tek.com/keithley

Model 2470
High Voltage SourceMeter® Instrument

Reference Manual

2470-901-01 Rev. B / September 2019

2470-901-01B

KEITHLEY
A Tektronix Company

Model 2470

High Voltage SourceMeter Instrument

Reference Manual

© 2019, Keithley Instruments, LLC

Cleveland, Ohio, U.S.A.

All rights reserved.

Any unauthorized reproduction, photocopy, or use of the information herein, in whole or in part, without the prior written approval of Keithley Instruments, LLC, is strictly prohibited.

These are the original instructions in English.

TSP®, TSP-Link®, and TSP-Net® are trademarks of Keithley Instruments, LLC. All Keithley Instruments product names are trademarks or registered trademarks of Keithley Instruments, LLC. Other brand names are trademarks or registered trademarks of their respective holders.

The Lua 5.0 software and associated documentation files are copyright © 1994 - 2015, Lua.org, PUC-Rio. You can access terms of license for the Lua software and associated documentation at the Lua licensing site (http://www.lua.org/license.html).

Microsoft, Visual C++, Excel, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Document number: 2470-901-01 Rev. B / September 2019

KEITHLEY
A Tektronix Company

Safety precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with nonhazardous voltages, there are situations where hazardous conditions may be present.

This product is intended for use by personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product. Refer to the user documentation for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product warranty may be impaired.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating properly, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the user documentation. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, perform safe installations, and repair products. Only properly trained service personnel may perform installation and service procedures.

Keithley products are designed for use with electrical signals that are measurement, control, and data I/O connections, with low transient overvoltages, and must not be directly connected to mains voltage or to voltage sources with high transient overvoltages. Measurement Category II (as referenced in IEC 60664) connections require protection for high transient overvoltages often associated with local AC mains connections. Certain Keithley measuring instruments may be connected to mains. These instruments will be marked as category II or higher.

Unless explicitly allowed in the specifications, operating manual, and instrument labels, do not connect any instrument to mains.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30 V RMS, 42.4 V peak, or 60 VDC are present. A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 V, no conductive part of the circuit may be exposed.

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance-limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, ensure that the line cord is connected to a properly-grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided in close proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

For safety, instruments and accessories must be used in accordance with the operating instructions. If the instruments or accessories are used in a manner not specified in the operating instructions, the protection provided by the equipment may be impaired.

Do not exceed the maximum signal levels of the instruments and accessories. Maximum signal levels are defined in the specifications and operating information and shown on the instrument panels, test fixture panels, and switching cards.

When fuses are used in a product, replace with the same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as protective earth (safety ground) connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

If a  screw is present, connect it to protective earth (safety ground) using the wire recommended in the user documentation.

The  symbol on an instrument means caution, risk of hazard. The user must refer to the operating instructions located in the user documentation in all cases where the symbol is marked on the instrument.

The  symbol on an instrument means warning, risk of electric shock. Use standard safety precautions to avoid personal contact with these voltages.

The  symbol on an instrument shows that the surface may be hot. Avoid personal contact to prevent burns.

The  symbol indicates a connection terminal to the equipment frame.

If this  symbol is on a product, it indicates that mercury is present in the display lamp. Please note that the lamp must be properly disposed of according to federal, state, and local laws.

The **WARNING** heading in the user documentation explains hazards that might result in personal injury or death. Always read the associated information very carefully before performing the indicated procedure.

The **CAUTION** heading in the user documentation explains hazards that could damage the instrument. Such damage may invalidate the warranty.

The **CAUTION** heading with the  symbol in the user documentation explains hazards that could result in moderate or minor injury or damage the instrument. Always read the associated information very carefully before performing the indicated procedure. Damage to the instrument may invalidate the warranty.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits — including the power transformer, test leads, and input jacks — must be purchased from Keithley. Standard fuses with applicable national safety approvals may be used if the rating and type are the same. The detachable mains power cord provided with the instrument may only be replaced with a similarly rated power cord. Other components that are not safety-related may be purchased from other suppliers as long as they are equivalent to the original component (note that selected parts should be purchased only through Keithley to maintain accuracy and functionality of the product). If you are unsure about the applicability of a replacement component, call a Keithley office for information.

Unless otherwise noted in product-specific literature, Keithley instruments are designed to operate indoors only, in the following environment: Altitude at or below 2,000 m (6,562 ft); temperature 0 °C to 50 °C (32 °F to 122 °F); and pollution degree 1 or 2.

To clean an instrument, use a cloth dampened with deionized water or mild, water-based cleaner. Clean the exterior of the instrument only. Do not apply cleaner directly to the instrument or allow liquids to enter or spill on the instrument. Products that consist of a circuit board with no case or chassis (e.g., a data acquisition board for installation into a computer) should never require cleaning if handled according to instructions. If the board becomes contaminated and operation is affected, the board should be returned to the factory for proper cleaning/servicing.

Safety precaution revision as of June 2017.

Table of contents

Introduction 1-1

Welcome 1-1

Extended warranty 1-2

Contact information 1-2

Organization of manual sections 1-3

Features 1-4

General ratings 1-4

Installation 2-1

Dimensions 2-1

Handle and bumpers 2-4

Removing the handle and bumpers 2-4

Instrument power 2-6

Connect the power cord 2-7

Power the instrument on or off 2-7

Remote communications interfaces 2-7

Supported remote interfaces 2-8

Comparison of the communications interfaces 2-8

GPIB setup 2-9

LAN communications 2-14

USB communications 2-22

2470 web interface 2-27

How to install the Keithley I/O Layer 2-33

Modifying, repairing, or removing Keithley I/O Layer software 2-34

Interface access 2-34

Changing the interface access type 2-35

Changing the password 2-35

Switching control interfaces 2-36

Determining the command set you will use 2-36

System information 2-37

Instrument description 3-1

Front-panel overview 3-1

Rear-panel overview 3-3

Turn the 2470 output on or off 3-4

Touchscreen display 3-5

Select items on the touchscreen 3-5

Scroll bars 3-6

Enter information 3-6

Adjust the backlight brightness and dimmer 3-7

Event messages 3-8

Screen descriptions 3-9

Home screen 3-9

Menu overview 3-18
QuickSet menu 3-18
Source menu 3-19
Measure menu 3-21
Views menu 3-25
Trigger menu 3-30
Scripts menu 3-33
System menu 3-36

APPS Manager 3-39
Download and run TSP applications 3-40

Display features 3-40
Setting the number of displayed digits 3-40
Setting the display format 3-41
Customizing a message for the USER swipe screen 3-42
Creating messages for interactive prompts 3-43
Save screen captures to a USB flash drive 3-44

Instrument sounds 3-44

Saving setups 3-45
Save a user setup to internal memory 3-45
Save a user setup to a USB flash drive 3-46
Copy a user setup 3-46
Delete a user setup 3-47
Recall a user setup 3-47
Define the setup used when power is turned on 3-48

Resets 3-49
Reset the instrument 3-50

Using the event log 3-50
Information provided for each event log entry 3-50
Event log settings 3-51
Effects of errors on scripts 3-52

Saving front-panel settings into a macro script 3-52
Recording a macro script 3-53
Running a macro script 3-53
Front-panel macro recording limitations 3-54

Sourcing and measuring 4-1

Test connections 4-1
Basic connections 4-2
Using the interlock 4-3
Front- or rear-panel test connections 4-6
Two-wire compared to four-wire measurements 4-7
Test fixtures 4-15
Output-off state 4-16

Source-measure overview 4-20
Source and measure order 4-20
Source and measure through the front panel 4-21
Source and measure using SCPI commands 4-33
Source and measure using TSP commands 4-34

Protection 4-35
Overvoltage protection 4-35
Source limits 4-36

Ranges 4-39

Source range ... 4-39
Measurement range ... 4-41

Automatic reference measurements ... 4-44
Setting autozero ... 4-44

Source readback ... 4-45
Setting source readback ... 4-45

Source delay ... 4-46
Setting the source delay ... 4-47

Relative offset ... 4-47
Establishing a relative offset value ... 4-48
Disabling the relative offset ... 4-49

Calculations that you can apply to measurements ... 4-50
mx+b ... 4-50
Percent ... 4-51
Reciprocal (1/X) ... 4-51
Setting percent math operations ... 4-52
Setting mx+b math operations ... 4-52
Setting reciprocal math operations ... 4-53
Switching math on the SETTINGS swipe screen ... 4-53
Displayed measurements ... 4-54

Sweep operation ... 4-54
Linear staircase sweep ... 4-54
Logarithmic staircase sweep ... 4-55
Setting up a sweep ... 4-56
Aborting a sweep ... 4-62
Sweep programming examples ... 4-63
Increasing the speed of sweeps ... 4-65

Limit testing and binning ... 4-66
Limit testing using the front-panel interface ... 4-66
Set up a limit test using the remote interface ... 4-68

Configuration lists ... 4-82
Configuration indexes ... 4-82
Working with configuration lists and indexes ... 4-84
Recall a configuration index ... 4-91
View configuration list contents ... 4-92
Delete a configuration index or list ... 4-94
Overwrite an existing index ... 4-95
List the available configuration lists ... 4-96
Determine the number of indexes in a configuration list ... 4-96
Save a configuration list ... 4-97
Remote commands for configuration list operations ... 4-98

Source-measure considerations ... 5-1

Circuit configurations ... 5-1
Source current ... 5-1
Source voltage ... 5-2

Operating boundaries ... 5-4
Current source operating boundaries ... 5-5
Voltage limit boundary examples ... 5-5
Current limit boundary examples ... 5-7

Output transient recovery ... 5-8

Load regulation ... 5-8

Using NPLCs to adjust speed and accuracy ... 5-9
Noise shield ... 5-11
Safety shield ... 5-11
Safety shielding ... 5-12
Noise and chassis ground ... 5-12
Floating the 2470 ... 5-13
Guarding ... 5-15
Using guard with a test fixture ... 5-15
Guard circuit drawing ... 5-16
Sink operation ... 5-16
Battery charge and discharge ... 5-17
Measurement settling time considerations ... 5-18
Overtemperature protection ... 5-19
Current breakdown protection ... 5-19
Calculating accuracy ... 5-20
Calculating source or measure accuracy ... 5-21
Calculate accuracy of a resistance measurement made by sourcing I and measuring V ... 5-21
Offset-compensated ohm calculations ... 5-22
Power calculations ... 5-23
High-capacitance operation ... 5-23
Enabling the high capacitance feature ... 5-24
Filtering measurement data ... 5-24
Repeating average filter ... 5-25
Moving average filter ... 5-25
Setting up the averaging filter ... 5-26
Order of operations ... 5-27
Reset default values ... 5-28
Default values ... 5-28

Reading buffers ... 6-1

Introduction to reading buffers ... 6-1
Getting started with buffers ... 6-2
Types of reading buffers ... 6-2
Effects of reset and power cycle on buffers ... 6-2
Buffer fill status ... 6-2
Timestamps ... 6-4
Creating buffers ... 6-5
Setting reading buffer options ... 6-9
Setting reading buffer capacity ... 6-9
Setting the fill mode ... 6-12
Selecting a buffer ... 6-14
Viewing and saving buffer content ... 6-17
Using the front panel to store readings in the selected buffer ... 6-20
Options when saving buffer data to a USB flash drive ... 6-21

Clearing buffers ... 6-23
Deleting buffers ... 6-25
Remote buffer operation ... 6-25
Storing data in buffers ... 6-26
Accessing the data in buffers ... 6-28
Buffer read-only attributes ... 6-29
Reading buffer time and date values ... 6-29
Reading buffer for . . . do loops ... 6-30
Writable reading buffers ... 6-31
Apply mathematical expressions to reading buffer data ... 6-33
Mathematical expressions for buffer math ... 6-33
Set up buffer math using SCPI commands ... 6-34
Set up buffer math using TSP commands ... 6-34
Using buffers across TSP-Link nodes ... 6-34

Graphing ... 7-1
Introduction ... 7-1
About the graph screens ... 7-1
How to work with the graph ... 7-3
Use the Graph swipe bar ... 7-4
Change the data that is graphed ... 7-6
Add, remove, and clear traces ... 7-6
Active buffer ... 7-7
Change the display of data ... 7-7
Change the scale of the graph ... 7-8
Set up triggers ... 7-9
Types of triggers ... 7-9
Trigger settings ... 7-10
Graph measurement using triggers ... 7-11
About the Histogram screen ... 7-12
How to work with the Histogram ... 7-12
Change the data that is binned ... 7-13

Triggering ... 8-1
Measurement methods ... 8-1
Continuous measurement triggering ... 8-1
Trigger key triggering ... 8-2
Trigger model triggering ... 8-2
Switching between measurement methods ... 8-2
Triggering ... 8-3
Command interface triggering ... 8-3
Triggering using hardware lines ... 8-4
LAN triggering overview ... 8-4
Trigger timers ... 8-5
Event blenders ... 8-9
Interactive triggering ... 8-10

Digital I/O 8-12
Digital I/O connector and pinouts 8-13
Digital I/O port configuration 8-14
Digital I/O lines 8-16
Remote digital I/O commands 8-21
Digital I/O bit weighting 8-23
Digital I/O programming examples 8-23

Trigger model 8-25
TriggerFlow Trigger Model 8-26
Trigger-model blocks 8-26
Trigger-model templates 8-45
Assembling trigger-model blocks 8-47
Running the trigger model 8-49
Using trigger events to start actions in the trigger model 8-52

TSP-Link and TSP-Net 9-1

TSP-Link System Expansion Interface 9-1
TSP-Link connections 9-1
TSP-Link nodes 9-3
Master and subordinates 9-4
Initializing the TSP-Link system 9-4
Sending commands to TSP-Link nodes 9-5
Using the reset() command 9-5
Terminating scripts on the TSP-Link system 9-6
Triggering using TSP-Link trigger lines 9-6
Running simultaneous test scripts 9-7
Using 2470 TSP-Link commands with other TSP-Link products 9-12

TSP-Net 9-13
Using TSP-Net with any ethernet-enabled instrument 9-14
Remote instrument events 9-16
TSP-Net instrument commands: General device control 9-16
TSP-Net instrument commands: TSP-enabled device control 9-16
Example: Using tspnet commands 9-17

Maintenance 10-1

Introduction 10-1

Line fuse replacement 10-1

Lithium battery 10-2

Front-panel display 10-2
Cleaning the front-panel display 10-2
Abnormal display operation 10-3
Removing ghost images or contrast irregularities 10-3

Upgrading the firmware 10-3
From the front panel 10-4
Using SCPI 10-5
Using TSP 10-6
Using TSB 10-7

Introduction to SCPI commands 11-1

Introduction to SCPI 11-1
Command execution rules 11-1

Command messages 11-1

SCPI command programming notes 11-3
SCPI command formatting 11-3
Using the SCPI command reference 11-5

Acquiring readings using SCPI commands 11-8

SCPI command reference 12-1

:FETCh? 12-1
:MEASure? 12-3
:READ? 12-6
*RCL 12-8
*SAV 12-9

CALCulate subsystem 12-10
:CALCulate[1]:<function>:MATH:FORMat 12-10
:CALCulate[1]:<function>:MATH:MBFactor 12-12
:CALCulate[1]:<function>:MATH:MMFactor 12-13
:CALCulate[1]:<function>:MATH:PERCent 12-15
:CALCulate[1]:<function>:MATH:STATe 12-16
:CALCulate2:<function>:LIMit<Y>:AUDible 12-17
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO 12-18
:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate] 12-19
:CALCulate2:<function>:LIMit<Y>:FAIL? 12-20
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] 12-21
:CALCulate2:<function>:LIMit<Y>:STATe 12-22
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] 12-23

DIGital subsystem 12-24
:DIGital:LINE<n>:MODE 12-24
:DIGital:LINE<n>:STATe 12-26
:DIGital:READ? 12-27
:DIGital:WRITe <n> 12-27

DISPlay subsystem 12-28
:DISPlay:BUFFer:ACTive 12-28
:DISPlay:CLEar 12-29
:DISPlay:<function>:DIGits 12-29
:DISPlay:LIGHt:STATe 12-30
:DISPlay:READing:FORMat 12-31
:DISPlay:SCReen 12-32
:DISPlay:USER<n>:TEXT[:DATA] 12-33

FORMat subsystem 12-34
:FORMat:ASCii:PRECision 12-34
:FORMat:BORDer 12-35
:FORMat[:DATA] 12-36

OUTPut subsystem 12-37
:OUTPut[1]:<function>:SMODe 12-37
:OUTPut[1]:INTerlock:STATe 12-39
:OUTPut[1]:INTerlock:TRIPped? 12-40
:OUTPut[1][:STATe] 12-41

ROUTe subsystem 12-41
:ROUTe:TERMinals 12-41

SCRipt subsystem 12-42
:SCRipt:RUN 12-42

SENSe1 subsystem 12-43
[:SENSe[1]]:<function>:AVERage:COUNt 12-43

[:SENSe[1]]:<function>:AVERage[:STATe] ... 12-44
[:SENSe[1]]:<function>:AVERage:TCONtrol ... 12-45
[:SENSe[1]]:<function>:AZERo[:STATe] ... 12-47
[:SENSe[1]]:<function>:DELay:USER<n> ... 12-48
[:SENSe[1]]:<function>:NPLCycles ... 12-49
[:SENSe[1]]:<function>:OCOMpensated ... 12-50
[:SENSe[1]]:<function>:RANGe:AUTO ... 12-51
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit ... 12-52
[:SENSe[1]]:<function>:RANGe:AUTO:REBound ... 12-53
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit ... 12-54
[:SENSe[1]]:<function>:RANGe[:UPPer] ... 12-55
[:SENSe[1]]:<function>:RELative ... 12-56
[:SENSe[1]]:<function>:RELative:ACQuire ... 12-57
[:SENSe[1]]:<function>:RELative:STATe ... 12-58
[:SENSe[1]]:<function>:RSENse ... 12-59
[:SENSe[1]]:<function>:UNIT ... 12-60
[:SENSe[1]]:AZERo:ONCE ... 12-60
[:SENSe[1]]:CONFiguration:LIST:CATalog? ... 12-61
[:SENSe[1]]:CONFiguration:LIST:CREate ... 12-62
[:SENSe[1]]:CONFiguration:LIST:DELete ... 12-62
[:SENSe[1]]:CONFiguration:LIST:QUERy? ... 12-63
[:SENSe[1]]:CONFiguration:LIST:RECall ... 12-64
[:SENSe[1]]:CONFiguration:LIST:SIZE? ... 12-65
[:SENSe[1]]:CONFiguration:LIST:STORe ... 12-66
[:SENSe[1]]:COUNt ... 12-67
[:SENSe[1]]:FUNCtion[:ON] ... 12-68

SOURce subsystem ... 12-68
:SOURce[1]:CONFiguration:LIST:CATalog? ... 12-68
:SOURce[1]:CONFiguration:LIST:CREate ... 12-69
:SOURce[1]:CONFiguration:LIST:DELete ... 12-70
:SOURce[1]:CONFiguration:LIST:QUERy? ... 12-71
:SOURce[1]:CONFiguration:LIST:RECall ... 12-72
:SOURce[1]:CONFiguration:LIST:SIZE? ... 12-73
:SOURce[1]:CONFiguration:LIST:STORe ... 12-73
:SOURce[1]:<function>:DELay ... 12-74
:SOURce[1]:<function>:DELay:AUTO ... 12-75
:SOURce[1]:<function>:DELay:USER<n> ... 12-76
:SOURce[1]:<function>:HIGH:CAPacitance ... 12-77
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] ... 12-78
:SOURce[1]:<function>:<x>LIMit[:LEVel] ... 12-79
:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped? ... 12-80
:SOURce[1]:FUNCtion[:MODE] ... 12-80
:SOURce[1]:<function>:PROTection[:LEVel] ... 12-81
:SOURce[1]:<function>:PROTection[:LEVel]:TRIPped? ... 12-82
:SOURce[1]:<function>:RANGe ... 12-82
:SOURce[1]:<function>:RANGe:AUTO ... 12-84
:SOURce[1]:<function>:READ:BACK ... 12-85
:SOURce[1]:LIST:<function> ... 12-86
:SOURce[1]:LIST:<function>:APPend ... 12-87
:SOURce[1]:LIST:<function>:POINts? ... 12-88
:SOURce[1]:SWEep:<function>:LINear ... 12-89
:SOURce[1]:SWEep:<function>:LINear:STEP ... 12-91
:SOURce[1]:SWEep:<function>:LIST ... 12-93
:SOURce[1]:SWEep:<function>:LOG ... 12-95

