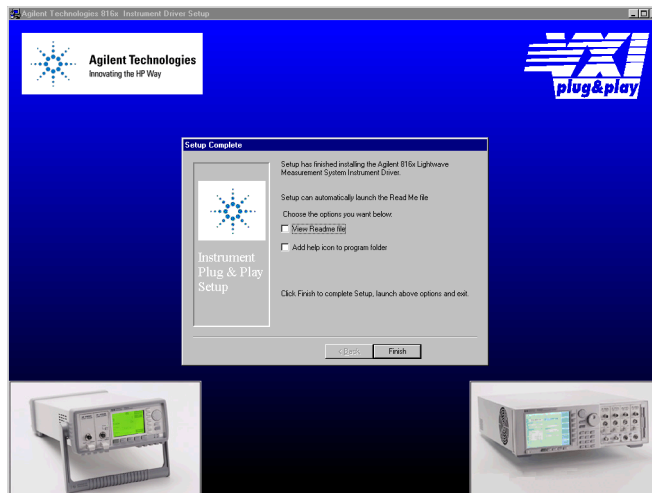
- 9 Press Next> to continue.

Review the settings that you have specified.

If you want to review or change any settings press Back>

- 10 Press Next> to continue.

The instrument driver is installed.



**Figure 4** Program Folder Item Options

You may elect to:

- Automatically launch the Readme file, which provides the instrument driver's version history
- Include a help icon in your program folder, which launches on-line documentation for the instrument driver

#### 11 Press Finish to complete installation

If you elected to automatically launch the Readme file, it is displayed.

A webpage explaining how to get started with the Agilent 816x VXIplug&play Instrument Driver using Agilent VEE or LabView appears.

## Using Visual Programming Environments

### Getting Started with Agilent VEE

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

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

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

Agilent VEE supports *VXIplug&play* drivers in the WIN, WIN95, WINNT, and Agilent-UX frameworks. In addition, versions 3.2 and above of Agilent VEE support the graphical Function Panel interface, providing a function tree of the hierarchy of the driver.

#### NOTE

This appendix assumes that you are using Windows 95. If you are using Windows NT, please replace every reference to win95 with winnt. Windows 95 and Windows NT are registered trademarks of Microsoft corporation.

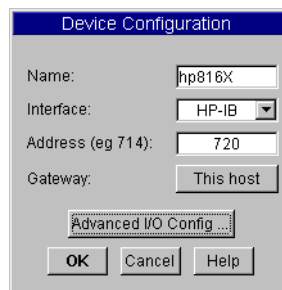
Agilent VEE automatically calls the *initialize* and *close* functions to perform automatic error checking.

---

## GPIB Interfacing in Agilent VEE

Agilent 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 5.



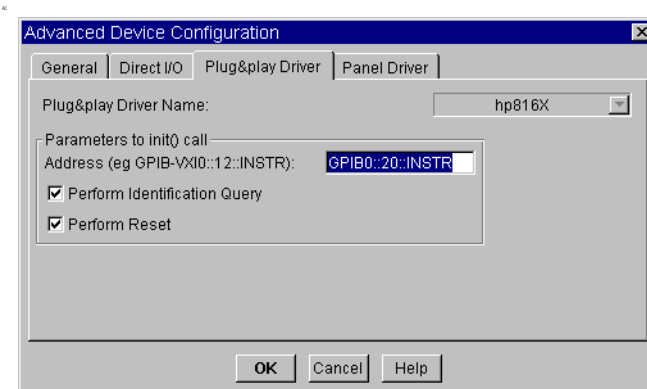
**Figure 5** Device Configuration

- 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 6 appears.



**Figure 6** Advanced Device Configuration - Plug&play Driver

- 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.

- 5 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.

- 6 Select whether to Perform Reset or to Perform Identification Query whenever Agilent VEE opens the instrument for interaction.
- 7 Confirm the selections pressing the OK button.
- 8 Return to the Instrument Manager screen and press the Save Config to save the configuration.

## Getting Started with LabView

The 32-bit Agilent 816x driver can be used with LabView 5.0 and above. LabView 5.0 is a 32-bit version of LabView which runs on Windows 95 and Windows NT.

After installing the Agilent 816x instrument driver, the driver must be converted for use with LabView.

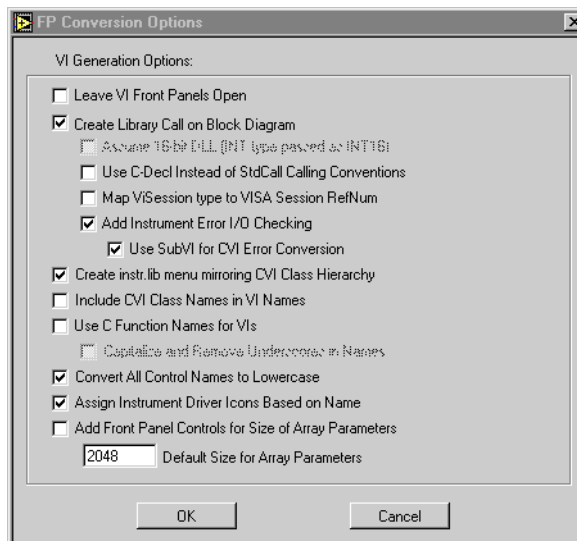
- 1 To convert the driver follow these steps:
  - a If you are updating from a previously installed driver, perform the following three steps:
  - b Locate the LabView program folder. By default, this is <drive>:\Program Files\National Instruments\LabView.
  - c This folder contains a subfolder named instr.lib.
- 2 Run LabView.
- 3 On the first window that appears, click on the Solution Wizards button.
- 4 The LabView Solution Wizard window appears, click on the Launch Wizard... button.
- 5 The Welcome to Instrument Wizard! window appears, click on the Next > button.
- 6 The Search for Instruments... window appears, click on the Next > button. Check that the options are the same as displayed in the figure below:



**Figure 7** Search for GPIB Instruments

- 7 Click on the Next > button.
- 8 The Identify Found Instruments... window appears, click on the Next > button.