STATus subsystem ... 12-97
:STATus:CLEar ... 12-97
:STATus:OPERation:CONDition? ... 12-98
:STATus:OPERation:ENABle ... 12-98
:STATus:OPERation[:EVENt]? ... 12-99
:STATus:OPERation:MAP ... 12-99

:STATus:PRESet 12-100
:STATus:QUEStionable:CONDition? 12-101
:STATus:QUEStionable:ENABle 12-101
:STATus:QUEStionable:MAP 12-102
:STATus:QUEStionable[:EVENt]? 12-103

SYSTem subsystem 12-103
:SYSTem:ACCess 12-103
:SYSTem:BEEPer[:IMMediate] 12-104
:SYSTem:BREakdown:PROTection 12-105
:SYSTem:CLEar 12-106
:SYSTem:COMMunication:LAN:CONFigure 12-106
:SYSTem:COMMunication:LAN:MACaddress? 12-107
:SYSTem:ERRor[:NEXT]? 12-108
:SYSTem:ERRor:CODE[:NEXT]? 12-108
:SYSTem:ERRor:COUNt? 12-109
:SYSTem:EVENtlog:COUNt? 12-109
:SYSTem:EVENtlog:NEXT? 12-110
:SYSTem:EVENtlog:POST 12-111
:SYSTem:EVENtlog:SAVE 12-112
:SYSTem:GPIB:ADDRess 12-113
:SYSTem:LFRequency? 12-113
:SYSTem:PASSword:NEW 12-114
:SYSTem:POSetup 12-115
:SYSTem:TIME 12-116
:SYSTem:VERSion? 12-117

TRACe subsystem 12-117
:TRACe:ACTual? 12-117
:TRACe:ACTual:END? 12-118
:TRACe:ACTual:STARt? 12-119
:TRACe:CLEar 12-120
:TRACe:DATA? 12-121
:TRACe:DELete 12-124
:TRACe:FILL:MODE 12-124
:TRACe:LOG:STATe 12-125
:TRACe:MAKE 12-126
:TRACe:MATH 12-128
:TRACe:POINts 12-130
:TRACe:SAVE 12-131
:TRACe:SAVE:APPend 12-133
:TRACe:STATistics:AVERage? 12-134
:TRACe:STATistics:CLEar 12-135
:TRACe:STATistics:MAXimum? 12-136
:TRACe:STATistics:MINimum? 12-136
:TRACe:STATistics:PK2Pk? 12-137
:TRACe:STATistics:STDDev? 12-138
:TRACe:TRIGger 12-138
:TRACe:UNIT 12-139
:TRACe:WRITe:FORMat 12-141
:TRACe:WRITe:READing 12-143

TRIGger subsystem 12-145
:ABORt 12-145
:INITiate[:IMMediate] 12-145
:TRIGger:BLENder<n>:CLEar 12-146
:TRIGger:BLENder<n>:MODE 12-146
:TRIGger:BLENder<n>:OVERrun? 12-147
:TRIGger:BLENder<n>:STIMulus<m> 12-148
:TRIGger:BLOCk:BRANch:ALWays 12-149
:TRIGger:BLOCk:BRANch:COUNter 12-150
:TRIGger:BLOCk:BRANch:COUNter:COUNt? 12-151

:TRIGger:BLOCk:BRANch:COUNter:RESet ... 12-152
:TRIGger:BLOCk:BRANch:DELTa ... 12-153
:TRIGger:BLOCk:BRANch:EVENt ... 12-154
:TRIGger:BLOCk:BRANch:LIMit:CONStant ... 12-155
:TRIGger:BLOCk:BRANch:LIMit:DYNamic ... 12-156
:TRIGger:BLOCk:BRANch:ONCE ... 12-158
:TRIGger:BLOCk:BRANch:ONCE:EXCLuded ... 12-158
:TRIGger:BLOCk:BUFFer:CLEar ... 12-159
:TRIGger:BLOCk:CONFig:NEXT ... 12-160
:TRIGger:BLOCk:CONFig:PREVious ... 12-161
:TRIGger:BLOCk:CONFig:RECall ... 12-162
:TRIGger:BLOCk:DELay:CONStant ... 12-163
:TRIGger:BLOCk:DELay:DYNamic ... 12-164
:TRIGger:BLOCk:DIGital:IO ... 12-165
:TRIGger:BLOCk:LIST? ... 12-166
:TRIGger:BLOCk:LOG:EVENt ... 12-166
:TRIGger:BLOCk:MDIGitize ... 12-167
:TRIGger:BLOCk:NOP ... 12-170
:TRIGger:BLOCk:NOTify ... 12-170
:TRIGger:BLOCk:SOURce:STATe ... 12-171
:TRIGger:BLOCk:WAIT ... 12-172
:TRIGger:CONTinuous ... 12-174
:TRIGger:DIGital<n>:IN:CLEar ... 12-175
:TRIGger:DIGital<n>:IN:EDGE ... 12-175
:TRIGger:DIGital<n>:IN:OVERrun? ... 12-176
:TRIGger:DIGital<n>:OUT:LOGic ... 12-177
:TRIGger:DIGital<n>:OUT:PULSewidth ... 12-177
:TRIGger:DIGital<n>:OUT:STIMulus ... 12-178
:TRIGger:LAN<n>:IN:CLEar ... 12-179
:TRIGger:LAN<n>:IN:EDGE ... 12-179
:TRIGger:LAN<n>:IN:OVERrun? ... 12-180
:TRIGger:LAN<n>:OUT:CONNect:STATe ... 12-181
:TRIGger:LAN<n>:OUT:IP:ADDRess ... 12-181
:TRIGger:LAN<n>:OUT:LOGic ... 12-182
:TRIGger:LAN<n>:OUT:PROTocol ... 12-183
:TRIGger:LAN<n>:OUT:STIMulus ... 12-184
:TRIGger:LOAD "ConfigList" ... 12-185
:TRIGger:LOAD "DurationLoop" ... 12-186
:TRIGger:LOAD "Empty" ... 12-187
:TRIGger:LOAD "GradeBinning" ... 12-188
:TRIGger:LOAD "LogicTrigger" ... 12-190
:TRIGger:LOAD "LoopUntilEvent" ... 12-191
:TRIGger:LOAD "SimpleLoop" ... 12-192
:TRIGger:LOAD "SortBinning" ... 12-193
:TRIGger:PAUSe ... 12-195
:TRIGger:RESume ... 12-196
:TRIGger:STATe? ... 12-196
:TRIGger:TIMer<n>:CLEar ... 12-197
:TRIGger:TIMer<n>:COUNt ... 12-197
:TRIGger:TIMer<n>:DELay ... 12-199
:TRIGger:TIMer<n>:STARt:FRACtional ... 12-199
:TRIGger:TIMer<n>:STARt:GENerate ... 12-200
:TRIGger:TIMer<n>:STARt:OVERrun? ... 12-200
:TRIGger:TIMer<n>:STARt:SEConds ... 12-201
:TRIGger:TIMer<n>:STARt:STIMulus ... 12-202
:TRIGger:TIMer<n>:STATe ... 12-203

Introduction to TSP commands ... 13-1

Introduction to TSP operation ... 13-1

Controlling the instrument by sending individual command messages 13-1
Queries 13-3
USB flash drive path 13-3
Information on scripting and programming 13-4

Fundamentals of scripting for TSP 13-4
What is a script? 13-4
Run-time and nonvolatile memory storage of scripts 13-5
What can be included in scripts? 13-5
Working with scripts 13-5

Fundamentals of programming for TSP 13-12
What is Lua? 13-13
Lua basics 13-13
Standard libraries 13-27

Test Script Builder (TSB) 13-31
Installing the TSB software 13-31
Installing the TSB add-in 13-31
Using Test Script Builder (TSB) 13-32
Project navigator 13-33
Script editor 13-34
Outline view 13-34
Programming interaction 13-34
Connecting an instrument in TSB 13-35
Creating a new TSP project 13-36
Adding a new TSP file to a project 13-37
Running a script 13-37
Creating a run configuration 13-38

Memory considerations for the run-time environment 13-41

About TSP Commands 13-42
Beeper control 13-42
Digital I/O 13-43
Configuration list 13-43
Display 13-44
Event log 13-44
File 13-44
Instrument identification 13-45
Miscellaneous 13-45
LAN 13-45
GPIB 13-46
Reading buffer 13-46
Reset 13-47
Queries and response messages 13-47
Scripting 13-48
SMU 13-48
Status model 13-50
Time 13-50
Triggering 13-51
Trigger model 13-52
TSP-Link 13-53
TSP-net 13-53
User strings 13-54

TSP command reference 14-1

TSP command programming notes 14-1
TSP syntax rules 14-1
Time and date values 14-2
Local and remote control 14-3

Using the TSP command reference ... 14-4
Command name, brief description, and summary table ... 14-5
Command usage ... 14-6
Command details ... 14-7
Example section ... 14-7
Related commands and information ... 14-7

TSP commands ... 14-8
available() ... 14-8
beeper.beep() ... 14-8
buffer.clearstats() ... 14-9
buffer.delete() ... 14-10
buffer.getstats() ... 14-11
buffer.make() ... 14-13
buffer.math() ... 14-15
buffer.save() ... 14-18
buffer.saveappend() ... 14-19
buffer.unit() ... 14-21
bufferVar.capacity ... 14-22
bufferVar.clear() ... 14-23
bufferVar.dates ... 14-24
bufferVar.endindex ... 14-25
bufferVar.extraformattedvalues ... 14-26
bufferVar.extravalues ... 14-28
bufferVar.extravalueunits ... 14-29
bufferVar.fillmode ... 14-30
bufferVar.formattedreadings ... 14-31
bufferVar.fractionalseconds ... 14-32
bufferVar.logstate ... 14-33
bufferVar.n ... 14-33
bufferVar.readings ... 14-34
bufferVar.relativetimestamps ... 14-35
bufferVar.seconds ... 14-36
bufferVar.sourceformattedvalues ... 14-37
bufferVar.sourcestatuses ... 14-38
bufferVar.sourceunits ... 14-40
bufferVar.sourcevalues ... 14-41
bufferVar.startindex ... 14-42
bufferVar.statuses ... 14-43
bufferVar.times ... 14-44
bufferVar.timestamps ... 14-45
bufferVar.units ... 14-47
buffer.write.format() ... 14-48
buffer.write.reading() ... 14-50
createconfigscript() ... 14-52
dataqueue.add() ... 14-53
dataqueue.CAPACITY ... 14-54
dataqueue.clear() ... 14-54
dataqueue.count ... 14-55
dataqueue.next() ... 14-56
delay() ... 14-57
digio.line[N].mode ... 14-58
digio.line[N].reset() ... 14-59
digio.line[N].state ... 14-61
digio.readport() ... 14-61
digio.writeport() ... 14-62
display.activebuffer ... 14-63
display.changescreen() ... 14-64
display.clear() ... 14-65
display.delete() ... 14-65
display.input.number() ... 14-66

display.input.option() 14-68
display.input.prompt() 14-69
display.input.string() 14-70
display.lightstate 14-72
display.prompt() 14-73
display.readingformat 14-74
display.settext() 14-75
display.waitevent() 14-76
eventlog.clear() 14-77
eventlog.getcount() 14-77
eventlog.next() 14-78
eventlog.post() 14-80
eventlog.save() 14-81
exit() 14-82
file.close() 14-82
file.flush() 14-83
file.mkdir() 14-84
file.open() 14-85
file.read() 14-86
file.usbdriveexists() 14-86
file.write() 14-88
format.asciiprecision 14-88
format.byteorder 14-89
format.data 14-90
fs.chdir() 14-91
fs.cwd() 14-92
fs.is_dir() 14-92
fs.is_file() 14-93
fs.mkdir() 14-94
fs.readdir() 14-94
fs.rmdir() 14-95
gpib.address 14-96
lan.ipconfig() 14-97
lan.lxidomain 14-98
lan.macaddress 14-98
localnode.access 14-99
localnode.gettime() 14-100
localnode.linefreq 14-100
localnode.model 14-101
localnode.password 14-101
localnode.prompts 14-102
localnode.prompts4882 14-103
localnode.serialno 14-103
localnode.settime() 14-104
localnode.showevents 14-105
localnode.version 14-106
node[N].execute() 14-106
node[N].getglobal() 14-107
node[N].setglobal() 14-108
opc() 14-108
print() 14-109
printbuffer() 14-110
printnumber() 14-113
reset() 14-114
script.catalog() 14-114
script.delete() 14-115
script.load() 14-116
scriptVar.run() 14-116
scriptVar.save() 14-117
scriptVar.source 14-118
smu.breakdownprotection 14-118

smu.interlock.enable 14-120
smu.interlock.tripped 14-121
smu.measure.autorange 14-122
smu.measure.autorangehigh 14-123
smu.measure.autorangelow 14-124
smu.measure.autorangerebound 14-125
smu.measure.autozero.enable 14-126
smu.measure.autozero.once() 14-127
smu.measure.configlist.catalog() 14-127
smu.measure.configlist.create() 14-128
smu.measure.configlist.delete() 14-129
smu.measure.configlist.query() 14-130
smu.measure.configlist.recall() 14-131
smu.measure.configlist.size() 14-132
smu.measure.configlist.store() 14-133
smu.measure.configlist.storefunc() 14-134
smu.measure.count 14-135
smu.measure.displaydigits 14-138
smu.measure.filter.count 14-139
smu.measure.filter.enable 14-139
smu.measure.filter.type 14-140
smu.measure.func 14-142
smu.measure.getattribute() 14-143
smu.measure.limit[Y].audible 14-143
smu.measure.limit[Y].autoclear 14-144
smu.measure.limit[Y].clear() 14-145
smu.measure.limit[Y].enable 14-146
smu.measure.limit[Y].fail 14-147
smu.measure.limit[Y].high.value 14-149
smu.measure.limit[Y].low.value 14-150
smu.measure.math.enable 14-151
smu.measure.math.format 14-152
smu.measure.math.mxb.bfactor 14-153
smu.measure.math.mxb.mfactor 14-154
smu.measure.math.percent 14-155
smu.measure.nplc 14-156
smu.measure.offsetcompensation 14-157
smu.measure.range 14-158
smu.measure.read() 14-159
smu.measure.readwithtime() 14-160
smu.measure.rel.acquire() 14-161
smu.measure.rel.enable 14-162
smu.measure.rel.level 14-163
smu.measure.sense 14-164
smu.measure.setattribute() 14-165
smu.measure.unit 14-167
smu.measure.userdelay[N] 14-168
smu.reset() 14-169
smu.source.autorange 14-170
smu.source.autodelay 14-171
smu.source.configlist.catalog() 14-172
smu.source.configlist.create() 14-172
smu.source.configlist.delete() 14-173
smu.source.configlist.query() 14-174
smu.source.configlist.recall() 14-175
smu.source.configlist.size() 14-176
smu.source.configlist.store() 14-177
smu.source.configlist.storefunc() 14-178
smu.source.delay 14-179
smu.source.func 14-180
smu.source.getattribute() 14-180

smu.source.highc 14-181
smu.source.level 14-182
smu.source.offmode 14-183
smu.source.output 14-185
smu.source.protect.level 14-186
smu.source.protect.tripped 14-187
smu.source.range 14-187
smu.source.readback 14-189
smu.source.setattribute() 14-190
smu.source.sweeplinear() 14-191
smu.source.sweeplinearstep() 14-193
smu.source.sweeplist() 14-196
smu.source.sweeplog() 14-198
smu.source.userdelay[N] 14-200
smu.source.xlimit.level 14-201
smu.source.xlimit.tripped 14-202
smu.terminals 14-203
status.clear() 14-203
status.condition 14-204
status.operation.condition 14-205
status.operation.enable 14-205
status.operation.event 14-206
status.operation.getmap() 14-207
status.operation.setmap() 14-207
status.preset() 14-208
status.questionable.condition 14-209
status.questionable.enable 14-209
status.questionable.event 14-210
status.questionable.getmap() 14-211
status.questionable.setmap() 14-211
status.request_enable 14-212
status.standard.enable 14-213
status.standard.event 14-215
timer.cleartime() 14-216
timer.gettime() 14-216
trigger.blender[N].clear() 14-217
trigger.blender[N].orenable 14-217
trigger.blender[N].overrun 14-218
trigger.blender[N].reset() 14-219
trigger.blender[N].stimulus[M] 14-219
trigger.blender[N].wait() 14-221
trigger.clear() 14-221
trigger.continuous 14-222
trigger.digin[N].clear() 14-223
trigger.digin[N].edge 14-224
trigger.digin[N].overrun 14-225
trigger.digin[N].wait() 14-225
trigger.digout[N].assert() 14-226
trigger.digout[N].logic 14-227
trigger.digout[N].pulsewidth 14-228
trigger.digout[N].release() 14-228
trigger.digout[N].stimulus 14-229
trigger.lanin[N].clear() 14-230
trigger.lanin[N].edge 14-231
trigger.lanin[N].overrun 14-231
trigger.lanin[N].wait() 14-232
trigger.lanout[N].assert() 14-233
trigger.lanout[N].connect() 14-234
trigger.lanout[N].connected 14-234
trigger.lanout[N].disconnect() 14-235
trigger.lanout[N].ipaddress 14-236

trigger.lanout[N].logic 14-236
trigger.lanout[N].protocol 14-237
trigger.lanout[N].stimulus 14-238
trigger.model.abort() 14-239
trigger.model.getblocklist() 14-239
trigger.model.getbranchcount() 14-240
trigger.model.initiate() 14-241
trigger.model.load() — ConfigList 14-241
trigger.model.load() — DurationLoop 14-242
trigger.model.load() — Empty 14-243
trigger.model.load() — GradeBinning 14-244
trigger.model.load() — LogicTrigger 14-245
trigger.model.load() — LoopUntilEvent 14-246
trigger.model.load() — SimpleLoop 14-248
trigger.model.load() — SortBinning 14-250
trigger.model.pause() 14-251
trigger.model.resume() 14-252
trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS 14-253
trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER 14-253
trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA 14-254
trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT 14-255
trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC 14-257
trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT 14-258
trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE 14-260
trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED 14-260
trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR 14-261
trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT 14-262
trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV 14-263
trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL 14-265
trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT 14-266
trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC 14-267
trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO 14-268
trigger.model.setblock() — trigger.BLOCK_LOG_EVENT 14-269
trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE 14-270
trigger.model.setblock() — trigger.BLOCK_NOP 14-272
trigger.model.setblock() — trigger.BLOCK_NOTIFY 14-273
trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT 14-274
trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT 14-275
trigger.model.setblock() — trigger.BLOCK_WAIT 14-276
trigger.model.state() 14-278
trigger.timer[N].clear() 14-279
trigger.timer[N].count 14-279
trigger.timer[N].delay 14-281
trigger.timer[N].delaylist 14-281
trigger.timer[N].enable 14-282
trigger.timer[N].reset() 14-283
trigger.timer[N].start.fractionalseconds 14-284
trigger.timer[N].start.generate 14-285
trigger.timer[N].start.overrun 14-286
trigger.timer[N].start.seconds 14-286
trigger.timer[N].start.stimulus 14-287
trigger.timer[N].wait() 14-288
trigger.tsplinkin[N].clear() 14-289
trigger.tsplinkin[N].edge 14-289
trigger.tsplinkin[N].overrun 14-290
trigger.tsplinkin[N].wait() 14-291
trigger.tsplinkout[N].assert() 14-291
trigger.tsplinkout[N].logic 14-292
trigger.tsplinkout[N].pulsewidth 14-293
trigger.tsplinkout[N].release() 14-293
trigger.tsplinkout[N].stimulus 14-294

trigger.wait() 14-295
tsplink.group 14-296
tsplink.initialize() 14-296
tsplink.line[N].mode 14-298
tsplink.line[N].reset() 14-298
tsplink.line[N].state 14-299
tsplink.master 14-300
tsplink.node 14-301
tsplink.readport() 14-301
tsplink.state 14-302
tsplink.writeport() 14-303
tspnet.clear() 14-303
tspnet.connect() 14-304
tspnet.disconnect() 14-305
tspnet.execute() 14-306
tspnet.idn() 14-307
tspnet.read() 14-308
tspnet.readavailable() 14-309
tspnet.reset() 14-309
tspnet.termination() 14-310
tspnet.timeout 14-311
tspnet.tsp.abort() 14-311
tspnet.tsp.abortonconnect 14-312
tspnet.tsp.rbtablecopy() 14-313
tspnet.tsp.runscript() 14-314
tspnet.write() 14-314
upgrade.previous() 14-315
upgrade.unit() 14-316
userstring.add() 14-316
userstring.catalog() 14-317
userstring.delete() 14-318
userstring.get() 14-318
waitcomplete() 14-319

Common commands 15-1

Introduction 15-1
*CLS 15-2
*ESE 15-2
*ESR? 15-4
*IDN? 15-5
*LANG 15-5
*OPC 15-6
*RST 15-7
*SRE 15-7
*STB? 15-8
*TRG 15-9
*TST? 15-9
*WAI 15-10

Status model 16-1

Overview 16-1
Standard Event Register 16-3
Programmable status register sets 16-4
Status Byte Register 16-10
Queues 16-12

Serial polling and SRQ 16-13

Programming enable registers 16-14
Reading the registers 16-14
Understanding bit settings 16-15
Clearing registers 16-16
Status model programming examples 16-17
SRQ when the SMU reaches its source limit 16-17
SRQ when trigger model is finished 16-18
SRQ on trigger model notify event 16-19
SRQ on error 16-21
SRQ when reading buffer becomes full 16-21
SRQ when a measurement completes 16-22

Frequently asked questions 17-1
I see a command that is not in the manual. What is it? 17-1
How do I display the instrument's serial number? 17-2
What VISA resource name is required? 17-2
Can I use Keysight GPIB cards with Keithley drivers? 17-3
How do I check the USB driver for the device? 17-3
Which Microsoft Windows operating systems are supported? 17-4
What to do if the GPIB controller is not recognized? 17-5
I am receiving GPIB timeout errors. What should I do? 17-5
How do I change the command set? 17-5
How do I upgrade the firmware? 17-6
Where can I find updated drivers? 17-7
Why can't the 2470 read my USB flash drive? 17-8
How do I download measurements onto the USB flash drive? 17-8
How do I save the present state of the instrument? 17-9
Why did my settings change? 17-10
What is NPLC? 17-10
What are the Quick Setup options? 17-10
What is the output-off state? 17-11
Why is OVP displayed? 17-12
How do I store readings into the buffer? 17-13
What should I do if I get a 5074 interlock error? 17-14
How do I trigger a sweep? 17-15
What are source limits? 17-15
What is offset compensation? 17-16
What is a configuration list? 17-16
What does –113 "Undefined header" error mean? 17-17

Why do I see the "incompatible settings" message? 17-17
What does –410 "Query interrupted" error mean? 17-17
What does –420 "Query unterminated" error mean? 17-18
How do I use the digital I/O port? 17-18
How do I trigger other instruments? 17-18

Next steps 18-1
Additional 2470 information 18-1

Section 1

Introduction

In this section:

Welcome 1-1
Extended warranty 1-2
Contact information 1-2
Organization of manual sections 1-3
Features 1-4
General ratings 1-4

Welcome

Thank you for choosing a Keithley Instruments product. The 2470 High Voltage SourceMeter Instrument is a precise, low-noise instrument that combines a stable DC power supply with a repeatable, high-impedance multimeter. This instrument features intuitive setup and control, enhanced signal quality and range, and better resistivity and resistance capabilities than similar products on the market.

With its 1100 V and 10 fA capability, the 2470 is optimized for characterizing and testing high voltage, low leakage devices, materials, and modules, such as silicon carbide (SiC), gallium nitride (GaN), power MOSFETs, transient suppression devices, circuit protection devices, power modules, and batteries.

SMU instruments offer a highly flexible, four-quadrant voltage and current source/load coupled with precision voltage and current measurements. The 2470 instrument can be used as a:

- Precision power supply with voltage and current readback
- True current source
- Digital multimeter (DCV, DCI, ohms, and power with 6½-digit resolution)
- Precision electronic load
- Pulse generator
- Trigger controller

Extended warranty

Additional years of warranty coverage are available on many products. These valuable contracts protect you from unbudgeted service expenses and provide additional years of protection at a fraction of the price of a repair. Extended warranties are available on new and existing products. Contact your local Keithley Instruments office, sales partner, or distributor for details.

Contact information

If you have any questions after you review the information in this documentation, please contact your local Keithley Instruments office, sales partner, or distributor. You can also call the corporate headquarters of Keithley Instruments (toll-free inside the U.S. and Canada only) at 1-800-935-5595, or from outside the U.S. at +1-440-248-0400. For worldwide contact numbers, visit the Keithley Instruments website (tek.com/keithley).

Organization of manual sections

The information in this manual is organized into the following major categories:

- **Installation:** Dimensions, installation, power up, instrument access, and remote communications setup information.
- **Instrument description:** Descriptions of the hardware and on-screen components of the instrument.
- **Sourcing and measuring:** Detail about sourcing and measuring test connections. Also include descriptions of features such as relative offset, calculations, limit testing, and configuration lists.
- **Source-measure considerations:** Best practices and recommended procedures that can increase measurement speed, accuracy, and sensitivity.
- **Reading buffers:** How to use default and user-defined reading buffers to capture data.
- **Graphing:** How to use the graphing and histogram features to view data on the front panel.
- **Triggering:** How to start and synchronize source and measure actions on one or more instruments with a trigger event or a combination of trigger events that you set. Also includes discussion of the trigger model.
- **TSP-Link and TSP-NET:** Connecting multiple instruments for high-speed trigger synchronization between instruments. Includes how to control the instrument using TSP commands and Test Script Builder (TSB) software, TSP-Link system expansion, and TSP-Net.
- **Maintenance:** Instructions for routine maintenance of the instrument, including fuse replacement and firmware upgrades.
- **Introduction to SCPI commands:** Describes how to control the instrument using SCPI commands.
- **SCPI command reference:** Contains programming notes and an alphabetical listing of SCPI commands available for the 2470.
- **Introduction to TSP commands:** The basics of using Test Script Processor (TSP®) commands to control the instrument.
- **TSP command reference:** Programming notes and an alphabetical listing of TSP commands available for the 2470.
- **Common commands:** Contains descriptions of IEEE Std 488.2 common commands.
- **Status model:** Describes the 2470 status model.
- **Frequently asked questions:** Answers to commonly asked questions.
- **Next steps:** Sources of additional information.

The PDF version of this manual contains bookmarks for each section. The manual sections are also listed in the Table of Contents at the beginning of this manual.

For more information about bookmarks, see Adobe® Acrobat® or Reader® help.

Features

The 2470 offers the following features:

- One tightly-coupled instrument that combines capabilities from analyzers, curve tracers, and I-V systems at a fraction of their cost.
- Wide coverage — up to 1100 V / 1 A DC, 20 W maximum.
- Five-inch, high resolution capacitive touchscreen GUI.
- 0.012% basic measure accuracy with 6½ digits.
- Source and sink (4-quadrant) operation.
- Four Quickset modes for fast setup and measurements.
- Context-sensitive help.
- Front-panel input banana jacks; rear-panel high-voltage input triaxial connections.
- SCPI and TSP® scripting programming modes.
- Front-panel USB 2.0 memory I/O port for transferring data, test scripts, or test configurations.

General ratings

The 2470 instrument's general ratings and connections are listed in the following table.

Category	**Specification**
Supply voltage range	100 V_{RMS} to 240 V_{RMS}, 50 Hz or 60 Hz (automatically detected at power up)
Input and output connections	See Rear panel overview (on page 3-3)
Environmental conditions	For indoor use only **Altitude:** Maximum 2000 meters (6562 feet) above sea level **Operating:** 0 °C to 50 °C, 70% relative humidity up to 35 °C; derate 3% relative humidity per °C, 35 °C to 50 °C **Storage:** –25 °C to 65 °C **Pollution category:** 2

Section 2

Installation

In this section:

Dimensions 2-1
Handle and bumpers 2-4
Instrument power 2-6
Remote communications interfaces 2-7
Interface access 2-34
Determining the command set you will use 2-36
System information 2-37

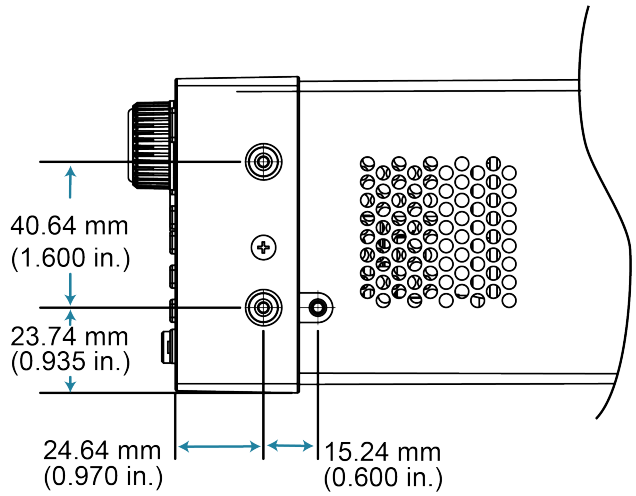
Dimensions

The following figures show the mounting screw locations and the dimensions of the instrument with and without the handle and bumpers.

The instrument weighs 4.54 kg (10.0 lb) with the bumpers and handle and 4.08 kg (9 lb) without them.

The following figure shows the mounting screw locations and dimensions. Mounting screws must be #6-32 with a maximum screw length of 11.12 mm (0.438 in.) or 7/16 in. The dimensions shown are typical for both sides of the instrument.

Figure 1: 2470 mounting screw locations and dimensions



The following figures show the dimensions when the handle and bumpers are installed.

Figure 2: 2470 dimensions front and rear with handle and bumpers

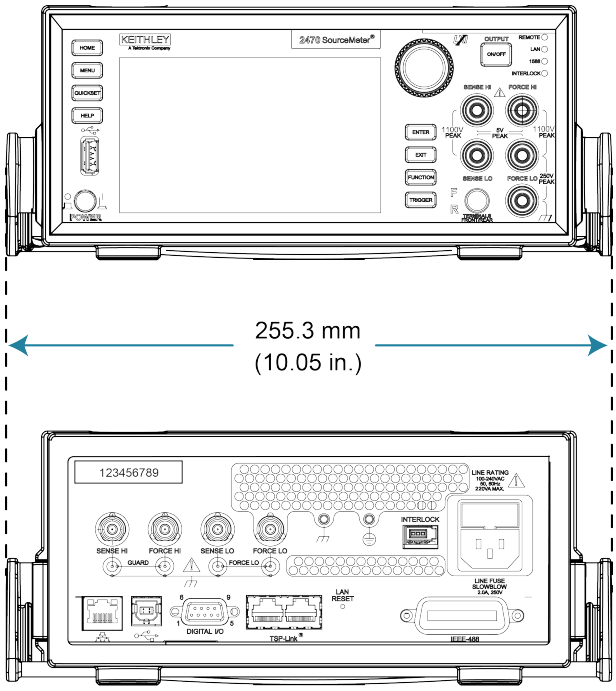
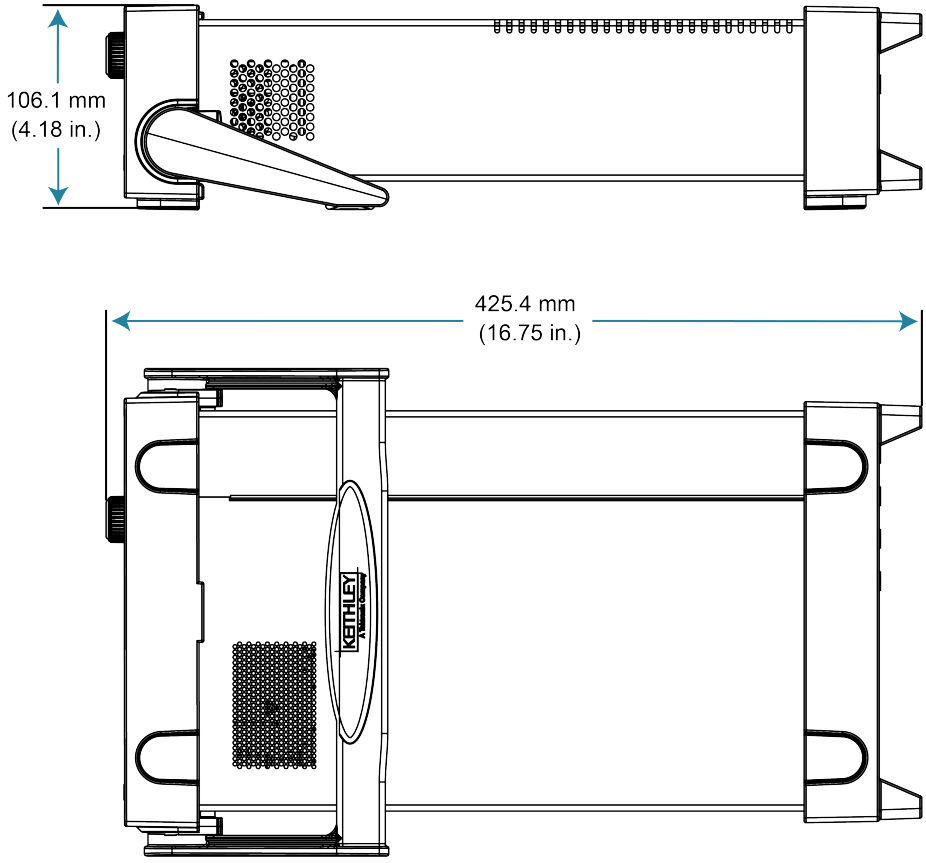


Figure 3: 2470 dimensions side and top with handle and bumpers



The following figures show the dimensions when the handle and bumpers have been removed.

Figure 4: 2470 front and rear panel dimensions with handle and bumpers removed

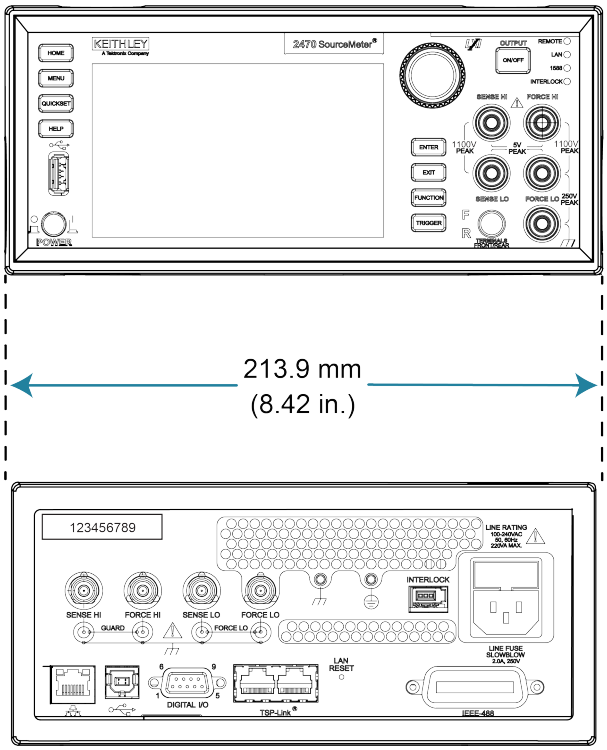
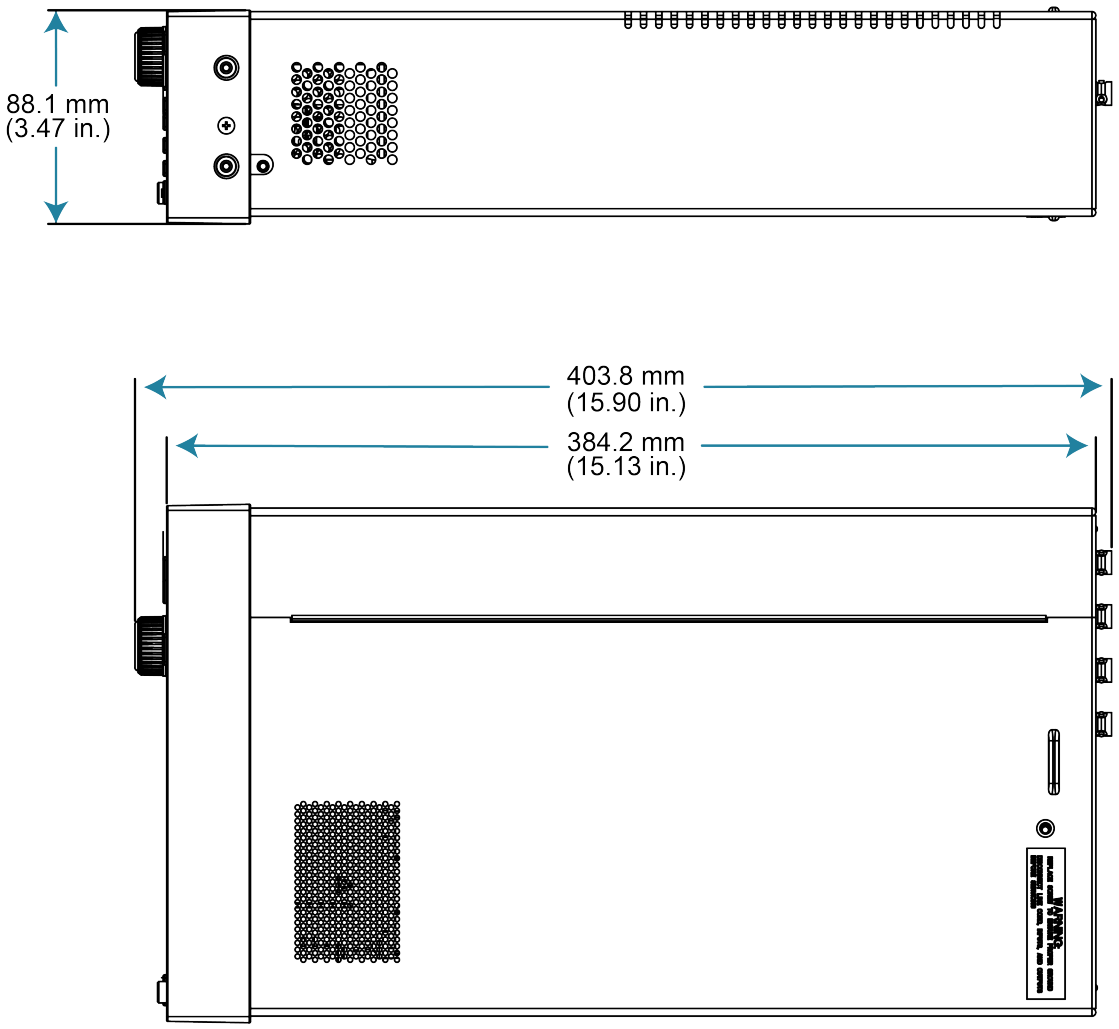


Figure 5: 2470 top and side dimensions with handle and bumpers removed



Handle and bumpers

The 2470 has a handle and front and rear bumpers for using the instrument on a benchtop. The handle rotates so that you can swing it below the bottom surface of the instrument to tilt the instrument up for easier front-panel viewing or carry the instrument.

Removing the handle and bumpers

You can remove handle and bumpers on the 2470 if you want to mount the instrument in a rack.

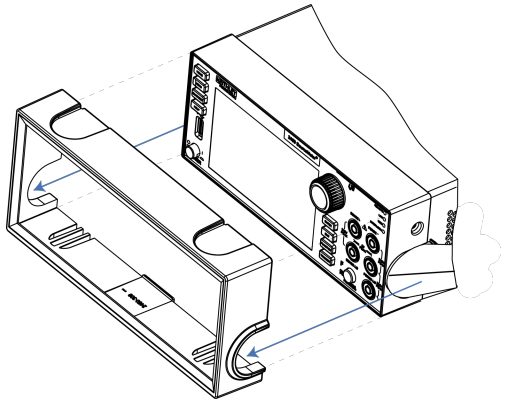
NOTE

If you remove the handle and bumpers, be sure to store them for future benchtop use.