- 9 The Update VXI Plug and Play Drivers window appears, select HP816x, and click on the Convert button.
- 10 The *Manage Instrument Drivers* window appears, click on the Finish button.
- 11 The first window appears again, click on the New VI button.
- 12 Select File and then select Convert CVI FP file.
- 13 The Select a CVI Function Panel file window appears, locate the hp816x.fp file, which is normally installed into the path <drive>:\VXIPNP\winXX\hp816x, where XX stands for NT, or 95.
- 14 Press Open.
- 15 The CVI Function Panel Converter window appears.
- 16 Click on Browse... and browse to the following Destination Directory: \LabView\instr.lib\hp816x\hp816x.llb
- 17 Press Save.
- 18 Press Options..., the FP Conversion Options window appears. Check that the options are the same as displayed in the figure below:.



**Figure 8** 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.



- 19 Press OK. The CVI Function Panel Converter window appears.
- 20 Press OK.
- 21 The Select a library window appears. Browse to <drive>:\vxiinp\winXX\Bin, where XX stands for NT, or 95, select hp816x\_32.dll and click on Open.
- 22 The CVI Conversion Status window is displayed until the conversion is completed.

**NOTE**

You must use the 32-bit version of the Agilent 816x VXI*plug&play* Instrument Driver with LabView 5.0.

---

**NOTE**

LabView is a trademark of National Instruments Corporation.

---

## Getting Started with LabWindows

The 32-bit Agilent 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 Agilent 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 Agilent 816x Instrument Driver

The Agilent 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.

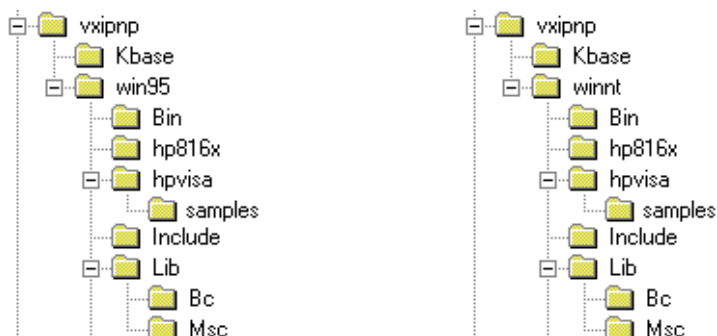
The following features are available:

- The Agilent 816x VXI*plug&play* Instrument Driver conforms with the VXI*plug&play* standard.
- There is one exception as the Agilent 816x driver does not have a soft front panel or a knowledge-based file.
- The Agilent 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 Agilent 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 Agilent VEE, LabWindows, and LabView.
- The Agilent 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 Agilent 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 Agilent 816x instrument driver creates the VXIPNP directory if it does not already exist. The structures for the Windows NT and Windows 95 vxipnp subdirectory tree are shown in Figure 9.



**Figure 9** Windows 95 and Windows NT VXIPNP Directory Structure

In the directory example, hp816x is a directory containing the instrument driver. There would be a directory for each instrument driver.

## Opening an Instrument Session

To control an instrument from a program, you must open a communication path between the computer/controller and the instrument. This path is known as an instrument session, and is opened with the function

```
ViStatus hp816x_init( ViRsrc InstrDesc, ViBoolean id_query, ViBoolean reset, ViPSession instrumentHandle );
```

Instruments are assigned a handle when the instrument session is opened. The handle, which is a pointer to the instrument, is the first parameter passed in all subsequent calls to driver functions.

The parameters of the function `hp816x_init` include:

- `ViRsrc InstrDesc`: the address of the instrument
- `ViBoolean id_query`: a Boolean flag which indicates if in-system verification should be performed.  
Passing `VI_TRUE (1)` will perform an in-system verification; passing `VI_FALSE (0)` will not.  
If you set `id_query` to false, you can use the generic functions of the instrument driver with other instruments.
- `ViBoolean reset`: a Boolean flag which indicates if the instrument should be reset when it is opened.  
Passing `VI_TRUE (1)` will perform a reset when the session is opened; passing `VI_FALSE (0)` will not perform a reset,
- `ViPSession instrumentHandle`: a pointer to an instrument session. `InstrumentHandle` is the handle which addresses the instrument, and is the first parameter passed in all driver functions.
- Successful completion of this function returns `VI_SUCCESS`

## Closing an Instrument Session

Sessions (instrumentHandle) opened with the hp816x\_init() function are closed with the function:

```
hp816x_close( ViSession instrumentHandle);
```

When no further communication with an instrument is required, the session must be explicitly closed (hp816x\_close() function).

VISA does not remove sessions unless they are explicitly closed. Closing the instrument session frees all data structures and system resources allocated to that session.

## VISA Data Types and Selected Constant Definitions

The driver functions use VISA data types. VISA data types are identified by the Vi prefix in the data type name (for example, ViInt16, ViUInt16, ViChar).

The file visatype.h contains a complete listing of the VISA data types, function call casts and some of the common constants.

### NOTE

You can find a partial list of the type definitions and constant definitions for the visatype.h in the Agilent 816x VXI*plug&play* Instrument Driver Online Help.

---

## Error Handling

Events and errors within a instrument control program can be detected by polling (querying) the instrument. Polling is used in application development environments (ADEs) that do not support asynchronous activities where callbacks can be used.

Programs can set up and use polling as shown below.

- 1 Declare a variable to contain the function completion code.

```
ViStatus errStatus;
```

Every driver function returns the completion code ViStatus.

If the function executes with no I/O errors, driver errors, or instrument errors, ViStatus is 0 (VI\_SUCCESS).

If an error occurs, ViStatus is a negative error code.

Warnings are positive error codes, and indicate the operation succeeded but special conditions exist.

- 2 Enable automatic instrument error checking following each function call.

```
hp816x_errorQueryDetect  
(instrumentHandle, VI_TRUE);
```

When enabled, the driver queries the instrument for an error condition before returning from the function.

If an error occurred, errStatus (Step 1) will contain a code indicating that an error was detected (hp816x\_INSTR\_ERROR\_DETECTED).

- 3 Check for an error (or event) after each function.

```
errStatus = hp816x_cmd(instrumentHandle, "SENS1:POW:RANG");  
check(instrumentHandle, errStatus);
```

After the function executes, errStatus contains the completion code.

The completion code and instrument ID are passed to an error checking routine. In the above statement, the routine is called 'check'.

- 4 Create a routine to respond to the error or event. This example queries whether an error has occurred, checks if the error is an instrument error and then checks if the error is a driver error.

```
void check (ViSession instrumentHandle, ViStatus errStatus)  
{
```

## 6 The Agilent 816x VXIplug&play Instrument Driver

```
/* variables for error code and message */
ViInt32 inst_err;
ViChar err_message[256];

/* VI_SUCCESS is 0 and is defined in VISATYPE.h */
if(VI_SUCCESS > errStatus)
{
/* hp816x_INSTR_ERROR_DETECTED defined in hp816x.h */
if(hp816x_INSTR_ERROR_DETECTED == errStatus)

{
/* query the instrument for the error */
hp816x_error_query(instrumentHandle, &inst_err, err_message);

/* display the error */
printf("Instrument Error : %ld, %s\n", inst_err, err_message);
}
else /* driver error */
{
/* get the driver error message */
hp816x_error_message(instrumentHandle, errStatus, err_message);

/* display the error */
printf("Driver Error : %ld, %s\n", errStatus, err_message);

}
/* optionally reset the instrument, close the instrument handle */
hp816x_reset(instrumentHandle);
hp816x_close(instrumentHandle);
exit(1);
}
return;
```



## Introduction to Programming

### Example Programs

See the Online Help and “VISA Programming Examples” on page 85.

### 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 Agilent *VXIplug&play* instrument drivers, you should enter the instrument addresses using only upper case letters. This is to ensure maximum portability.

For example, use GPIB0::22 rather than gpib0::22.

#### Callbacks

Callbacks are not supported by this driver.

### Development Environments

These sections contains suggestions as to how you can use hp816x\_32.dll within various application development environments.

#### Microsoft Visual C++ 4.0 (or higher) and Borland C++ 4.5 (or higher)

Please refer to your Microsoft Visual C++ or Borland C++ manuals for information on linking and calling DLLs.

#### Microsoft Visual Basic 4.0 (or higher)

Please refer to your Microsoft Visual Basic manual for information on calling DLLs.

The BASIC include file is hp816x.bas. You can find this file in the directory ~vxipnp\win95\include, where ~ is the directory in the VXIPNP variable.

By default, ~ is equivalent to C:\. This means that the file is in C:\vxipnp\win95\include.

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

### **Agilent VEE 5.01 (or higher)**

Your copy of Agilent VEE for Windows contains a document titled *Using VXIplug&play drivers with Agilent VEE for Windows*. This document contains the detailed information you need for Agilent VEE applications.

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

The Agilent 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 Agilent 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 `vxipnp\win95\hp816x`.

Do not include `hp816x.C` in your project.

You must provide an include file for `hp816x.H`. You do this by ensuring that the directory `~vxipnp\win95\include` is added to the include paths (CVI Project Option menu).

`~` is the directory in the VXIPNP variable. By default, `~` is equivalent to `C:\`. This means that the file is in `C:\vxipnp\win95\include`.

## Online Information

The latest copy of this driver can be downloaded via:

<http://www.agilent.com/comms/comp-test>

If you do not have web access, use the version of hp816x.exe on your OCT Support CD, or contact your Agilent 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 a Multi Frame Lambda Scan - a Lambda Logging operation where an Agilent 816xA/B Lightwave Measurement System with a Tunable Laser module performs a wavelength sweep and the Tunable Laser module is coordinated with the Agilent N7744A / N7745A Multiport Power Meter. The instruments must be connected by GPIB, LAN or USB. The Output Trigger Connector of the Agilent 816xA/B Lightwave Measurement System mainframe must be connected to the Input Trigger Connector of the Multiport Power Meter.