To remove the bumpers:

1. Swivel the handle to a position above or below the instrument so that it will not interfere with the removal of the front bumper.
2. Grasp the front bumper on each side of the 2470 and gently pull it toward you until the bumper comes off the instrument.

Figure 6: Removing the front bumper



NOTE

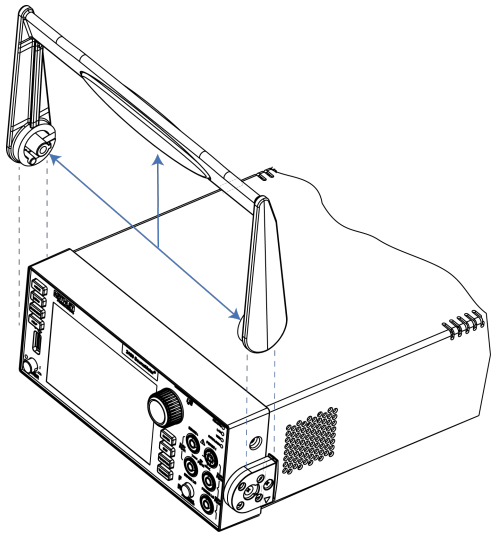
Remove all connections to the rear panel of the 2470 before removing the rear bumper.

3. To remove the rear bumper, repeat the procedure in step 2.

To remove the handle assembly:

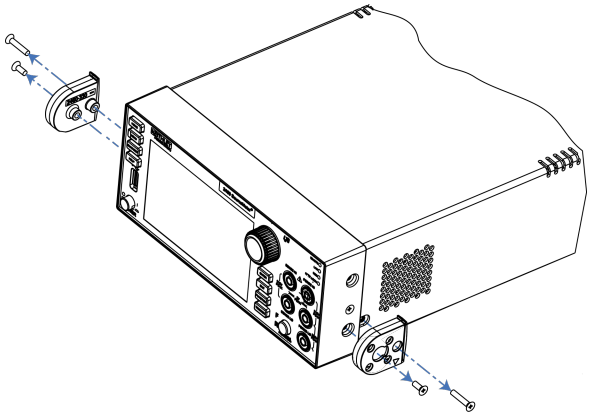
1. Grasp the sides of the handle near where it attaches to the instrument on both sides and gently pull the handle ends apart to widen the handle as you slide it over the instrument case.

Figure 7: Removing the handle



2. Use a Phillips screwdriver to loosen and remove the two screws holding the handle-mount assembly to one side of the 2470. The handle-mount assembly will fall away from the instrument chassis when the screws are removed.

Figure 8: Removing the handle mount



3. Repeat step 2 on the other side of the 2470.
4. Store the handle-mount assembly, screws, and handle together for future use.

Instrument power

Follow the steps below to connect the 2470 to line power and turn on the instrument. The 2470 operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz or 60 Hz. It automatically senses line voltage and frequency. Make sure the operating voltage in your area is compatible.

You must turn on the 2470 and allow it to warm up for at least one hour to achieve rated accuracies.

CAUTION

Operating the instrument on an incorrect line voltage may cause damage to the instrument, possibly voiding the warranty.

⚠ WARNING

The power cord supplied with the 2470 contains a separate protective earth (safety ground) wire for use with grounded outlets. When proper connections are made, the instrument chassis is connected to power-line ground through the ground wire in the power cord. In addition, a redundant protective earth connection is provided through a screw on the rear panel. This terminal should be connected to a known protective earth. In the event of a failure, not using a properly grounded protective earth and grounded outlet may result in personal injury or death due to electric shock.

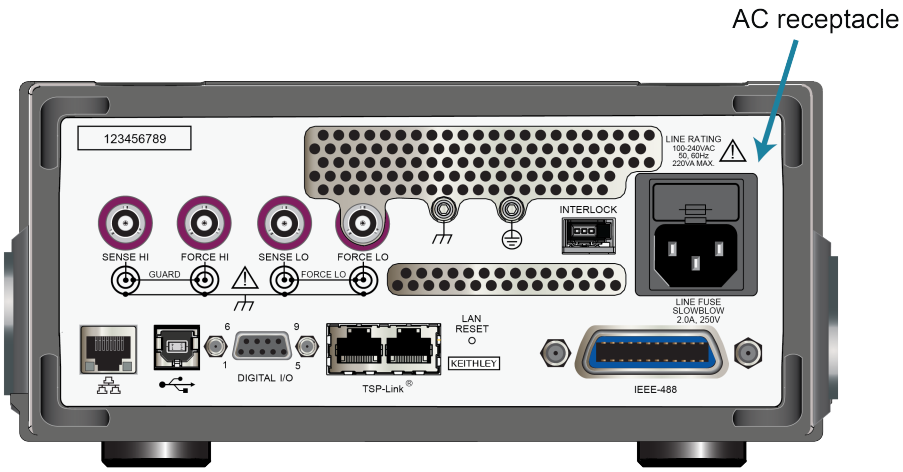
Do not replace detachable mains supply cords with inadequately rated cords. Failure to use properly rated cords may result in personal injury or death due to electric shock.

Connect the power cord

To connect the power cord:

1. Make sure that the front-panel POWER switch is in the off (O) position.
2. Connect the female end of the supplied power cord to the AC receptacle on the rear panel.
3. Connect the male end of the power cord to a grounded AC outlet.

Figure 9: 2470 AC receptacle on rear panel



Power the instrument on or off

Before turning the instrument on, disconnect any devices under test (DUTs) from the 2470.

To turn your instrument on, press the front-panel **POWER** switch to place it in the on (|) position. The instrument displays a status bar as it powers on. The home screen is displayed when power on is complete.

To turn your instrument off, press the front-panel **POWER** switch to place it in the off (O) position.

Remote communications interfaces

You can choose from one of several communication interfaces to send commands to and receive responses from the 2470.

You can control the 2470 from only one communications interface at a time. The first interface on which the instrument receives a message takes control of the instrument. If another interface sends a message, that interface can take control of the instrument. You may need to enter a password to change the interface, depending on the setting of interface access.

The 2470 automatically detects the type of communications interface (LAN, USB, or GPIB) when you connect to the respective port on the rear panel of the instrument. In most cases, you do not need to configure anything on the instrument. In addition, you do not need to reboot if you change the type of interface that is connected.

Supported remote interfaces

The 2470 supports the following remote interfaces:

- **GPIB:** IEEE-488 instrumentation general-purpose interface bus
- **USB:** Type B USB port
- **Ethernet:** Local-area-network communications
- **TSP-Link:** A high-speed trigger synchronization and communications bus that test system builders can use to connect multiple instruments in a master-and-subordinate configuration

For details about TSP-Link, see TSP-Link System Expansion Interface (on page 9-1).

Comparison of the communications interfaces

The following topics discuss some of the advantages and disadvantages of the communications interfaces that are available for the 2470.

Simplicity

The GPIB interface is the simplest configuration. Connections are simple, and the only necessary software configuration is setting the instrument address.

An ethernet network is a simple configuration if you can use the automatic settings. It is more complicated if you need to set it up manually. If you must set up your ethernet network manually, you need some knowledge of networking. In addition, your corporate information technology (IT) department may have restrictions that prevent using an ethernet network.

A USB interface is also simple to set up. However, it requires an instrument-specific device driver to communicate with the instrument. This can limit the operating systems that are available for use with the instrument.

Triggering

The GPIB interface provides the fastest, most consistent triggering. It has the lowest trigger latency of the available communications types. Trigger latency is the time that it takes the trigger to go from the computer to the instrument. GPIB also allows you to send triggers to multiple instruments simultaneously.

If you use a USB interface, it is difficult to synchronize triggers that are sent to multiple instruments. For applications that require synchronized triggering, you must use digital I/O. The trigger latency with a USB interface is higher than latency with a GPIB interface, but it is lower and more consistent than latency with an ethernet interface.

Transfer rate

Of the available interfaces, USB has the fastest transfer rate, followed by the ethernet and GPIB interfaces. The GPIB interface, however, offers the most consistent transfer rate.

Instrument naming

Names for instruments that are named through NI-VISA™ are in a human-readable format. USB instrument names are not intended to be human-readable.

Distance and instrument limitations

For GPIB and USB interfaces, the cabling distances between the controller and instrument or hub are limited to 30 feet. In a system connected with GPIB or USB, you can have up to 15 instruments attached to each controller.

The distances for ethernet interfaces are unlimited if the ethernet address of the instrument and ports for the various services it uses are visible publicly (for example, port 80 for web service). If you are using an ethernet interface, you can communicate with an instrument anywhere in the world. In a system that is connected through ethernet, the number of instruments you can attach to each controller is only limited by the controller and the connections available on that controller.

Expense

The GPIB interface is the most expensive method because of the costs for cabling and related equipment. Ethernet and USB connections are inexpensive options because most computers have built-in ethernet and USB ports. In addition, cables and hubs for ethernet and USB interfaces are inexpensive.

GPIB setup

This topic contains information about GPIB standards, bus connections, and primary address selection.

The 2470 GPIB interface is IEEE Std 488.1 compliant and supports IEEE Std 488.2 common commands and status model topology.

You can have up to 15 devices connected to a GPIB interface, including the controller. The maximum cable length is the lesser of either:

- The number of devices multiplied by 2 m (6.5 ft)
- 20 m (65.6 ft)

You may see erratic bus operation if you ignore these limits.

Install the GPIB driver software

Check the documentation for your GPIB controller for information about where to acquire drivers. Keithley Instruments also recommends that you check the website of the GPIB controller for the latest version of drivers or software.

It is important that you install the drivers before you connect the hardware. This prevents associating the incorrect driver to the hardware.

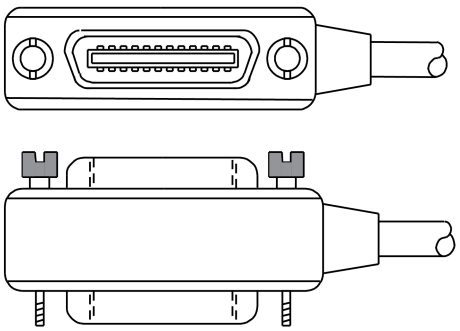
Install the GPIB cards in your computer

Refer to the documentation from the GPIB controller vendor for information about installing the GPIB controllers.

Connect GPIB cables to your instrument

To connect a 2470 to the GPIB interface, use a cable equipped with standard GPIB connectors, as shown below.

Figure 10: GPIB connector

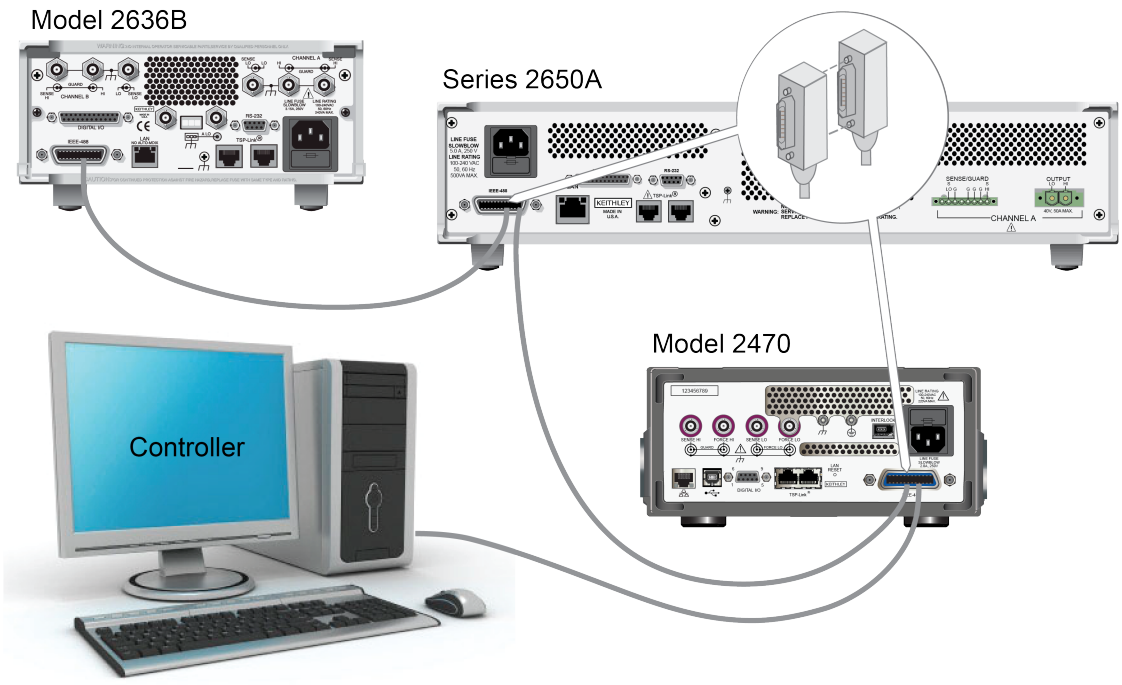


To allow many parallel connections to one instrument, stack the connectors. Each connector has two screws on it to ensure that connections remain secure. The figure below shows a typical connection diagram for a test system with multiple instruments.

CAUTION

To avoid possible mechanical damage, stack no more than three connectors on any one instrument. To minimize interference caused by electromagnetic radiation, use only shielded GPIB cables. Contact Keithley Instruments for shielded cables.

Figure 11: IEEE-488 connection example



To connect the GPIB cable to the instrument:

1. Align the cable connector with the connector on the 2470 rear panel.
2. Attach the connector. Tighten the screws securely but do not overtighten them.
3. Connect any additional connectors from other instruments, as required for your application.
4. Make sure that the end of the cable is properly connected to the controller.

Set the GPIB address

The default GPIB address is 18. You can set the address from 1 to 30 if it is unique in the system. This address cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

NOTE

GPIB controllers are usually set to 0 or 21. To be safe, do not configure any instrument to have an address of 21.

The instrument saves the address in nonvolatile memory. It does not change when you send a reset command or when you turn the power off and on again.

To set the GPIB address from the front panel:

1. Press the **MENU** key.
2. Select **Communication**.
3. Select the **GPIB** tab.
4. Set the GPIB **Address**.
5. Select **OK**.

NOTE

You can also set the GPIB address using remote commands. Set the GPIB address with the SCPI command `:SYSTem:GPIB:ADDRess` or the TSP command `gpib.address`.

Effect of GPIB line events on 2470

The GPIB has control lines that allow predefined information, called events, to be transferred quickly. The following information lists some of the GPIB line events and how the 2470 reacts to them.

DCL

This event clears the GPIB interface. When the 2470 detects a device clear (DCL) event, it does the following:

- Clears the input buffer, output queue, and command queue
- Cancels deferred commands
- Clears any command that prevents the processing of any other device command

A DCL event does not affect instrument settings and stored data.

GET

The group execute trigger (GET) command is a GPIB trigger that triggers the instrument to take readings from a remote interface.

GTL

When the instrument detects the go to local (GTL) event, it exits remote operation and enters local operation. When the instrument is operating locally, you can control the instrument from the front panel.

IFC

When the instrument detects an interface clear (IFC) event, the instrument enters the talker and the listener idle state. When the instrument is in this state, the GPIB ↑↓ indicators on the front panel are not displayed.

An IFC event does not interrupt the transfer of command messages to and from the instrument. However, messages are suspended. If the transfer of a response message from the instrument is suspended by an IFC event, the transfer resumes when the instrument is addressed to talk. If transfer of a command message to the instrument is suspended by an IFC event, the rest of the message can be sent when the instrument is addressed to listen.

LLO

When the instrument detects a local-lockout (LLO) event, all front-panel controls except the OUTPUT ON/OFF and POWER switches are disabled.

To enable the front panel, use the go-to-local (GTL) event.

REN

When the instrument detects the remote enable (REN) event, it is set up for remote operation. The instrument is not placed in remote mode when it detects the REN event; the instrument must be addressed to listen after the REN event before it goes into remote mode.

You should place the instrument into remote mode before you attempt to program it using a remote interface.

SDC

The selective device clear (SDC) event is similar to the device clear (DCL) event. However, the SDC event clears the interface for an individual instrument instead of clearing the interface of all instruments.

When the 2470 detects an SDC event, it will do the following for the selected instrument:

- Clears the input buffer, output queue, and command queue
- Cancels deferred commands
- Clears any command that prevents the processing of any other device command

An SDC event does not affect instrument settings and stored data.

SPE, SPD

When the instrument detects the serial polling enable (SPE) and serial polling disable (SPD) events, it sends the status byte of the instrument. This contains the serial poll byte of the instrument.

The serial poll byte contains information about internal functions. See the Status model (on page 16-1) for detail. Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line.

LAN communications

You can communicate with the instrument using a local area network (LAN). The LAN interface can be used to build flexible test systems that include web access. This section provides an overview of LAN communications for the 2470.

When you connect using a LAN, you can use a web browser to access the internal web page of the instrument and change some of the instrument settings.

The 2470 is a version 1.5 LXI Device Specification 2016 instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet LAN). There is one LAN port (located on the rear panel of the instrument) that supports full connectivity on a 10 Mbps or 100 Mbps network. The 2470 automatically detects the speed.

The 2470 also supports Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are useful on a LAN with no central administration.

NOTE

Contact your network administrator to confirm your specific network requirements before setting up a LAN connection.

If you have problems setting up the LAN, refer to LAN troubleshooting suggestions (on page 2-22).

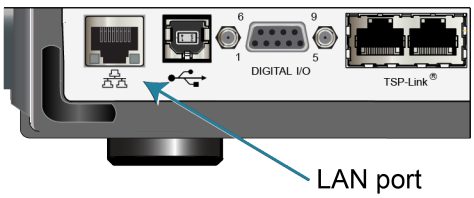
LAN cable connection

The 2470 includes a LAN crossover cable. You can use this cable for the TSP-Link® network or LAN communications.

However, you can use any standard LAN crossover cable (RJ-45, male-to-male) or straight-through cable to connect your equipment. The instrument automatically senses which cable you have connected.

The following figure shows the location of the LAN port on the rear panel of the instrument. Connect the LAN cable between this connection and the LAN port on the computer.

Figure 12: 2470 LAN port



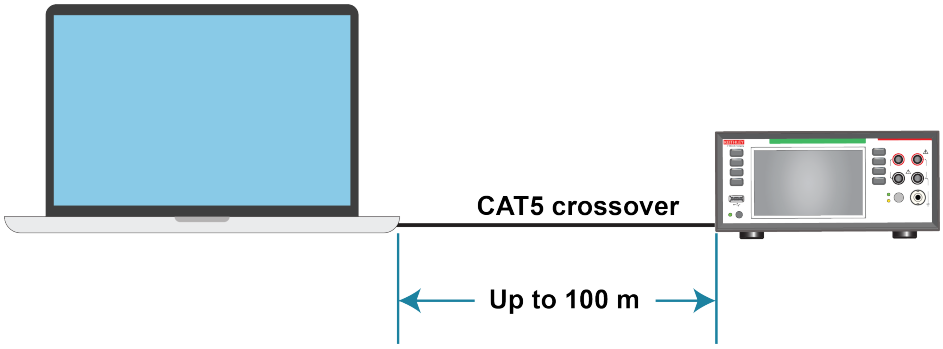
You can connect the instrument to the LAN in a one-to-one, one-to-many, two network card, or enterprise configuration, as described in the following topics.

One-to-one connection

With most instruments, a one-to-one connection is done only when you are connecting a single instrument to a single network interface card.

A one-to-one connection using a network crossover cable connection is similar to a typical RS-232 system using a null modem cable. The crossover cable has its receive (RX) and transmit (TX) lines crossed to allow the receive line input to be connected to the transmit line output on the network interfaces.