The following two functions apply to 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 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.

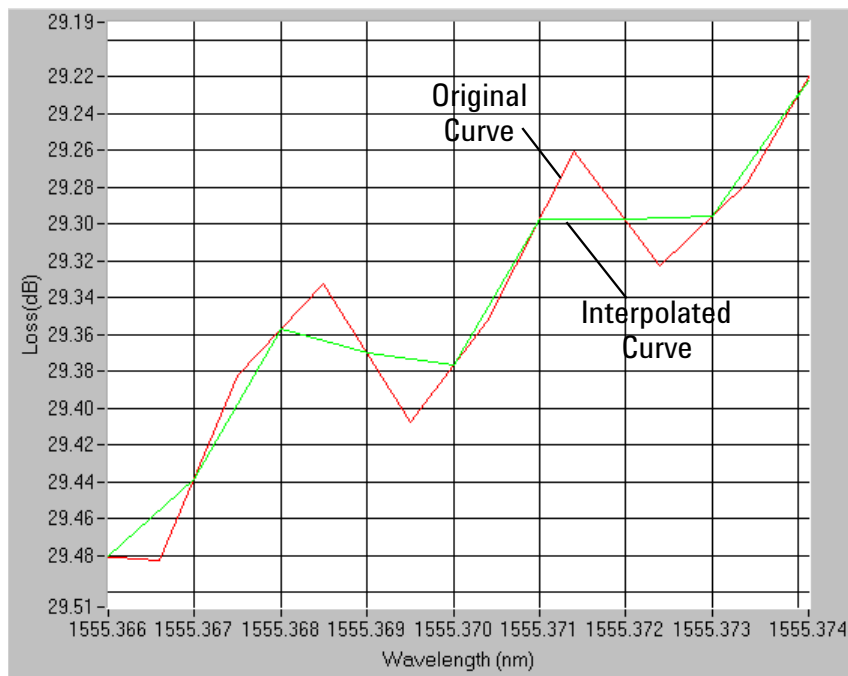
### Equally Spaced Datapoints

A linear interpolation is 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.

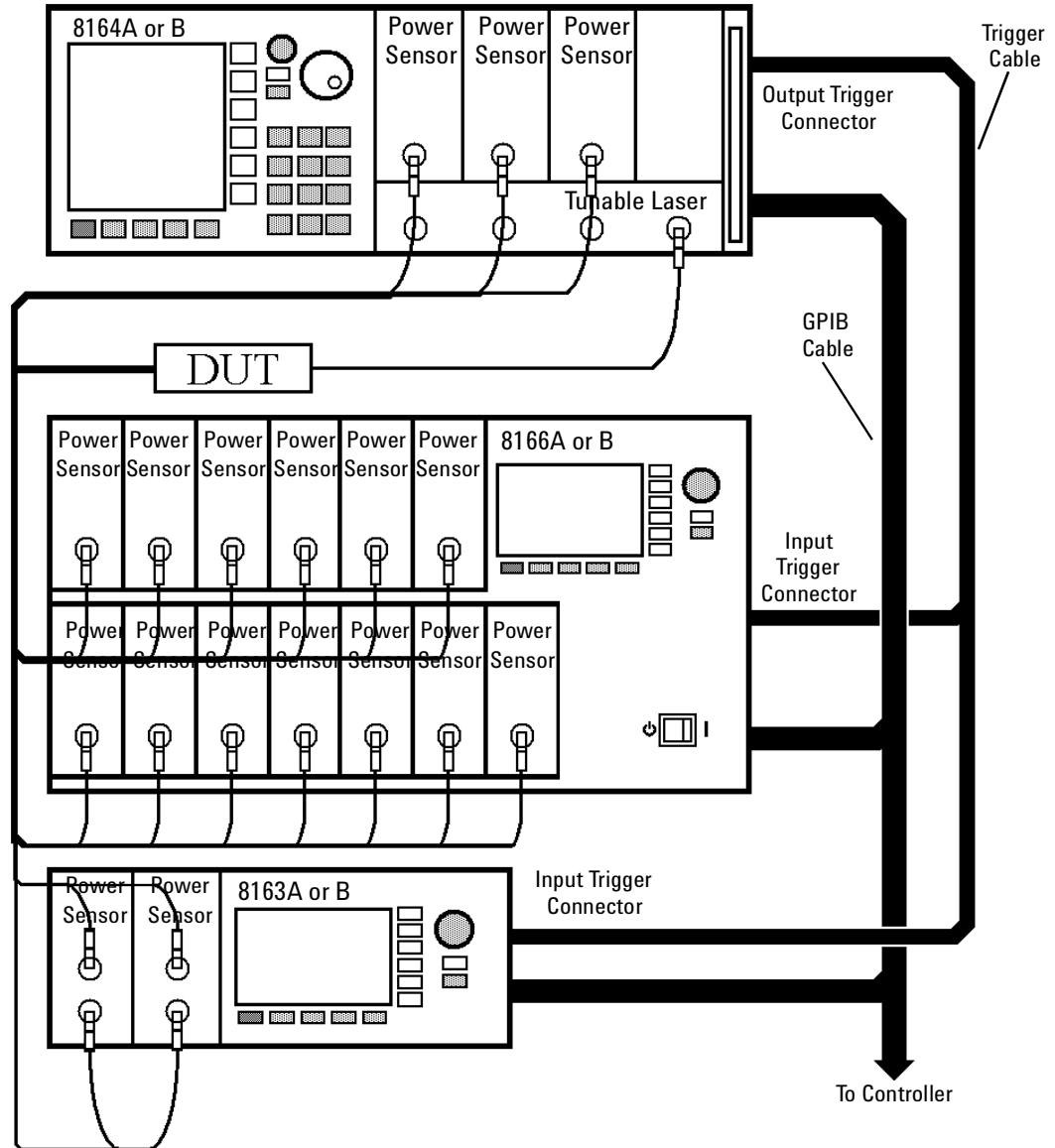
The disadvantage of using equally spaced datapoints is that the linear interpolation is analogous to the use of a low pass filter. Figure 10 shows the original curve as measured directly by a Power Meter and the interpolated curve.