Figure 13: One-to-one connection with a crossover cable



NOTE

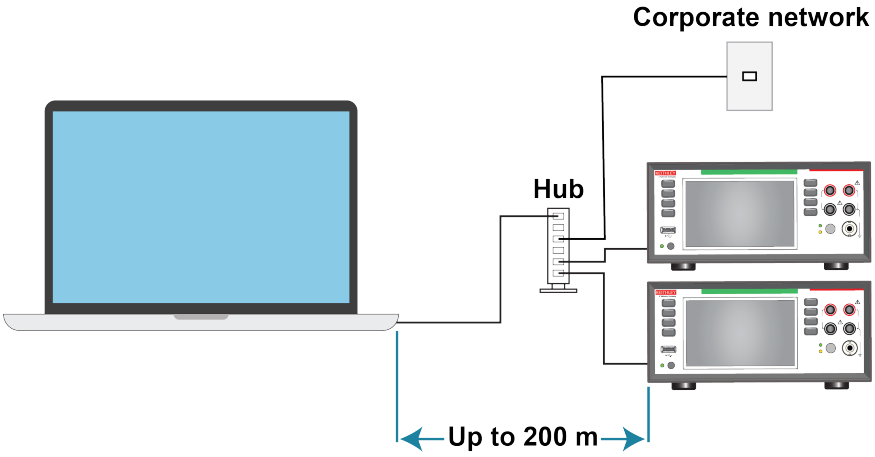
The 2470 supports Auto-MDIX and can use either normal LAN CAT-5 cables (patch) or crossover cables. The instrument automatically adjusts to support either cable.

One-to-many connection

With a LAN hub, a single network interface card can be connected to as many instruments as the hub can support. This requires straight-through network (not crossover) cables for hub connections.

The advantage of this method is easy expansion of measurement channels when the test requirements exceed the capacity of a single instrument. With only the instruments connected to the hub, this is an isolated instrumentation network. However, with a corporate network attached to the hub, the instruments become part of the larger network.

Figure 14: One-to-many connection using a network hub or switch

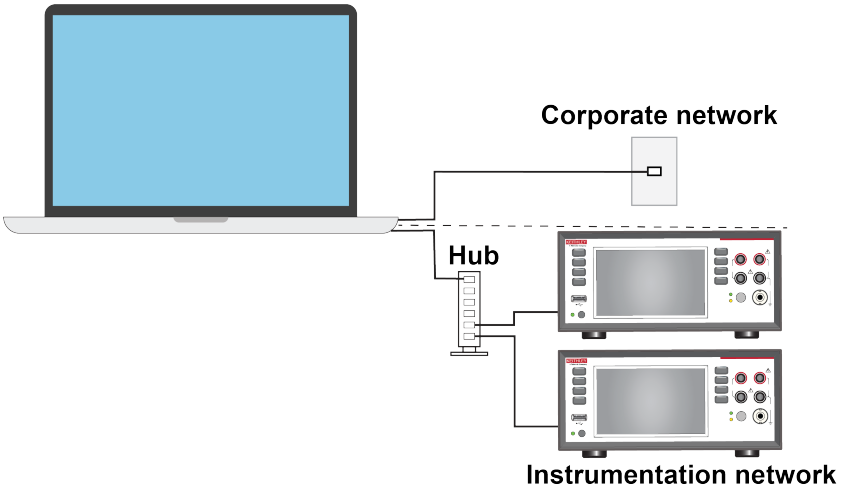


Two network card connection

If you need to connect independent corporate and instrumentation networks, two network interface cards are required in the computer controller. Though the two networks are independent, stations on the corporate network can access the instruments and the instruments can access the corporate network using the same computer.

This configuration resembles a GPIB setup in which the computer is connected to a corporate network, but also has a GPIB card in the computer to communicate with instruments.

Figure 15: Two network card connection

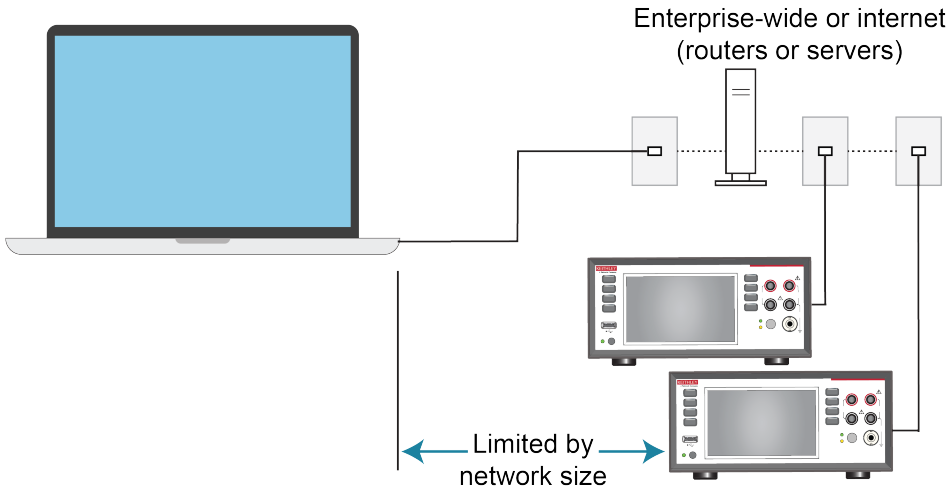


Instrumentation connection to enterprise routers or servers

This connection uses an existing network infrastructure to connect instruments to the computer controller. In this case, you must get the network resources from the network administrator.

Usually, the instruments are kept inside the corporate firewall, but the network administrator can assign resources that allow them to be outside the firewall. This allows instruments to be connected to the internet using appropriate security methods. Data collection and distribution can be controlled from virtually any location.

Figure 16: Instrumentation connection to enterprise routers or servers



Set up LAN communications on the instrument

This section describes how to set up manual or automatic LAN communications on the instrument.

Check communication settings

Before configuring the LAN, you can check the communications settings on the instrument without making any changes.

To check communications settings on the instrument:

1. Press the **MENU** key.
2. Under System, select **Communication**. The SYSTEM COMMUNICATIONS window opens.
3. Select one of the four tabs (**GPIB**, **USB**, **LAN**, or **TSP-Link**) to see the settings for that interface.
4. Press the **EXIT** key to leave the SYSTEM COMMUNICATIONS window without making any changes.

Set up automatic LAN configuration

If you are connecting to a LAN that has a DHCP server or if you have a direct connection between the instrument and a host computer, you can use automatic IP address selection.

If you select Auto, the instrument attempts to get an IP address from a DHCP server. If this fails, it reverts to an IP address in the range of 169.254.1.0 through 169.254.254.255.

NOTE

Both the host computer and the instrument should be set to use automatic LAN configuration. Though it is possible to have one set to manual configuration, it is more complicated to set up.

To set up automatic IP address selection using the front panel:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. For TCP/IP Mode, select **Auto**.
5. Select **Apply Settings** to save your settings.

Set up manual LAN configuration

If necessary, you can set the IP address on the instrument manually.

You can also enable or disable the DNS settings and assign a host name to the DNS server.

NOTE

Contact your corporate information technology (IT) department to secure a valid IP address for the instrument when placing the instrument on a corporate network.

The instrument IP address has leading zeros, but the computer IP address cannot.

To set up manual IP address selection on the instrument:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. For TCP/IP Mode, select **Manual**.
5. Enter the **IP Address**.
6. Enter the **Gateway** address.
7. Enter the **Subnet** mask.
8. Select **Apply Settings** to save your settings.

Set up LAN communications on the computer

This section describes how to set up the LAN communications on your computer.

NOTE

Do not change your IP address without consulting your system administrator. If you enter an incorrect IP address, it can prevent your computer from connecting to your corporate network or it may cause interference with another networked computer.

Record all network configurations before modifying any existing network configuration information on the network interface card. Once the network configuration settings are updated, the previous information is lost. This may cause a problem reconnecting the host computer to a corporate network, particularly if DHCP is disabled.

Be sure to return all settings to their original configuration before reconnecting the host computer to a corporate network. Contact your system administrator for more information.

Verify the LAN connection on the 2470

Make sure that your 2470 is connected to the network by confirming that your instrument was assigned an IP address.

To verify the LAN connection:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.

A green LAN status indicator on the lower left of the LAN tab confirms that your instrument was assigned an IP address.

In addition, the green LAN LED on the upper right of the front panel is on when your instrument is connected to the network.

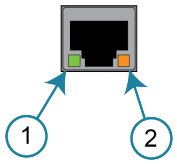
Use the LXI Discovery Tool

To find the IP address of the 2470, use the LXI Discovery Tool, a utility that is available from the Resources tab of the LXI Consortium website (lxistandard.org).

LAN status LEDs

The figure below illustrates the two status light emitting diodes (LED) that are on the LAN port of the instrument. The table below the figure provides explanations of the LED states.

Figure 17: LAN status LEDs



1	When lit, indicates that the LAN port is connected to a 100 Mbps network
2	When blinking, indicates that the port is receiving or sending information

If neither LED is lit, the network is not connected.

LAN interface protocols

You can use one of following LAN protocols to communicate with the 2470:

- Telnet
- VXI-11
- Raw socket

You can also use a dead socket termination port to troubleshoot communication problems.

NOTE

You can only use one remote interface at a time. Although multiple ethernet connections to the instrument can be opened, only one can be used to control the instrument at a time.

The port numbers for the LAN protocols and dead socket termination are listed in the following table.

LAN protocols

Port number	Protocol
23	Telnet
1024	VXI-11
5025	Raw socket
5030	Dead socket termination

Raw socket connection

All Keithley instruments that have LAN connections support raw socket communication. This means that you can connect to the TCP/IP port on the instrument and send and receive commands. A programmer can easily communicate with the instrument using the Winsock API on computers with the Microsoft® Windows® operating system or using the Berkeley Sockets API on Linux® or Apple® computers.

VXI-11 connection

This remote interface is similar to GPIB and supports message boundaries, serial poll, and service requests (SRQs). A VXI-11 driver or NI-VISA™ software is required. Test Script Builder (TSB) uses NI-VISA and can be used with the VXI-11 interface. You can expect a slower connection with this protocol.

Telnet connection

The Telnet protocol is similar to raw socket and can be used when you need to interact directly with the instrument. Telnet is often used for debugging and troubleshooting. You will need a separate Telnet program to use this protocol.

The 2470 supports the Telnet protocol, which you can use over a TCP/IP connection to send commands to the instrument. You can use a Telnet connection to interact with scripts or send real-time commands.

Dead socket connection

The dead socket termination (DST) port is used to terminate all existing ethernet connections. A dead socket is a socket that is held open by the instrument because it has not been properly closed. This most often happens when the host computer is turned off or restarted without first closing the socket. This port cannot be used for command and control functions.

Use the dead socket termination port to manually disconnect a dead session on any open socket. All existing ethernet connections will be terminated and closed when the connection to the dead socket termination port is closed.

Reset LAN settings

You can reset the password and the LAN settings from the rear panel by inserting a straightened paper clip into the hole below LAN RESET.

LAN troubleshooting suggestions

If you are unable to connect to the web interface of the instrument, check the following items:

- The network cable is in the LAN port on the rear panel of the instrument, not one of the TSP-Link® ports.
- The network cable is in the correct port on the computer. The LAN port of a laptop may be disabled when the laptop is in a docking station.
- The setup procedure used the configuration information for the correct ethernet card.
- The network card of the computer is enabled.
- The IP address of the instrument is compatible with the IP address on the computer.
- The subnet mask address of the instrument is the same as the subnet mask address of the computer.

You can also try restarting the computer and the instrument.

To restart the instrument:

1. Turn the power to the instrument off, and then on.
2. Wait at least 60 seconds for the network configuration to be completed.

To set up LAN communications:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. Verify the settings.

If the above actions do not correct the problem, contact your system administrator.

USB communications

To use the rear-panel USB port, you must have the Virtual Instrument Software Architecture (VISA) layer on the host computer. See How to install the Keithley I/O Layer (on page 2-33) for more information.

VISA contains a USB-class driver for the USB Test and Measurement Class (USBTMC) protocol that, once installed, allows the Microsoft® Windows® operating system to recognize the instrument.

When you connect a USB device that implements the USBTMC or USBTMC-USB488 protocol to the computer, the VISA driver automatically detects the device. Note that the VISA driver only automatically recognizes USBTMC and USBTMC-USB488 devices. It does not recognize other USB devices, such as printers, scanners, and storage devices.

In this section, "USB instruments" refers to devices that implement the USBTMC or USBTMC-USB488 protocol.

Using USB

To communicate from a computer to the instrument, you need a USB cable with a USB Type B connector end and a USB Type A connector end. You need a separate USB cable for each instrument you plan to connect to the computer at the same time using the USB interface.

To connect an instrument to a computer using USB:

1. Connect the Type A end of the cable to the computer.
2. Connect the Type B end of the cable to the instrument.
3. Turn on the instrument power. When the computer detects the new USB connection, the Found New Hardware Wizard starts.
4. If the "Can Windows connect to Windows Update to search for software?" dialog box opens, select **No**, and then select **Next**.
5. On the "USB Test and Measurement device" dialog box, select **Next**, and then select **Finish**.

Communicate with the instrument

For the instrument to communicate with the USB device, you must use NI-VISA™. VISA requires a resource string in the following format to connect to the correct USB instrument:

```
USB0::0x05e6::0x2470::[serial number]::INSTR
```

Where:

- `0x05e6`: The Keithley vendor ID
- `0x2470`: The instrument model number
- `[serial number]`: The serial number of the instrument (the serial number is also on the rear panel)
- `INSTR`: Use the USBTMC protocol

The resource string is displayed on the bottom right of the System Communications screen. Select **Menu**, then **Communication** to open the System Communications menu and select the **USB** tab.

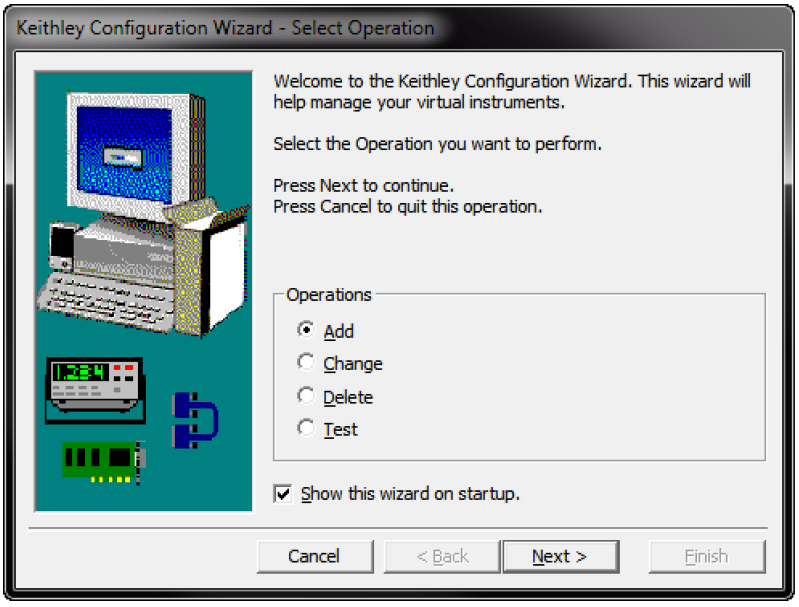
You can also retrieve the resource string by running the Keithley Configuration Panel, which automatically detects all instruments connected to the computer.

If you installed the Keithley I/O Layer, you can access the Keithley Configuration Panel through the Microsoft® Windows® Start menu.

To use the Keithley Configuration Panel to determine the VISA resource string:

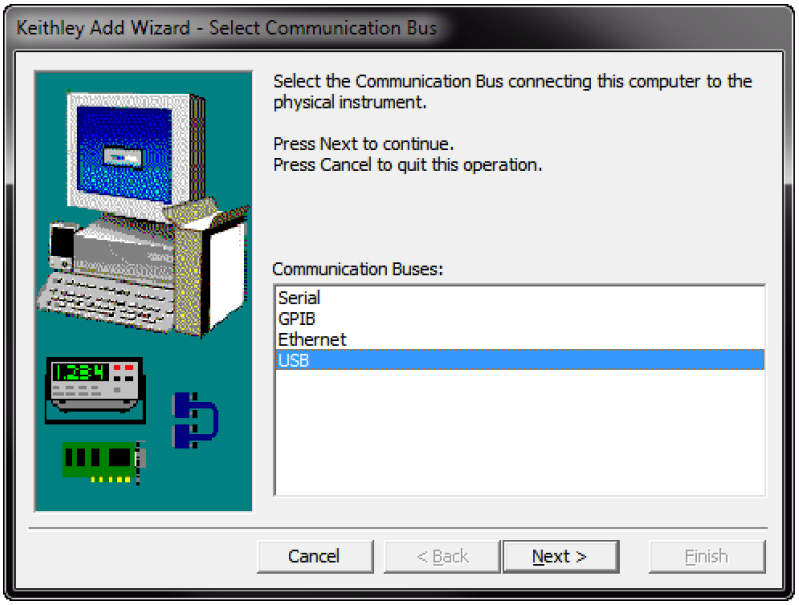
1. Select **Start > Keithley Instruments > Keithley Configuration Panel**. The Select Operation dialog box is displayed.

Figure 18: Select Operation dialog box



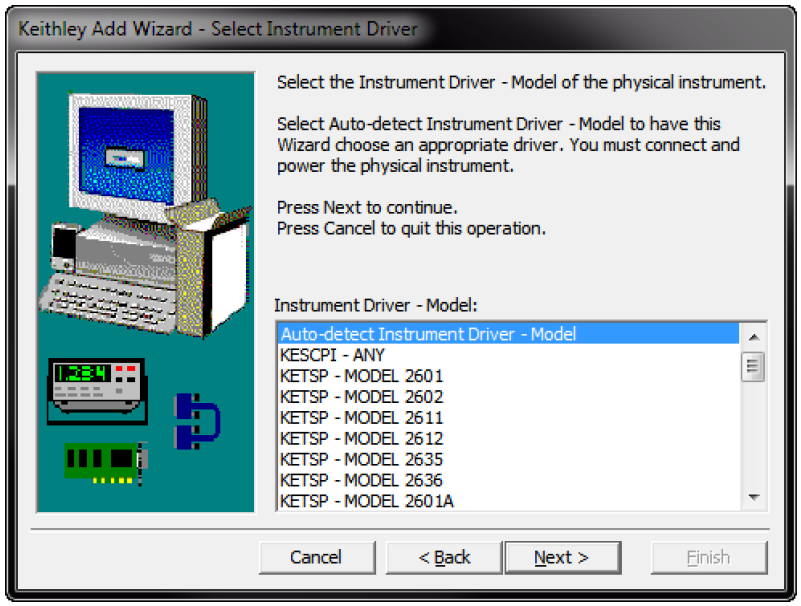
2. Select **Add**.
3. Select **Next**. The Select Communication Bus dialog box is displayed.

Figure 19: Select Communication Bus dialog box



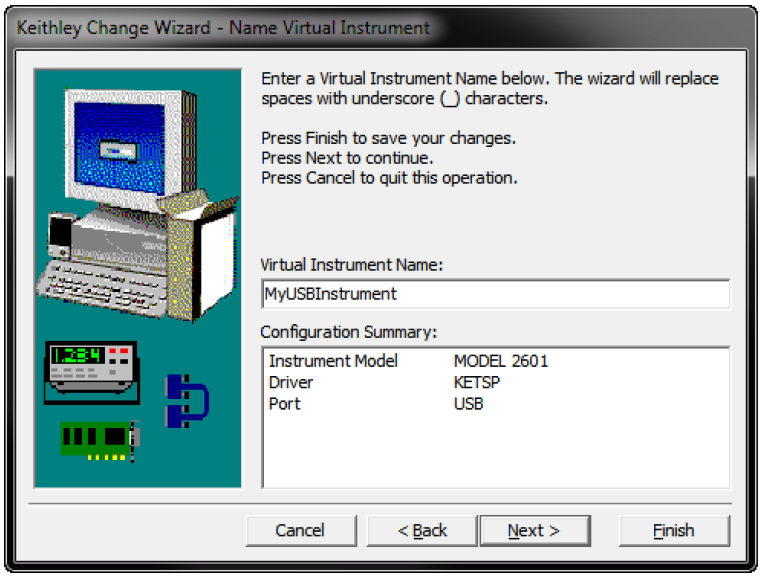
4. Select **USB**.
5. Select **Next**. The Select Instrument Driver dialog box is displayed.

Figure 20: Select Instrument Driver dialog box



6. Select **Auto-detect Instrument Driver - Model**.
7. Select **Next**. The Configure USB Instrument dialog box is displayed with the detected instrument VISA resource string visible.
8. Select **Next**. The Name Virtual Instrument dialog box is displayed.