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



**Figure 10** Equally Spaced Datapoints

## How to Perform a Multi-Frame Lambda Scan Application



**Figure 11** 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 116 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 Agilent 8164A/B Lightwave Measurement System mainframe that the Tunable Laser module is installed in.

**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 5.0 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 Agilent 8164A/B Lightwave Measurement System with a back-loadable Tunable Laser module and Power Meter Channels located in the

Multiport Power Meter. The function performs a wavelength sweep where the Tunable Laser module and Power Sensors are co-ordinated with each other.

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 back-loadable Tunable Laser Sources. The function scans each mainframe in the order that they were originally registered by the Register Mainframe function (`hp816x_registerMainframe`). The first back-loadable Tunable Laser Source found will perform the sweep operation.

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. All mainframes must be connected to the GPIB bus and have their Input Trigger Connector connected to the Output Trigger Connector of the Agilent 8164A/B Lightwave Measurement System 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 too 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.

That is, it executes an operation where a Agilent 8164A/B Lightwave Measurement System with a back-loadable Tunable Laser module and a Multiport Power Meter, performs a wavelength sweep where the Tunable Laser module and Power Sensors are coordinated with each other.

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

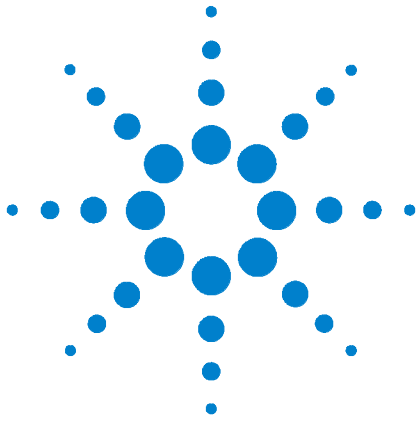
The Get Number of PWM Channels (hp816x\_getNoOfRegPWMChannnels\_Q) function returns the number of Power Meter channels in a test setup.

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.



## 7 Error Codes

This chapter gives information about error codes used with the Agilent N7744A / N7745A Multiport Power Meter.

GPIO Error Strings 124



## GPIB Error Strings

Error strings in the range -100 to -183 are defined by the SCPI standard, downloadable from:

<http://www.scpiconsortium.org/scpistandard.htm>

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

**Table 1** 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: <a href="http://www.scpiconsortium.org/scpistandard.htm">http://www.scpiconsortium.org/scpistandard.htm</a> String descriptions taken from this standard (VERSION 1999.0 May, 1999), whether in whole or in part, are enclosed by [ ] in this table.	
<b>-100 to -199 Command Errors</b>		
Standard	-100	"Command Error" [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 <i>IEEE 488.2, 11.5.1.1.4</i> has occurred.]
Standard	-101	"Invalid character" [A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of error -114 and perhaps some others.]
Standard	-102	"Syntax error" [An unrecognized command or data type was encountered; for example, a string was received when the <b>device</b> does not accept strings.]
Standard	-103	"Invalid separator" [The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit]
Standard	-104	"Data type error" [The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.]
Standard	-105	"GET not allowed" [A Group Execute Trigger was received within a program message (see <i>IEEE488.2, 7.7</i> ).]
Standard	-108	"Parameter not allowed" [More parameters were received than expected for the header]
Standard	-109	"Missing parameter" [Fewer parameters were received than required for the header]
Standard	-112	"Program mnemonic too long" [The header contains more than twelve characters (see <i>IEEE 488.2, 7.6.1.4.1</i> ).]

Table 1 Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-113	"Undefined header" [The header is syntactically correct, but it is undefined for this specific <b>device</b> ; for example, *XYZ is not defined for any device.]
Standard	-120	"Numeric data error" [This error, as well as errors -121 through -129, are generated when parsing a data element which appears to be numeric, including the nondecimal numeric types. This error message is used if the <b>device</b> cannot detect a more specific error.]
Standard	-121	"Invalid character in number" [An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric]
Standard	-123	"Exponent too large" [The magnitude of the exponent was larger than 32000 (see <i>IEEE 488.2, 7.7.2.4.1</i> ).]
Standard	-124	"Too many digits" [The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see <i>IEEE 488.2, 7.7.2.4.1</i> ).]
Standard	-128	"Numeric data not allowed" [A legal numeric data element was received, but the <b>device</b> does not accept one in this position for the header.]
Standard	-131	"Invalid suffix" [The suffix does not follow the syntax described in <i>IEEE 488.2, 7.7.3.2</i> , or the suffix is inappropriate for this <b>device</b> .]
Standard	-134	"Suffix too long" [The suffix contained more than 12 characters (see <i>IEEE 488.2, 7.7.3.4</i> ).]
Standard	-138	"Suffix not allowed" [A suffix was encountered after a numeric element which does not allow suffixes.]
Standard	-141	"Invalid character data" [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" [A legal character data element was encountered where prohibited by the <b>device</b> .]
Standard	-150	"String data error" [This error, as well as errors -151 through -159, are generated when parsing a string data element. This error message is used when the <b>device</b> cannot detect a more specific error.]
Standard	-151	"Invalid string data" [A string data element was expected, but was invalid for some reason (see <i>IEEE 488.2, 7.7.5.2</i> ); for example, an END message was received before the terminal quote character.]
Standard	-158	"String data not allowed" [A string data element was encountered but was not allowed by the <b>device</b> at this point in parsing.]

Table 1 Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-161	<p>"Invalid block data"</p> <p>[A block data element was expected, but was invalid for some reason (see <i>IEEE 488.2</i>, 7.7.6.2); for example, an END message was received before the length was satisfied.]</p>
Standard	-168	<p>"Block data not allowed"</p> <p>[A legal block data element was encountered but was not allowed by the <b>device</b> at this point in parsing.]</p>
Standard	-170	<p>"Expression error"</p> <p>[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 <b>device</b> cannot detect a more specific error.]</p>
Standard	-171	<p>"Invalid expression"</p> <p>[The expression data element was invalid (see <i>IEEE 488.2</i>, 7.7.7.2); for example, unmatched parentheses or an illegal character.]</p>
Standard	-178	<p>"Expression data not allowed"</p> <p>[A legal expression data was encountered but was not allowed by the <b>device</b> at this point in parsing.]</p>
Standard	-181	<p>"Invalid outside macro definition"</p> <p>[Indicates that a macro parameter placeholder (\$&lt;number&gt;) was encountered outside of a macro definition.]</p>
Standard	-183	<p>"Invalid inside macro definition"</p> <p>[Indicates that the program message unit sequence, sent with a *DDT or *DMC command, is syntactically invalid (see <i>IEEE 488.2</i>, 10.7.6.3).]</p>
New	-185	<p>"Subop out of range"</p> <p><i>Description:</i> 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.</p> <p><i>Example:</i> 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).</p> <p><i>Note:</i> Incorrect slots and channels addresses are handled by error code -301</p>
<b>-200 to -299 Execution Errors</b>		

Table 1 Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-200	<p>"Execution error (StatExecError)"</p> <p><i>Description:</i> 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 for a number of reasons.</p> <p><i>Example:</i> 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.</p>
New	-201	<p>"Please be patient - GPIB currently locked out"</p> <p><i>Description:</i> Some operations block the complete system. Since no sensible measurements are possible while this is true, the GPIB is locked out.</p> <p><i>Example:</i> When ARA, Lambda zeroing or zeroing is executing on a TLS module, the GPIB is not accessible.</p>
New	-205	<p>"Powermeter not running (StatMeterNotRunning)"</p> <p><i>Description:</i> Some command and actions may stop the data acquisition 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.</p>
Old	-211	<p>"Trigger ignored"</p> <p><i>Description:</i> A trigger has been detected but ignored because of timing constraints. (For Example: average time to large).</p>
Old	-212	<p>"Arm ignored"</p> <p><i>Description:</i> The user can set the automatic re-arming option for input and output trigger events (see <a href="#">"Triggering - The TRIGGER Subsystem"</a> on page 82). When this error occurs, the device ignores the setting because the current module status does not allow the change of trigger settings.</p>
Old	-213	<p>"Init ignored"</p> <p><i>Description:</i> The INIT:IMM command (<a href="#">page 64</a>) 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.</p>
Old	-220	<p>"Parameter error (StatParmError)"</p> <p><i>Description:</i> The user has passed a parameter that cannot be changed in this way. The device cannot detect one of the following more specific errors:</p>
Old	-220	<p>-220, "Parameter error (StatParmOutOfRange)"</p> <p><i>Description:</i> The user has passed a parameter that exceeds the valid range for this parameter.</p>

**Table 1** Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Old	-220	"Parameter error (StatParmIllegalVal)" <i>Description:</i> The user has passed a parameter that does not match a value in a list of possible values.
Old	-221	"Settings conflict (StatParmInconsistent)" <i>Description:</i> The user has passed a parameter that conflicts with other already configured parameters. <i>Example:</i> There are constraints for TLS sweep parameters: this error is generated when lambda step size exceeds the difference between start and stop wavelength. If error -221 is returned after you try to start a wavelength sweep, one of the following cases of sweep parameter inconsistency has occurred: Continuous Sweep mode AND I Start is less than I Stop. Continuous Sweep mode AND Sweep Time is too short. Adjust Sweep Speed, I Start, or I Stop. Continuous Sweep mode AND Sweep Time is too long. Adjust Sweep Speed, I Start, or I Stop. Continuous Sweep mode AND Trigger Frequency is too high. Adjust Step Size. Trigger Frequency is the Sweep Speed divided by the Step Size. Stepped Sweep mode AND Lambda Logging Enabled. Continuous Sweep mode AND Lambda Logging Enabled AND Output trigger mode not set to STFinished (Step finished). Continuous Sweep mode AND Lambda Logging is Enabled AND Modulation Source is not set to OFF. Continuous Sweep mode AND Lambda Logging is Enabled AND Sweep Cycles is not set to 1. Continuous Sweep mode AND Coherence Control is Enabled.
Standard	-222	"Data out of range (StatParmTooLarge)" <i>Description:</i> The user has passed a continuous parameter that is too large. <i>Example:</i> Wavelength 1800nm when maximum wavelength is 1700nm.
Standard	-222	"Data out of range (StatParmTooSmall)" <i>Description:</i> The user has passed a continuous parameter that is too small. <i>Example:</i> Wavelength 700nm when minimum wavelength is 800nm.