Figure 21: Name Virtual Instrument dialog box

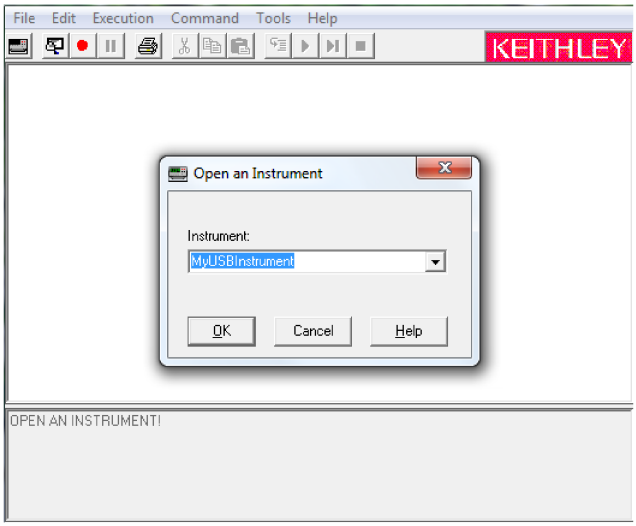


9. In the Virtual Instrument Name box, enter a name that you want to use to refer to the instrument.
10. Select **Finish**.
11. Select **Cancel** to close the Wizard.
12. Save the configuration. From the Keithley Configuration Panel, select **File > Save**.

Verify the instrument through the Keithley Communicator:

1. Set the instrument to use the SCPI command set. Refer to How do I change the command set? (on page 17-5) for instruction.
2. Select **Start > Keithley Instruments > Keithley Communicator**.
3. Select **File > Open Instrument** to open the instrument you just named.

Figure 22: Keithley Communicator Open an Instrument



4. Select **OK**.
5. Send a command to the instrument and see if it responds.

NOTE

If you have a full version of NI-VISA on your system, you can run NI-MAX or the VISA Interactive Control utility. See the National Instruments documentation for information.

2470 web interface

The 2470 web interface allows you to make settings and control your instrument through a web page. The web page includes:

- Instrument status.
- The instrument model, serial number, firmware revision, and the last LXI message.
- An ID button to help you locate the instrument.
- A virtual front panel and command interface that you can use to control the instrument.
- Ability to download data from specific reading buffers into a CSV file.
- Administrative options and LXI information.

The instrument web page resides in the firmware of the instrument. Changes you make through the web interface are immediately made in the instrument.

When the LAN and instrument establish a connection, you can open the web page for the instrument.

To access the web interface:

1. Open a web browser on the host computer.
2. Enter the IP address of the instrument in the address box of the web browser. For example, if the instrument IP address is 192.168.1.101, enter `192.168.1.101` in the browser address box.
3. Press **Enter** on the computer keyboard to open the instrument web page.
4. If prompted, enter a user name and password. The default is `admin` for both.

NOTE

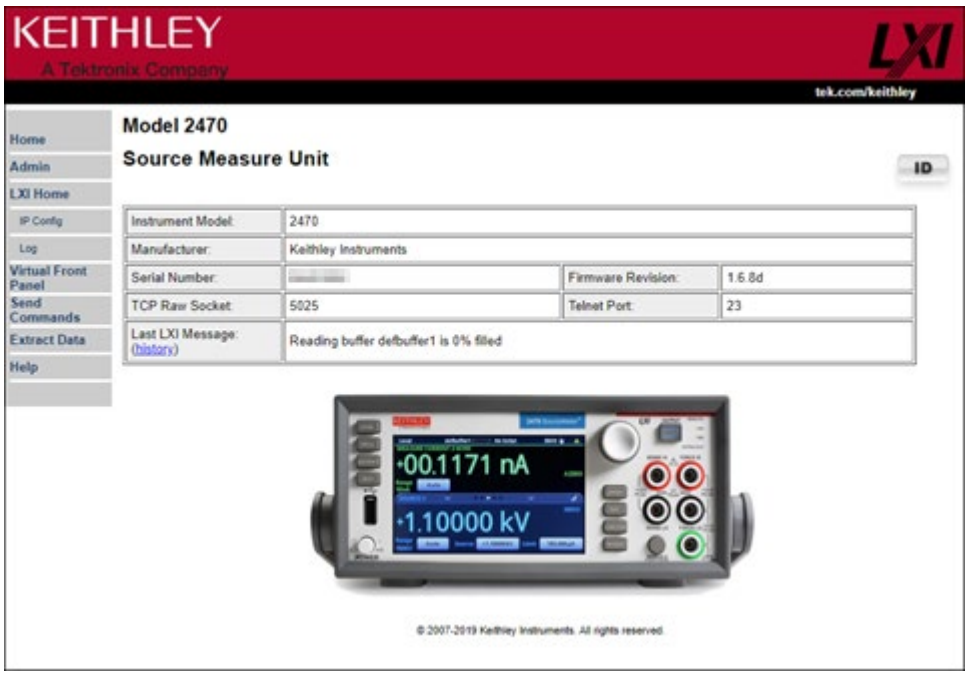
If the web page does not open in the browser, see LAN troubleshooting suggestions (on page 2-22).

NOTE

To find the IP Address of the instrument, press the Communications indicator in the upper left corner of the home screen.

Web interface Home page

Figure 23: 2470 web interface Home page



The Home page of the instrument provides information about the instrument. It includes:

- The instrument model number, manufacturer, serial number, and firmware revision number.
- The TCP Raw Socket number and Telnet Port number.
- The last LXI message. The history link opens the LXI Home page (on page 2-29).
- The ID button, which allows you to identify the instrument. Refer to Identify the instrument (on page 2-28).

Identify the instrument

If you have a bank of instruments, you can select the ID button to determine which one you are communicating with.

To identify the instrument:

1. On the Home page, select the **ID** button. The button turns green and the LAN status indicator on the front panel of the instrument blinks. On instruments with a front-panel interface, the System Communications menu also opens and the LXI LAN indicator on the LAN tab blinks.
2. Select the **ID** button again to return the button to its original color and return the LAN status indicators to steady on.

LXI Home page

The LXI Home page displays instrument information, including the host name, MAC address, and VISA resource string. You cannot change the information from this page.

You can use the host name instead of the IP address to connect to the instrument.

It also includes the ID button, which you can use to identify the instrument. See Identify the instrument (on page 2-28).

Change the IP configuration through the web interface

You can change the LAN settings, such as IP address, subnet mask, gateway, and DNS address, through the web page of the instrument.

If you change the IP address through the web page, the web page tries to redirect to the IP address that is configured in the instrument. In some cases, this may fail. This generally happens if you switch from IP address assignment that uses a static address to IP address assignment that uses a DHCP server. If this happens, you need to revert to either using the front panel to set the IP address or use an automatic discovery tool to determine the new IP address.

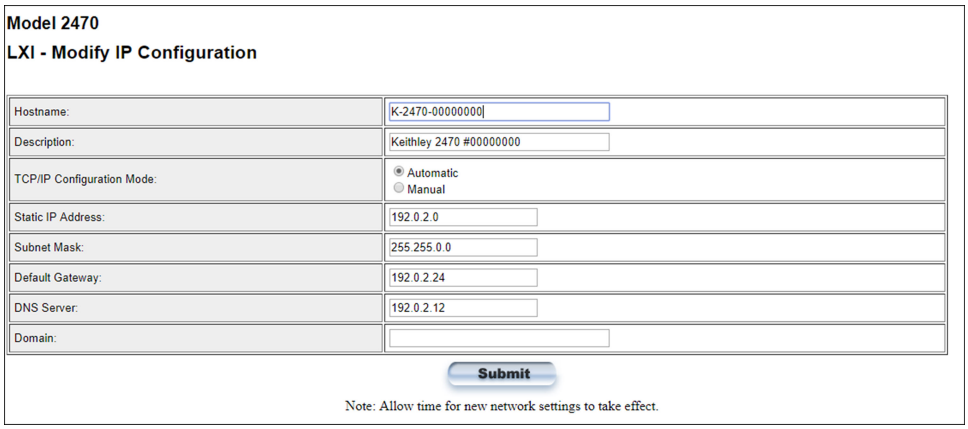
NOTE

You can also change the IP configuration through the front panel or with TSP and SCPI commands. See Set up LAN communications on the instrument (on page 2-17) for information.

To change the IP configuration using the instrument web page:

1. Access the internal web page as described in Connecting to the instrument through the web interface (on page 2-27).
2. From the navigation bar on the left, in the LXI Home menu, select **IP Config**.
3. Select **Modify**. The Modify IP Configuration page is displayed.

Figure 24: LXI - Modify IP Configuration page



4. Change the values.
5. Select **Submit**. The instrument reconfigures its settings, which may take a few moments.

NOTE

You may lose your connection with the web interface after selecting **Submit**. This is normal and does not indicate an error or failure of the operation. If this occurs, find the correct IP address and reopen the web page of the instrument to continue.

Review events in the event log

Under LXI Home, the Log option opens the event log. The event log records all LXI events that the instrument generates and receives. The log includes the following information:

- The EventID column, which shows the identifier of the event that generated the event message.
- The System Timestamp column, which displays the seconds and nanoseconds when the event occurred.
- The Data column, which displays the text of the event message.

To clear the event log and update the information on the screen, select the **Refresh** button.

Using the 2470 virtual front panel

The Virtual Front Panel page allows you to control the instrument from a computer as if you were using the front panel. You can operate the instrument using a mouse to select options.

The virtual front panel operates the same way as the actual front panel, with the following exceptions:

- The navigation control cannot be turned.
- You cannot switch the instrument on or off with the power switch.
- To scroll up or down on a screen, hold the left mouse button down and swipe up or down.
- To scroll right or left, hold the left mouse button down and swipe left or right. You can also select the dots on the bar above the swipe screens to move from screen to screen.
- You cannot use pinch and zoom on the graph screen.
- You can improve communication speed with the instrument by right-clicking and clearing **High resolution**. The default screen display resolution of 800 x 480 is reduced to 400 x 240 resolution when high resolution is cleared.
- You can display the instrument display only with no other front-panel options by right-clicking and selecting **Screen only**.
- You can download a screen capture by right-clicking and selecting **Download screenshot.**

To use the virtual front panel, you can use any of the standard web browsers. If you are using Microsoft Internet Explorer, it must be version 9 or later. Earlier versions will not allow the swipe motion to work.

NOTE

Using graphing through the virtual front panel requires significant system resources and may slow instrument operation.

NOTE

The 2470 allows fewer than three clients to open the virtual front panel web page at the same time. Only the first successfully connected client can operate the instrument. Other clients can view the virtual front panel.

For information on the options, see Screen descriptions (on page 3-9).

See the following figure for an example of the virtual front panel.

Figure 25: Virtual front panel



Change the date and time through the web interface

You can change the instrument date and time through the web interface. This is the same as changing the date and time through the front-panel System Settings menu. The date and time is used for the event log entries and data timestamps.

To change the date and time:

1. From the web interface page, select **Admin**.
2. In the Local time table, change the information as needed.
3. Select **Submit**.

Change the password through the web interface

You can change the instrument password from the web interface.

The default user name and password is `admin`. Note that you cannot change the user name; it remains at `admin` even if the password has changed.

To change the password:

1. From the web interface Home page, select **Admin**.
2. In the **Current password** box, enter the presently used password.
3. In the **New password** and **Confirm new password** boxes, enter the new password.
4. Select **Submit**.

Send commands using the web interface

You can send individual commands using the web interface.

The active command set is listed above the Command box.

To send commands using the web page:

1. From the navigation bar on the left, select **Send Commands**.
2. If requested, log in.
3. In the **Command** box, enter the command.
4. Select **Send Command** to send the command to the instrument. The command is displayed in the Command Output box. If there is a response to the command, it is displayed after the command.
5. To view any events that have occurred, select **Return Error**.
6. To clear the Command Output list, select **Clear Output**.

Extract buffer data using the web interface

The Extract Data page of the web interface allows you to download reading buffer data from the instrument.

To download buffer data:

1. From the web interface page, select **Extract Data**.
2. In the CSV File column, select the name of the file that you want to download.
3. Follow the instructions for your browser to open the file. Typically, the file opens in Microsoft Excel.

How to install the Keithley I/O Layer

NOTE

Before installing, it is a good practice to check the Product Support web page (tek.com/product-support) to see if a later version of the Keithley I/O Layer is available. Search for `Keithley I/O Layer`.

You can download the Keithley I/O Layer from the Keithley website.

The software installs the following components:

- Microsoft® .NET Framework
- NI™ IVI Compliance Package
- NI-VISA™ Run-Time Engine
- Keithley SCPI-based Instrument IVI-C driver
- Keithley I/O Layer

To install the Keithley I/O Layer from the Keithley website:

1. Download the Keithley I/O Layer Software from the Product Support web page (tek.com/product-support), as described above. The software is a single compressed file and should be downloaded to a temporary directory.
2. Run the downloaded file from the temporary directory.
3. Follow the instructions on the screen to install the software.
4. Reboot your computer to complete the installation.

Modifying, repairing, or removing Keithley I/O Layer software

The Keithley I/O Layer interconnects many other installers.

To remove all the KIOL components, you need to uninstall the following applications using Control Panel Add/Remove programs:

- National Instruments NI™ IVI Compliance Package
- National Instruments NI-VISA™ Run-Time Engine
- IVI Shared Components
- Visa Shared Components
- Keithley SCPI Driver

After uninstalling components, reboot the computer.

Interface access

You can specify that the control interfaces request access before taking control of the instrument. There are several modes of access.

You can set one of the following levels of access to the instrument:

- **Full**: Allows full access for all users from all interfaces
- **Exclusive**: Allows access by one remote interface at a time with logins required from other interfaces
- **Protected**: Allows access by one remote interface at a time with passwords required on all interfaces
- **Lockout**: Allows access by one interface (including the front panel) at a time with passwords required on all interfaces

NOTE

The front panel is read-only when you are using a remote interface. You can view information and swipe screens without being prompted to leave remote mode. If you attempt to make a change from the front panel while the instrument is controlled from a remote interface, you will be prompted to enter a password to gain access.

When you set access to full, the instrument accepts commands from any interface with no passwords required. You can change interfaces as needed.

When you set access to exclusive, you must log out of one remote interface before you can log in with another interface. You do not need a password with this access.

Protected access is similar to exclusive access, except that you must enter a password when logging in.

When you set access to locked out, a password is required to change interfaces, including the front-panel interface.

Changing the interface access type

To change the type of interface access from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**. The SYSTEM SETTINGS menu opens.
3. Select **Interface Access**.
4. Select the level of password access control you want to enable.

Using SCPI commands

Send the command that is appropriate for the level of access you want to enable:

```
SYSTem:ACCess FULL
SYSTem:ACCess EXCLusive
SYSTem:ACCess PROTected
SYSTem:ACCess LOCKout
```

Using TSP commands

Send the command that is appropriate for the level of access you want to enable:

```
localnode.access = localnode.ACCESS_FULL
localnode.access = localnode.ACCESS_EXCLUSIVE
localnode.access = localnode.ACCESS_PROTECTED
localnode.access = localnode.ACCESS_LOCKOUT
```

Changing the password

If interface access is set to Protected or Lockout, you must enter a password to change to a new control interface. You can set the password, as described below.

The default password is `admin`.

To change the password from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select **Password**. A keypad opens.
4. Enter the new password.
5. Select the **OK** button on the displayed keyboard. A verification screen is displayed.
6. Enter the new password.
7. Select the **OK** button on the displayed keyboard. The password is reset.

NOTE

You can reset the password by pressing the **MENU** key, selecting **Info/Manage** (under System), and selecting **Password Reset**. When you do this, the password returns to the default setting.

To change the password using SCPI commands:

```
:SYSTem:PASSword:NEW "<password>"
```

Where <password> is the new password.

To change the password using TSP commands:

```
localnode.password = "password"
```

Where *password* is the new password.

Switching control interfaces

When the interface access is set to anything other than Full, you need to log in to the instrument from the new interface before you can change any settings.

If you are changing to the front panel, when you attempt to make a selection, the Display Lockout - Enter Password keypad is displayed. Enter the password and select the **OK** button on the displayed keyboard.

When you change the remote interface, you must send the following TSP or SCPI command before sending commands:

```
login password
```

Replace *password* with the instrument password.

Determining the command set you will use

You can change the command set that you use with the 2470. The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

NOTE

As delivered from Keithley Instruments, the 2470 is set to work with the SCPI command set.

To set the command set from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the appropriate **Command Set**.

You are prompted to confirm the change to the command set and reboot.

To verify which command set is selected from a remote interface, send the command:

```
*LANG?
```

To change to the SCPI command set from a remote interface, send the command:

```
*LANG SCPI
```

Reboot the instrument.

To change to the TSP command set from a remote interface, send the command:

```
*LANG TSP
```

Reboot the instrument.

System information

You can get the serial number, firmware build, detected line frequency, calibration verify date, calibration adjust date, and calibration adjust count information from the instrument.

To view the version and serial number information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Info/Manage**.

The firmware version and serial number are displayed at the top of the screen.

To view the calibration information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Calibration**.

The adjust date, adjust count, and calibration date are displayed.

To view the line frequency information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Scroll down to display the line frequency.

To view system information using SCPI commands:

To retrieve the manufacturer, model number, serial number, and firmware version, send the command:

```
*IDN?
```

To read the line frequency, send the command:

```
SYStem:LFRequency?
```

The firmware build, memory available, and factory calibration date are not available when using SCPI commands.

To view system information using TSP commands:

To read the model number, send the command:

```
print(localnode.model)
```

To read the serial number, send the command:

```
print(localnode.serialno)
```

To read the firmware version, send the command:

```
print(localnode.version)
```

To read the line frequency, send the command:

```
print(localnode.linefreq)
```

The factory calibration date is not available with TSP commands.

You can also create user-defined strings to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location. See the TSP command reference (on page 14-1) for detail about the `userstring` functions.

Section 3

Instrument description

In this section:

Front-panel overview 3-1
Rear-panel overview 3-3
Turn the 2470 output on or off 3-4
Touchscreen display 3-5
Screen descriptions 3-9
Menu overview 3-18
APPS Manager 3-39
Display features 3-40
Instrument sounds 3-44
Saving setups 3-45
Resets 3-49
Using the event log 3-50
Saving front-panel settings into a macro script 3-52

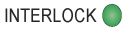
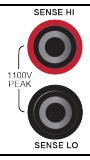
Front-panel overview

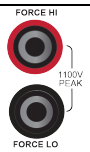
The front panel of the 2470 is shown below. Descriptions of the controls on the front panel follow the figure.

Figure 26: 2470 front panel



POWER switch	POWER	Turns the instrument on or off. To turn the instrument on, press the power switch so that it is in the on position (|). To turn it off, press the power switch so that it is in the off position (O).
HOME key	HOME	Returns the display to the home screen.
MENU key	MENU	Opens the main menu. Press the icons on the main menu to open source, measure, views, trigger, scripts, and system screens. For details, refer to Menu overview (on page 3-18).