Table 1 Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-223	<p>"Too much data"</p> <p><i>Description:</i> A function returns more data or the user requests more data than the application is able to handle.</p> <p><i>Example:</i> A tunable laser source produces more data when lambda values of a sweep are stored than the N7744A instrument is able to handle. Use the new SENSE:FUNC:RES:BLOCK? command to split the data acquisition into multiple parts.</p>
Standard	-224	<p>"Illegal parameter value"</p> <p>[Used where exact value, from a list of possibles, was expected.]</p>
New	-225	<p>"Out of memory"</p> <p><i>Description:</i> The request application or function cannot be executed because the instrument runs out of memory.</p>
Old	-231	<p>"Data questionable (StatValNYetAcc)"</p> <p><i>Description:</i> The data that is returned is not accurate or reliable. The user should repeat the operation. The reason for this error is unspecified.</p> <p><i>Example:</i> A powermeter configured a long average time has not completed its current measurement cycle when the user queries the current power.</p>
Old	-231	<p>"Data questionable (StatRangeTooLow)"</p> <p><i>Description:</i> 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.</p>
Old	-261	<p>"Math error in expression (StatUnitCalculationError)"</p> <p><i>Description:</i> This may occur when the user attempts to transform data in a way that is currently not possible.</p> <p><i>Example:</i> 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.</p>
Standard	-272	<p>"Macro execution error"</p> <p>[Indicates that a syntactically legal macro program data sequence could not be executed due to some error in the macro definition (see <i>IEEE 488.2</i>, 10.7.6.3.)]</p>

**Table 1** Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-273	<p>"Illegal macro label"</p> <p>[Indicates that the macro label defined in the *DMC command was a legal string syntax, but could not be accepted by the <b>device</b> (see <i>IEEE 488.2</i>, 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.]</p>
Standard	-276	<p>"Macro recursion error"</p> <p>[Indicates that a syntactically legal macro program data sequence could not be executed because the device found it to be recursive (see <i>IEEE 488.2</i>, 10.7.6.6).]</p>
Standard	-277	<p>"Macro redefinition not allowed"</p> <p>[Indicates that a syntactically legal macro label in the *DMC command could not be executed because the macro label was already defined (see <i>IEEE 488.2</i>,10.7.6.4).]</p>
Standard	-278	<p>"Macro header not found"</p> <p>[Indicates that a syntactically legal macro label in the *GMC? query could not be executed because the header was not previously defined.]</p>
Old	-284	<p>"Function currently running (StatModuleBusy)"</p> <p><i>Description:</i> This error is generated when a function is currently running on a module so that it cannot process another commands.</p> <p><i>Example:</i> When a powermeter is running a logging application, you are not able to configure the logging application parameters (also see -200).</p>
Old	-286	<p>"No function currently running"</p> <p><i>Description:</i> This error is generated when a user tries to execute a command which requires a particular set of data that is not available.</p> <p><i>Example:</i> 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).</p>
New	-290	<p>"Application currently running - no GPIB support"</p> <p><i>Description:</i> The instrument has built-in applications that have no GPIB support ( such as Logging,Stability,PACT).</p> <p><i>Example</i> When an application is running error -290 will be returned if any command other than one the following is sent: *WAI *OPC? :SPECial:REBoot :SYSTem:ERRor? :SYSTem:VERSion?</p>

**-300 to -399 or between 1 and 32767 Device-Specific Errors (Module)**

Table 1 Overview for Supported Strings

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

**Table 1** Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-321	"Out of memory" [An internal operation needed more memory than was available.]
New	-322	"Flash programming error (StatFlashEraseFailed)" "Flash programming error (StatFlashWriteFailed)" "Flash programming error (StatFlashDataCntError)" "Flash programming error (StatFlashDPAlgoFailed)" <i>Description:</i> An error has occurred 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)" It is not possible to activate the offset (I) functionality when the offset table is empty "Flash programming error (StatUserCalTable Full)" The offset (I) table is full and no more ? can be stored "Flash programming error (StatUserCalActive)" It is not possible to program the offset (I) table when the offset (I) feature is activated. Deactivate first.
Old	-330	"Self-test failed" <i>Description:</i> You have started the self test, but the module has detected an error while executing it
New	-340	"Printing error (StatPrintError)" <i>Description:</i> An unspecified problem occurred while communicating with the printer.
New	-341	"Printing error - paper out (StatPaperOut)" <i>Description:</i> The instrument cannot print because there is no paper in the connected printer.
New	-342	"Printing error - offline (StatOffline)" <i>Description:</i> The instrument cannot print because the connected printer is offline.
Standard	-350	"Queue overflow" [A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.]
<b>-400 to -499 Query Errors</b>		
Standard	-400	"Query error" [This is the generic query error for <b>devices</b> that cannot detect more specific errors. This code indicates only that a Query Error as defined in <i>IEEE 488.2</i> , 11.5.1.1.7 and 6.3 has occurred.]
Standard	-410	"Query INTERRUPTED" [Indicates that a condition causing an INTERRUPTED Query error occurred (see <i>IEEE 488.2</i> , 6.3.2.3); for example, a query followed by DAB or GET before a response was completely sent.]

**Table 1** Overview for Supported Strings

New/Old/Standard	Error	
	Number	String
Standard	-420	“Query UNTERMINATED” [Indicates that a condition causing an UNTERMINATED Query error occurred (see <i>IEEE 488.2</i> , 6.3.2.2); for example, the <b>device</b> was addressed to talk and an incomplete program message was received.]
Standard	-430	“Query DEADLOCKED” [Indicates that a condition causing an DEADLOCKED Query error occurred (see <i>IEEE 488.2</i> , 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” [Indicates that a query was received in the same program message after an query requesting an indefinite response was executed (see <i>IEEE 488.2</i> , 6.5.7.5).]

**Table 2** Overview for Unsupported Strings

New/Old/Standard	Error	
	Number	String
Old	all positive errors	
Old	-110	“Command header error”
Old	-111	“Header separator error”
Old	-114	“Header suffix out of range”
Old	-130	“Suffix error”
Old	-140	“Character data error”
Old	-144	“Character data too long”
Old	-160	“Block data error”
Old	-201	“Invalid while in local”
Old	-202	“Settings lost due to ???”
Old	-210	“Trigger error”
Old	-214	“Trigger deadlock”
Old	-215	“Arm deadlock”
Old	-230	“Data corrupt or stale”
Old	-240	“Hardware error”
Old	-241	“Hardware missing”
Old	-260	“Expression error”
Old	-280	“Program error”
Old	-281	“Cannot create program”
Old	-282	“Illegal program name”
Old	-283	“Illegal variable name”
Old	-285	“Program syntax error”

## 7 Error Codes

**Table 2** Overview for Unsupported Strings

New/Old/Standard	Error	
	Number	String
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

## A

Agilent VEE, 100  
AutoIP, 50

## B

Binary block, 16

## C

Channel Numbers, 16  
Command summary, 28  
Common commands, 18  
Continuous measurement, 64, 65

## D

Data Types, 16  
default gateway, 52, 54  
DHCP, 50  
DNS, 50  
domain name, 51, 53

## E

Error handling, 111  
Error strings  
  GPIB, 124  
Ethernet parameters, 52  
Event register  
  operation enable, 41, 43  
  questionable enable, 45, 46  
Event Status Enable, 35  
Event Status Register, 35

## F

FETCh subsystem, 63

## G

GPIB Interface, 8

## H

host name, 51, 52

## I

Identification, 35  
IEEE-Common Commands, 34  
INITiate subsystem, 64  
Input queue, 12  
Installed options, 37  
Instrument addresses, 113  
Instrument Behaviour Settings, 47  
Instrument driver, 106  
Instrument driver installation, 96  
Interface  
  behaviour settings, 47  
IP address, 51, 53

## L

LabView, 102  
LabWindows, 105  
Lambda scan  
  get result function, 121  
  mult-frame, 119  
Lambda scans, 116

## M

MAC address, 50  
Measurement  
  start, 64  
Measurement Functions, 63  
Message queues, 12

## O

Operation Complete, 36  
Operation enable, 41, 43  
Options, 37  
Output queue, 12

## P

Power measurement  
  example of FETCh and READ usage, 88  
Power Meter  
  continuous measurement, 64  
  start measurement, 64  
Power meter  
  configure all, 65  
  continuous measurement, 65  
  current value, 66  
  read all, 65

## Q

Questionable enable, 45, 46

## R

READ subsystem, 66  
Register  
  Operational Slot Status, 24  
  Questionable slot status, 24  
  Standard Event Status, 23  
  Status byte, 23  
  Status summary, 23  
register mainframe, 119  
Reset, 37  
Root layer commands, 60

## S

SCPI revision, 49  
Self-test, 38  
SENSe subsystem, 66  
settings, 56  
Slot Numbers, 16  
SLOT subsystem, 60  
Specific Command Summary, 28  
Start  
  measurement, 64  
  power meter measurement, 64  
Status Byte, 37  
Status Command Summary, 25  
Status Information, 18  
Status Reporting, 40  
STATus subsystem, 40  
subnet mask, 51, 53  
Subsystem  
  FETCh, 63  
  INITiate, 64  
  READ, 66  
  SENSe, 66  
  SLOT, 60  
  STATus, 40  
  SYSTem, 47  
  TRIGger, 82  
SYSTem subsystem, 47

## T

Test, 38  
Trace Data Access, 49  
TRIGger Subsystem, 82

## Index

### U

Units, [15](#)  
unregister mainframe, [119](#)

### V

Visa calls  
    How to use, [86](#)  
VISA data types, [110](#)  
Visual programming environment, [100](#)  
Vxipnp directory, [107](#)

### W

Wait, [39](#)





© Agilent Technologies Deutschland  
GmbH 2009

Printed in Germany  
Second edition, February 2009

N7744-90C01