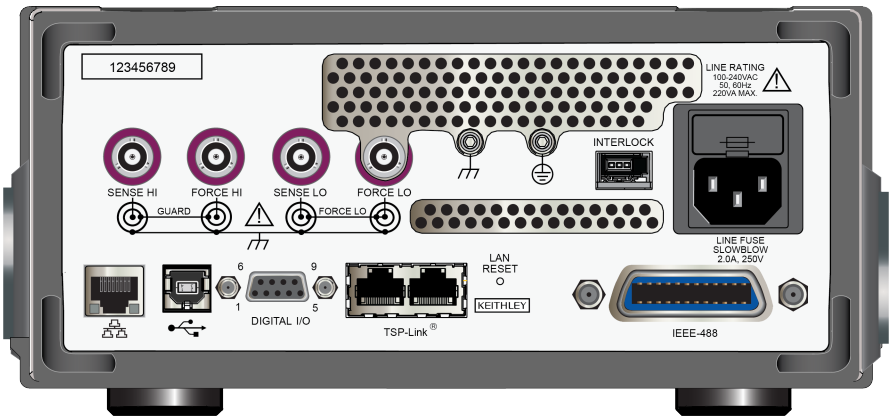
QUICKSET key		Opens a menu of preconfigured setups, including voltmeter, ammeter, ohmmeter, and power supply. Also allows you to choose source and measure functions and adjust performance for better resolution or speed. For details, see QuickSet menu (on page 3-18).
HELP key		Opens help for the area or item that is selected on the display. If there is no selection when you press the **HELP** key, it displays overview information for the screen you are viewing. To display help, hold the on-screen button while pressing the **HELP** key.
USB port		Saves reading buffer data and screen snapshots to a USB flash drive. You can also store and retrieve scripts to and from a USB flash drive. The flash drive must be formatted as a FAT or FAT32 drive.
Touchscreen		The 2470 has a high-resolution, five-inch color touchscreen display. The touchscreen accesses swipe screens and menu options. You can access additional screens by pressing the front-panel **MENU**, **QUICKSET**, and **FUNCTION** keys. Refer to Touchscreen display (on page 3-5) for details.
Navigation control		Moves the cursor and makes screen selections. **Turning the navigation control:** Moves the cursor to highlight a list value or menu item so that you can select it. Turning the control when the cursor is in a value entry field increases or decreases the value in the field. **Pressing the navigation control:** Selects the highlighted choice or allows you to edit the selected field.
ENTER key		Selects the highlighted choice or allows you to edit the selected field.
EXIT key		Returns to the previous screen or closes a dialog box. For example, press the **EXIT** key when the main menu is displayed to return to the home screen. When you are viewing a subscreen (for example, the Event Log screen), press the **EXIT** key to return to the main menu screen.
FUNCTION key		Displays instrument functions. To select a function, touch the function name on the screen.
TRIGGER key		Accesses trigger-related settings and operations. The action of the **TRIGGER** key depends on the instrument state. For details, see Switching between measurement methods (on page 8-2).
OUTPUT ON/OFF switch		Turns the output source on or off. The switch illuminates when the source output is on.
REMOTE LED indicator		Illuminates when the instrument is controlled through a remote interface.
LAN LED indicator		Illuminates when the instrument is connected to a local area network (LAN).
1588 LED indicator		1588 functionality is not supported at this time.
INTERLOCK LED indicator		Illuminates when the interlock is enabled.
Sense terminals		Use the SENSE HI and SENSE LO terminal connections to measure voltage at the device under test (DUT). When you use sense leads, measurement of the voltage drop across the force leads is eliminated. This produces more accurate voltage sourcing and measurement at the DUT.

Force terminals		Use FORCE HI and FORCE LO terminal connections to source or sink voltage or current to or from a device under test (DUT).
FRONT/REAR TERMINALS switch		Activates the terminals on the front or rear panel. When the front-panel terminals are active, a green “F” is visible to the left of the FRONT/REAR switch. When the rear-panel terminals are active, a yellow “R” is visible to the left of the switch.
Chassis connection		Banana jack connector that provides a chassis connection.

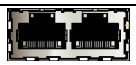
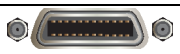
Rear-panel overview

The rear panel of the 2470 is shown below. Descriptions of the options follow the figure.

Figure 27: 2470 rear panel



SENSE and FORCE connectors		These triaxial terminals provide connections for SENSE HI and SENSE LO, FORCE HI and FORCE LO, GUARD, and chassis ground.
Chassis ground		Ground screw for connections to chassis ground. This provides a connection terminal to the equipment frame.
Protective earth (safety ground)		Ground screw for connection to protective earth (safety ground). Connect to protective earth using recommended wire size (#16 AWG or larger).
Interlock connector		Interlock connection for use with an interlock switch, such as a test fixture. When properly connected, the safety interlock of the 2470 places the outputs of the instrument in a safe state. For details, see Using the interlock (on page 4-3).
Line fuse and power receptacle		Connect the line cord to the power receptacle and a grounded AC power outlet. The line fuse, located just above the power receptacle, protects the power line input of the instrument. For safety precautions and other details, see Power the instrument on or off (on page 2-7) and Line fuse replacement (on page 10-1).
LAN port		Supports full connectivity on a 10 Mbps or 100 Mbps network. The 2470 is a version 1.5 LXI Device Specification 2016 instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet LAN). See LAN communications (on page 2-14).

USB port		USB-B connection for communication, control, and data transfer. For details, see USB communications (on page 2-22).
Digital I/O port		A digital input/output port that detects and outputs digital signals. The port provides six digital I/O lines. Each output is set high (+5 V) or low (0 V) and can read high or low logic levels. Each digital I/O line is an open-drain signal. Refer to Digital I/O (on page 8-12) for information.
TSP-Link ports		TSP-Link® system expansion interface, which builders of test systems can use to connect multiple instruments in a master and subordinate configuration. TSP-Link is a high-speed trigger synchronization and communication bus. For details, see TSP-Link System Expansion Interface (on page 9-1).
LAN reset		Reverts the LAN settings and the instrument password to default values. See Reset LAN settings (on page 2-21) for more information.
IEEE-488 port		GPIB connection; the default setting for the 2470 is 18. Refer to GPIB setup (on page 2-9) for details.

Turn the 2470 output on or off

You can turn the 2470 output on from the front panel or by sending remote commands.

⚠ WARNING

Turning the 2470 output off does not place the instrument in a safe state (an interlock is provided for this function).

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the 2470 while the instrument is powered on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the 2470 before handling cables. Putting the equipment into an output-off state does not guarantee that the outputs are powered off if a hardware or software fault occurs.

When the source of the instrument is turned off, it may not completely isolate the instrument from the external circuit. You can use the Output Off setting to place the 2470 in a known, noninteractive state during idle periods, such as when you are changing the device under test. The output-off states that can be selected for a 2470 are normal, high-impedance, zero, or guard.

See Output-off state (on page 4-16) for additional details.

Using the front panel:

Press the **OUTPUT ON/OFF** switch. The instrument is in the output-on state when the switch is illuminated. The instrument is in the output-off state when the switch is not illuminated.

Using SCPI commands:

To turn the output on, send the command:

```
:OUTPut:STATe ON
```

To turn the output off, send the command:

```
:OUTPut:STATe OFF
```

Using TSP commands:

To turn the output on, send the command:

```
smu.source.output = smu.ON
```

To turn the output off, send the command:

```
smu.source.output = smu.OFF
```

Touchscreen display

The touchscreen display gives you quick front-panel access to source and measure settings, system configuration, instrument and test status, reading buffer information, and other instrument functionality. The display has multiple swipe screens that you can access by swiping the front panel. You can access additional interactive screens by pressing the front-panel MENU, QUICKSET, and FUNCTION keys.

CAUTION

Do not use sharp metal objects, such as tweezers or screwdrivers, or pointed objects, such as pens or pencils, to touch the touchscreen. It is strongly recommended that you use only fingers to operate the instrument. Use of clean-room gloves to operate the touchscreen is supported.

Select items on the touchscreen

To select an item on the displayed screen, do one of the following:

- Touch it with your finger
- Turn the navigation control to highlight the item, and then press the navigation control to select it

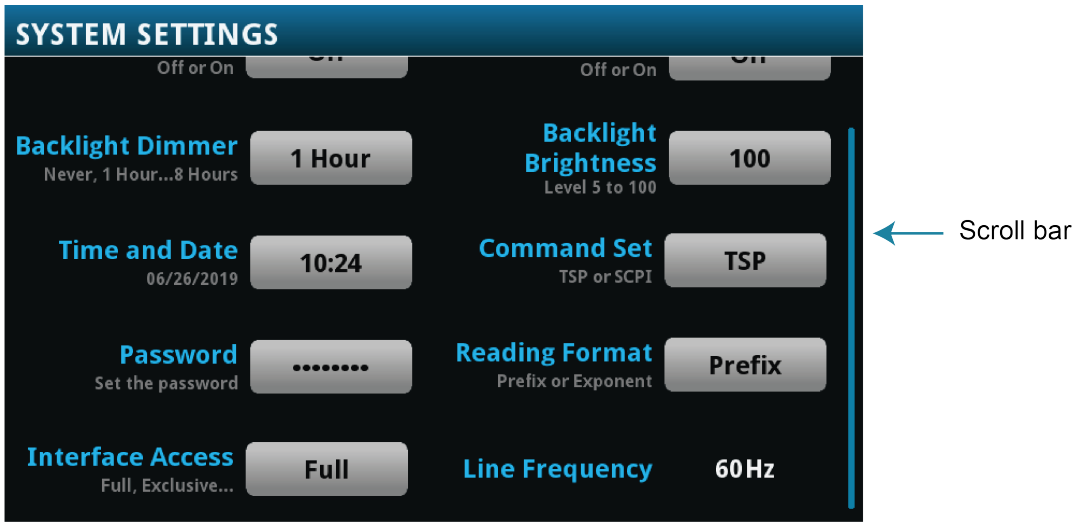
The following topics describe the 2470 touchscreen in more detail.

Scroll bars

Some of the interactive screens have additional options that are only visible when you scroll down the screen. A scroll indicator on the right side of the touchscreen identifies these screens. Swipe the screen up or down to view the additional options.

The figure below shows a screen with a scroll bar.

Figure 28: Scroll bar



Enter information

Some of the menu options open a keypad or keyboard that you can use to enter information. For example, if you are setting the name of a buffer from the front panel, you see the keyboard shown in the following figure.

Figure 29: 2470 front-panel keyboard



You can enter information by touching the screen to select characters and options from the keypad or keyboard. You can move the cursor in the entry box by touching the screen. The cursor is moved to the spot in the entry box where you touched the screen.

Some numeric keypads include Min, Max, and Inf options. Min sets the lowest value for the setting. Max sets the highest value. Inf sets the value to infinite. On number keypads, you can also use the navigation control to move the cursor to a specific number.

On keyboards and keypads, you can use the navigation control to select characters.

To set values using the number keypad with the navigation control:

1. Turn the control to underline the character that you want to change.
2. Press the control to select the character for edit.
3. Turn the control to scroll through the options.
4. Press the control to set the character.
5. Press the **ENTER** key to save the change.

Adjust the backlight brightness and dimmer

You can adjust the brightness of the 2470 touchscreen display and buttons from the front panel or over a remote interface. You can also set the backlight to dim after a specified period has passed with no front-panel activity (available from the front-panel display only). The backlight settings set through the front-panel display are saved through a reset or power cycle.

NOTE

Screen life is affected by how long the screen is on at full brightness. The higher the brightness setting and the longer the screen is bright, the shorter the screen life.

To adjust the backlight brightness from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select **Backlight Brightness**.
4. Drag the sliding adjustment to set the backlight.
5. Select **OK** to save your setting.

To set the backlight dimmer from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select **Backlight Dimmer**. The Backlight Dimmer dialog box opens.
4. Select a dimmer setting.

To adjust the brightness using the SCPI remote interface, send the following command:

```
:DISPlay:LIGHt:STATe <brightness>
```

Where `<brightness>` is one of the following options:

- Full brightness: `ON100`
- 75% brightness: `ON75`
- 50% brightness: `ON50`
- 25% brightness: `ON25`
- Display off: `OFF`
- Display, key lights, and all indicators off: `BLACkout`

To adjust the backlight using TSP commands, send the following command:

```
display.lightstate = brightness
```

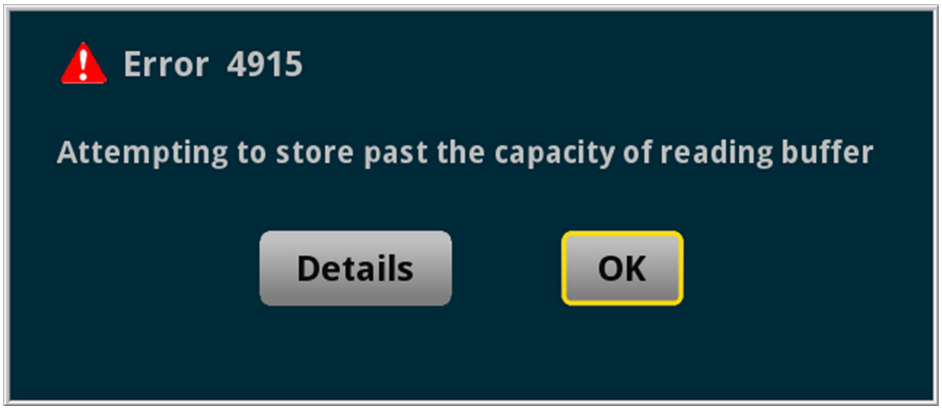
Where *`brightness`* is one of the following options:

- Full brightness: `display.STATE_LCD_100`
- 75% brightness: `display.STATE_LCD_75`
- 50% brightness: `display.STATE_LCD_50`
- 25% brightness: `display.STATE_LCD_25`
- Display off: `display.STATE_LCD_OFF`
- Display, key lights, and all indicators off: `display.STATE_BLACKOUT`

Event messages

During operation and programming, front-panel messages may be displayed. Messages are information, warning, or error notifications. For information on event messages, refer to Using the event log (on page 3-50).

Figure 30: Example front-panel error message



Screen descriptions

The following topics describe the screens and options that you can view on the 2470 front-panel display.

Home screen

This is the default screen that you see whenever you turn the 2470 on or when you press the HOME key. The options available on the home screen are described in the following topics.

Figure 31: 2470 home screen



Status and event indicators

The indicators at the top of the home screen contain information about instrument settings and states. Some of the indicators also provide access to instrument settings.

Select an indicator to get more information about the present state of the instrument. You can also select the indicators by turning the navigation control to select an indicator and then pressing ENTER.

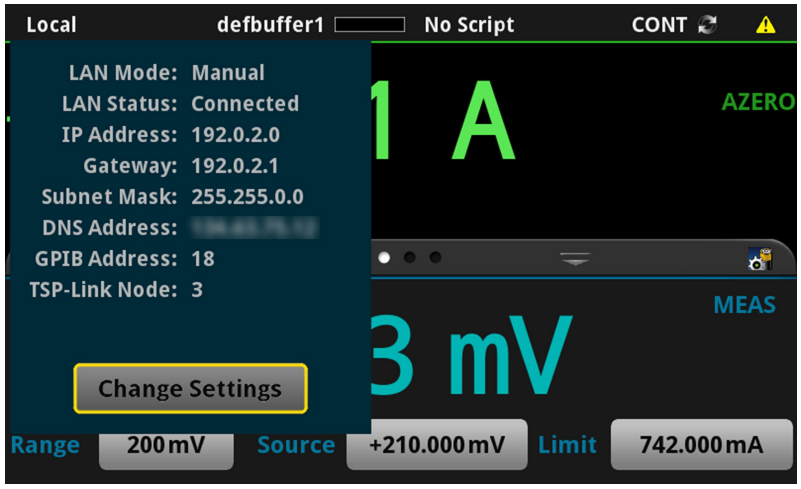
Figure 32: Home screen status bar



Communications indicator

The communications indicator displays the type of communications the instrument is using. Select the indicator to display the present communications settings. Select **Change Settings** at the bottom of the dialog box to open the System Communications screen, where you can change the settings. Refer to Remote communications interfaces (on page 2-7) for detail on the options that are available.

Figure 33: Communications indicator expanded



Indicator	Instrument communication
GPIB	Instrument is communicating through a GPIB interface
Local	Instrument is controlled from the front panel
Slave	Instrument is a subordinate in a TSP-Link system
TCPIP	Instrument is communicating through a LAN interface
Telnet	Instrument is communicating through Telnet
TSP-Link	Instrument is communicating through TSP-Link
USBTMC	Instrument is communicating through a USB interface
VXI-11	Instrument is communicating through an ethernet interface using the VXI-11 TCP/IP instrument protocol

Communications activity indicator

The activity indicator is located to the right of the communications indicator. When the instrument is communicating with a remote interface, the up and down arrows flash.

Figure 34: Communications indicator



If a service request has been generated, SRQ is displayed to the right of the up and down arrows. You can instruct the instrument to generate a service request (SRQ) when one or more events or conditions occur. This indicator stays on until the serial poll byte is read or all the conditions that caused SRQ are cleared.

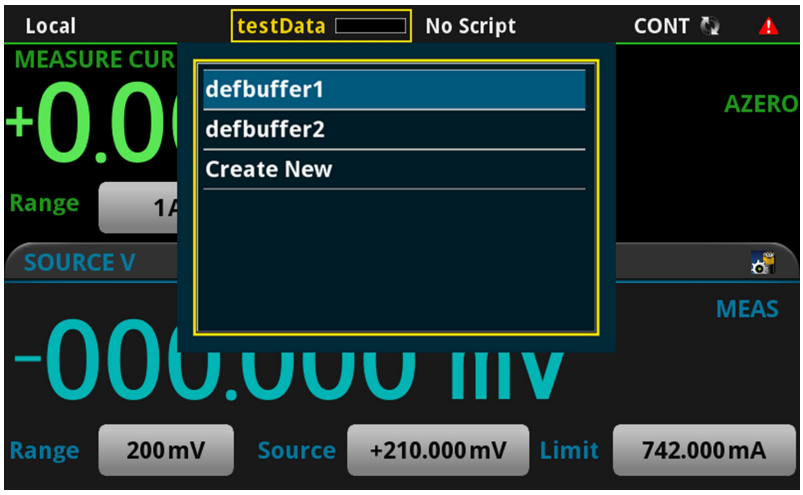
Active buffer indicator

The Active Buffer indicator shows the name of the active reading buffer. Select the indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar.

The green bar next to the buffer name indicates how full the buffer is.

To create a new buffer, select **Create New**. The new buffer is automatically assigned to be the active buffer.

Figure 35: Active buffer indicator menu



Active script indicator

This indicator shows script activity and allows you to control script action from the home screen.

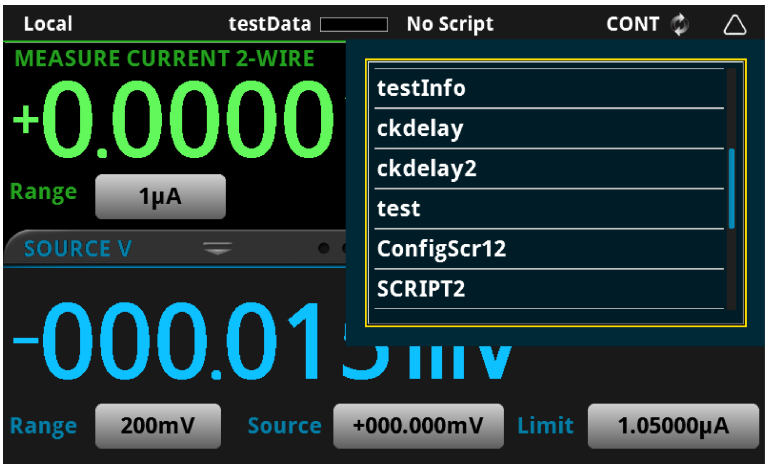
If there is no script activity, the indicator displays "No Script."

You can select the indicator to display a menu of available scripts. Select a script name to run that script. You are prompted to confirm that you want to run the script.

If a script is running from the instrument or the USB flash drive, the name of the script is displayed. If a script from TSP is running, `TSP_Script` is displayed. If you select the indicator, you are prompted to abort the running script.

If the instrument is recording a macro script, "Recording" is displayed. You can select the indicator to select an option to stop or cancel recording.

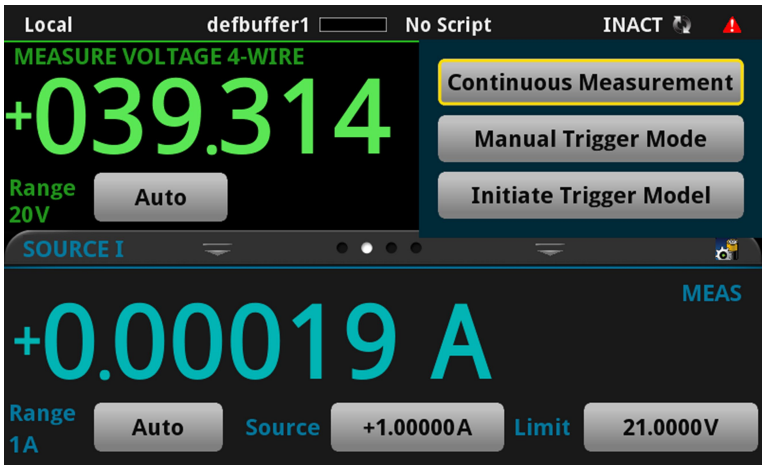
Figure 36: 2470 active script indicator



Measurement method indicator

Located to the right of the active script indicator, this indicator shows the active measurement method. Select the indicator to open a menu. Select one of the buttons on the menu to change the measurement method or initiate or abort the trigger model. In the figure below, Continuous Measurement is the present measurement method.

Figure 37: Measurement method indicator



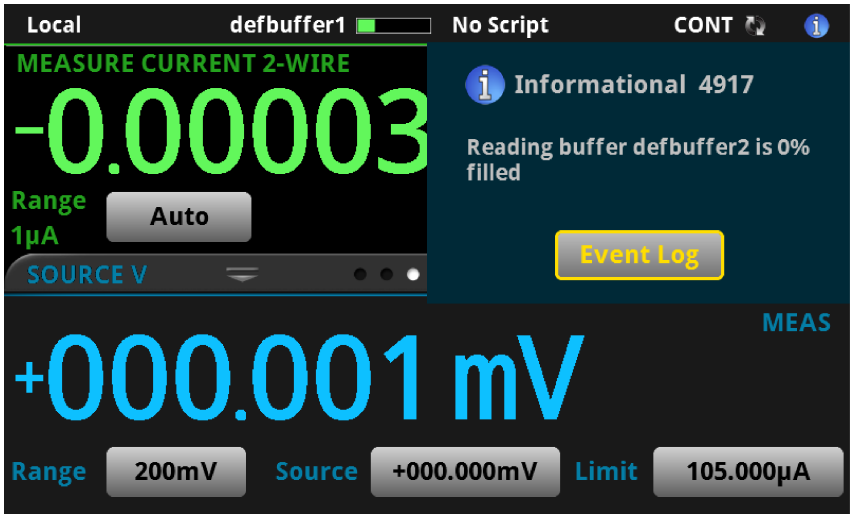
Indicator	Meaning
CONT	**Continuous measurement:** The instrument is making measurements continuously.
IDLE	Trigger model measurement method. The trigger model is not running.
INACT	The trigger model is inactive. This occurs when the trigger model cannot run, such as when the count is more than the reading buffer capacity.
MAN	**Manual trigger mode:** Press the front-panel TRIGGER key to initiate a single measurement.
RUN	Trigger model measurement method. The instrument is running the presently selected trigger model.
WAIT	Trigger model measurement method. The trigger model is waiting on an event.

System event indicator

On the right side of the instrument status indicator bar, this indicator changes based on the type of event that has been logged.

Select the indicator to open a message screen with a brief description of the error, warning, or event. Select the Event Log button to open the System Events tab of the event log, which you can use to access detailed descriptions of the events. For more information about the Event Log, see Using the event log (on page 3-50).

Figure 38: Error and message indicator



The following table describes the icons.

Icon	Description
	An empty triangle means that no new events were logged in the event log since the last time you viewed the event log.
	A blue circle means that an informational event message was logged. The message is for information only. This indicates status changes or information that may be helpful. If the Log Command option is on, it also includes commands.
	A yellow triangle means that a warning event message was logged. This message indicates that a change occurred that could affect operation.
	A red triangle means that an error event message was logged. When an error occurs, the requested change is not implemented.

Measure view area

The Measure view area of the home screen displays the value of the present measurement and other measurement information.

Figure 39: MEASURE area of the home screen



The Range button in the Measure area displays the presently selected measure range. Select the button to change the range. If the Range label is yellow, the instrument is limiting the range.

The indicators on the right edge of the Measure view area show any measure settings that affect the displayed measurement value. The indicators and what they mean are defined in the following table.

Indicator	Meaning
AZERO	Instrument automatically retrieves reference values
FILT	A filter is applied to the measurement
L1FAIL	Limit test one is enabled and measurement failed
L1PASS	Limit test one is enabled and measurement passed
L2FAIL	Limit test two is enabled and measurement failed
L2PASS	Limit test two is enabled and measurement passed
MATH	A percent, mx+b, or reciprocal calculation is applied
REL	Relative offset is applied

Swipe screens

The 2470 touchscreen display has multiple screens that you can access by swiping left or right on the lower half of the display. The options available in the swipe screens are described in the following topics.

Swipe screen heading bar

The heading bar of the swipe screen contains the following options.

Figure 40: Swipe screens, maximized and minimized



#	Screen element	Description
1	Minimize indicator 	You can swipe down to minimize the swipe screens.
2	Swipe screen indicator 	Each circle represents one swipe screen. As you swipe right or left, a different circle changes color, indicating where you are in the screen sequence. Select a circle to move the swipe screen without swiping.
3	Calculations shortcut 	Select to open the CALCULATION SETTINGS menu.
4	Measure Settings shortcut 	Select to open the MEASURE SETTINGS menu for the selected function.
5	Restore indicator 	Indicates that you can swipe up to display the swipe screen.
6	Graph shortcut 	Select to open the Graph screen.

SOURCE swipe screen

The SOURCE swipe screen shows the present value of the source and the set values for source, source range, and source limit. You can change the set values from the front panel by selecting the buttons on this screen.

Figure 41: SOURCE swipe screen



Source function indicators on the right side of the screen signify settings that affect the displayed source value:

- **MEAS:** Source readback is on and the value shown is the measured value of the source.
- **PROG:** Source readback is off and the value shown is the programmed source value. If the output is off, the displayed source value is replaced with **Output Off**.

When Limit label is shown in yellow, the instrument is limiting the source.

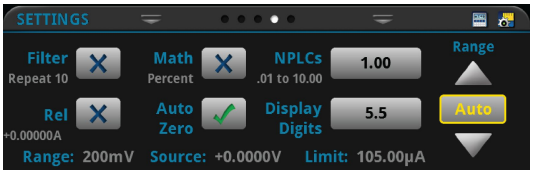
The present value is updated continuously when the measurement method is set to Continuous. When the measurement method is set to Manual Trigger Mode or Initiate Trigger Model, the value is updated when the next measurement occurs. Refer to Source limits for more information.

The icon on the right side of the swipe screen heading bar is a shortcut to the full SOURCE SETTINGS menu.

SETTINGS swipe screen

The SETTINGS swipe screen gives you front-panel access to some instrument settings. It shows you the present settings and allows you to change, enable, or disable them quickly.

Figure 42: SETTINGS swipe screen



To disable or enable a setting, select the box next to the setting so that it shows an X (disabled) or a check mark (enabled).

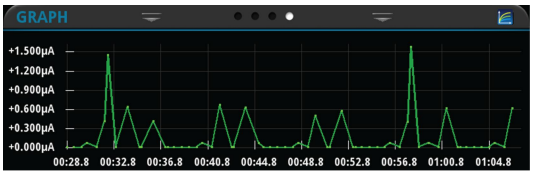
The icons on the right side of the swipe screen heading bar are shortcuts to the CALCULATIONS SETTINGS and MEASURE SETTINGS menus.

For descriptions of the settings, use the navigation control to select the button, then press the HELP key.

GRAPH swipe screen

The GRAPH swipe screen shows a graphical representation of the readings in the presently selected reading buffer.

Figure 43: GRAPH swipe screen



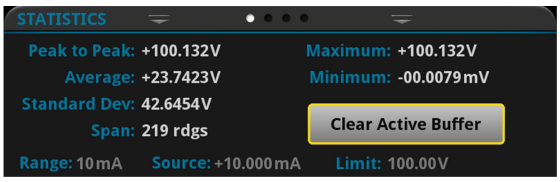
To view the graph on the full screen and to access graph settings, select the graph icon on the right side of the swipe screen header. You can also open the full-function Graph screen by pressing the **MENU** key and selecting **Graph** under Views.

For more information about graphing measurements, see Graphing (on page 7-1).

STATISTICS swipe screen

The STATISTICS swipe screen contains information about the readings in the active reading buffer. When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make. You can use the **Clear Active Buffer** button on this screen to clear the data from the active reading buffer.

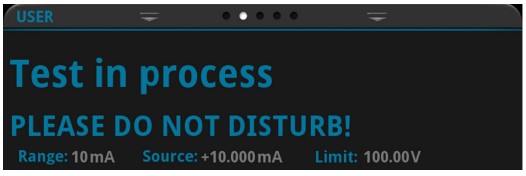
Figure 44: STATISTICS swipe screen



USER swipe screen

If you program custom text, it is displayed on the USER swipe screen. For example, you can program the 2470 to show that a test is in process. This swipe screen is only displayed if custom text has been defined. For details about using remote commands to program the display, refer to Customizing a message for the USER swipe screen (on page 3-42).

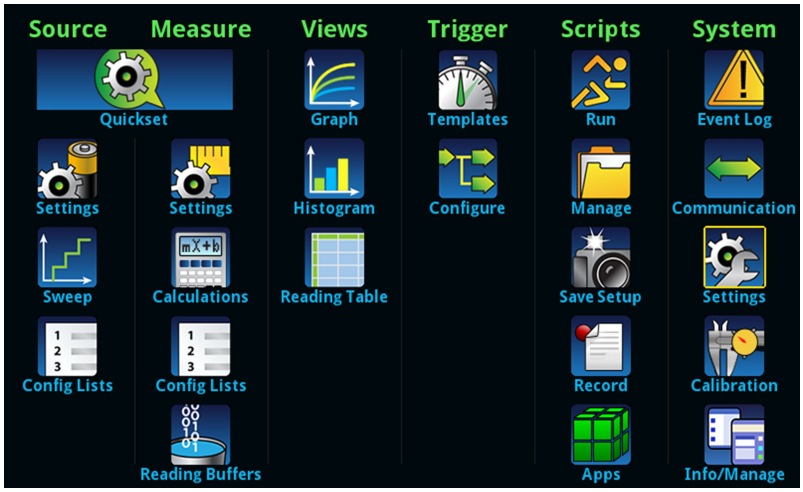
Figure 45: USER swipe screen



Menu overview

To access the main menu, press the MENU key on the 2470 front panel. The figure below shows the organization of the main menu.

Figure 46: 2470 main menu



The main menu includes submenus that are labeled in green across the top of the display. Selecting an option in a submenu opens an interactive screen.

QuickSet menu



The QuickSet menu, which is centered under Source and Measure on the main menu, allows you to:

- Select predefined setups for the source and measure functions
- Use the Performance slider to adjust for performance (resolution versus speed)
- Select Quick Setups that provide instrument-type emulation

CAUTION

When you select a Quick Setup, the instrument turns the output on. Carefully consider and configure the appropriate output-off state, source, and limits before connecting the 2470 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

When you adjust the Performance slider, the instrument adjusts settings based on where you position the slider. As you increase speed, you lower the amount of resolution. As you increase resolution, you decrease the reading speed. The settings that the instrument adjusts include autozero, autodelay and filter settings, display digits, NPLC, and source readback. These settings take effect the next time the output is turned on and measurements are made.

When you select a Quick Setup, the home screen is displayed so you can see the readings generated by the Quick Setup.

Source menu

The menus organized under Source in the main menu allow you to select, configure, and perform source and sweep operations from the front panel. The following topics describe the settings that are available on these screens.

Source Settings menu



You can change the following settings by pressing the **MENU** key and selecting Source **Settings**.

Setting	Description
Range	Set the source range for the selected source function. For more information, see Source range (on page 4-39).
Output Off	Select from **Hi Impedance**, **Normal**, **Zero**, and **Guard** output-off states. For more information, see Output-off state (on page 4-16).
Overvoltage Protection Limit	Set the overvoltage protection setting of the source output to restrict the maximum voltage level that the instrument can source. For more information, see Overvoltage protection (on page 4-35).
Interlock	Determines if the output can be turned on when the interlock is not engaged. For more information, see Using the interlock (on page 4-3).
High Capacitance	Turn on this setting to minimize overshoot, ringing, and instability when measuring low current while driving a capacitive load. For more information, see High-capacitance operation (on page 5-23).
Source Readback	Turn on this setting to have the instrument record and display the actual value of the source, instead of the programmed value. Using source readback results in more accurate measurements, at the cost of a reduction in measurement speed. Turn off this setting to increase measurement speed. For more information, see Source readback (on page 4-45).
Source Delay	Set a delay for the selected source function. This delay is in addition to normal settling times. Select **Auto Delay** to have the instrument automatically set the delay or select **Specify Delay** to enter a value. For more information, see Setting the source delay (on page 4-47).

Source Sweep menu



This menu allows you to set up a sweep and generate a source configuration list. It also builds a trigger model for the sweep.

Setting	Description
Generate	Select to create a source configuration list and trigger model using the settings on this menu.
Sweep Type	Select the type of sweep: Linear, logarithmic, linear dual, or logarithmic dual.
Start	Set the sweep voltage or current level at which the sweep starts.
Stop	Set the sweep voltage or current value at which the sweep stops.
Definition	Set this to Number of Points or Step Size. If you select Step Size, the instrument sets the number of steps based on the step size.
Step	When the Sweep Definition is set to **Step Size**, this sets the size of the steps that the instrument uses to calculate the points for the sweep.
Points	When the Sweep Definition is set to **Number of Points**, you can set the number of points for the sweep.
Source Delay	The delay time between measurement points. You can select an automatic delay or set a specific delay from 0 to 9999.999 s.
Count	Specifies the number of times to run the sweep; you can set it to run infinitely or a specific number of times.
Source Limit	Specifies the source limit value for the sweep.
Abort on Limit	Enable or disable aborting a sweep when it reaches the programmed limit.
Source Ranging	Select **Auto** to automatically set the most sensitive source range for each source level in the sweep. Select **Best Fixed** to have the instrument select a single fixed source range that will accommodate all the source levels in the sweep. Select **Fixed** to have the source remain on the range that is set when the sweep is started.

For more information about setting up sweeps, see Sweep operation (on page 4-54).

Source Config Lists menu



This menu allows you to select an existing source configuration list, create a new list, load configuration settings to and from the instrument, delete a configuration index, and view the settings of a point in a source configuration list. For more information about using configuration lists, see Configuration lists (on page 4-82).

Setting	Description
Add Settings	Saves the present instrument settings to an index at the end of the selected configuration list. If an index is selected, you are prompted to append the settings to the end of the list or overwrite the settings at the selected index.
Create New	Displayed if no source configuration lists have been created. Adds a source configuration list.
Index Details	Displays the details of the selected configuration index. Details include settings such as function, level, protection limit, source readback, and delay. You can scroll the displayed list to view additional settings.

Setting	Description
Jump to Index	Opens a number pad that you can use to select an index.
Last Index	Lists the last index in the selected configuration list.
Recall Index	Restores the instrument to the settings stored in the selected configuration list index.
Remove Index	Deletes a configuration list index from the selected configuration list.
Select	Displays a list of available source configuration lists from which you can choose.

Measure menu

The Measure menus allow you to select, configure, and perform measure operations from the front panel. The following topics describe the settings that are available on these screens.

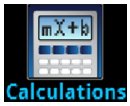
Measure Settings menu



This menu contains settings for the presently selected measure function, which is identified by an indicator in the upper right corner of the menu. Select the function indicator to change the source and measure function.

Setting	Description
Auto Range Low Limit	Set the Auto Range Low Limit to prevent the instrument from selecting a range that is lower than appropriate for your application.
Auto Range Rebound	Determines if the instrument restores the measure range to match the limit range after making a measurement. Refer to Source limits for additional information.
Auto Zero	Set Auto Zero to On to set the instrument so that it periodically gets new measurements of its internal ground and voltage reference. This setting increases measurement accuracy but may slow measurement time.
Count	Sets the number of readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-40).
NPLC	Set the amount of time that the input signal is measured. Lower NPLC settings result in faster reading rates, but increased noise. Higher NPLC settings result in lower reading noise, but slower reading rates.
Offset Compensation	The offset compensation setting is only available when the instrument is set to measure resistance (SVMR(Ω) or SIMR(Ω)). Turn this setting on to improve low-resistance measurement accuracy.
Range	Set the measure range for the presently selected measure function. You can select automatic or a specific range. You can only select a measure range if you are sourcing one type of measurement and measuring another.
Sense	Select 2-wire (local) or 4-wire (remote) sense mode. Use 2-wire sense if the error contributed by test lead IR drop is acceptable. For more accurate voltage source and measurement accuracy, use 4-wire sense.

Measure Calculations menu



The **Calculations** menu contains settings that specify the way measurement information is processed and returned.

Relative offset

Setting	Description
Rel	Use the relative offset feature to subtract a set value or a baseline reading from measurement readings. When you enable relative offset, all subsequent measurements are displayed as the difference between the actual measured value and the relative offset value.
Rel Value	Sets the relative offset value to be applied to measurements.

Filter

Setting	Description
Count	This sets the number of measurements that are averaged when filtering is enabled.
Filter	Enables or disables the averaging filter for measurements of the selected function.
Type	Selects the type of averaging filter that is used for the selected measure function when the measurement filter is enabled. Select the moving average filter to continuously add measurements to the stack on a first-in, first-out basis, replacing the oldest measurement in the stack with a new measurement. Select the repeating average filter to average a set of measurements and then flush the data out of the stack before averaging a new set of measurements.
Settings	Displays the settings that are available for the averaging filter.

Math

Math	This setting enables or disables math operations. When this is on, the math operation specified by Math Format is applied to the measurement.

Setting	Description
b(Offset)	Defines the constant for the offset factor.
m(Scalar)	Defines the constant for the scale factor.
Math Format	When Math is enabled, you can specify which math operation is performed on measurements. You can choose one of the following math operations: ■ **mx+b:** Manipulate normal display readings by adjusting the m and b factors. ■ **Percent:** Specify a constant that is applied to the measurement and display readings as percentages. ■ **Reciprocal:** The reciprocal math operation displays measurement values as reciprocals. The displayed value is 1/X, where X is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).
Settings	Displays the settings that are available for the math functions.
Zero Reference	When the Math State is set to On, this setting specifies the reference used when the math operation is set to percent; the range is –1e12 to +1e12.

Limit

Setting	Description
Limit 1 and Limit 2	These settings enable or disable limit testing. The Limit options allow you to do pass-or-fail limit testing using the front panel of the instrument. When you do a limit test, the home screen displays the pass or fail result of the test. Limit State enables or disables a limit test for the selected measurement function. When testing is enabled, limit testing occurs on each measurement. Limit testing compares the measurements to the high and low limit values. If a measurement is outside these limits, the test fails. If a measurement is in the limits, it passes.
Audible	This determines if the instrument beeper sounds when a limit test passes or fails.
Auto Clear	Specifies whether the high and low limits should be cleared automatically.
High Value	The Limit High Level specifies the upper limit for a limit test.
Low Value	The Low Value specifies the lower limit for limit tests.
Settings	Displays the settings that are available for the limit functions.

Measure Config Lists menu



The **Config Lists** menu allows you to select an existing measure configuration list, create a new list, load configuration settings to and from the instrument, and view the settings of an index in a configuration list. For more information about using configuration lists, see Configuration lists (on page 4-82).

Setting	Description
Add Settings	If no index is selected, saves the present instrument settings to an index at the end of the selected configuration list. If an index is selected, you are prompted to append the settings to the end of the list or overwrite the settings at the selected index.
Create New	This option is displayed if no measure configuration lists have been created. When select, it adds a measure configuration list.
Index Details	Displays the details of the selected configuration index. Details include settings such as function, value, delay, calculations, and range. You can scroll the displayed list to view additional settings.
Jump to Index	Opens a number pad that you can use to select an index.
Last Index	Lists the last index in the selected configuration list.
Recall Index	Restores the instrument to the settings stored in the selected configuration list index.
Remove Index	Deletes a configuration list index from the selected configuration list.
Select	Displays a list of available measure configuration lists from which you can choose.

Measure Reading Buffers menu



The **Reading Buffers** menu allows you to view the list of existing reading buffers and select one to be the active buffer. You can also create, save, delete, resize, and clear buffers from this screen.

To create a new reading buffer, select **Buffer** and select **Create New**. The new buffer is automatically set to be the active buffer.

Setting	Description
Amount Filled	The percentage of data that is presently in the buffer.
Buffer	Selects an existing buffer to configure. Includes the Create New option, which allows you to create a new buffer.
Capacity	Sets the maximum number of readings that the buffer can store. Note that when you change the capacity of a buffer, the readings in that buffer are cleared.
Clear	Clears data from the selected buffer. You can also press the MENU and EXIT keys simultaneously to clear the active buffer.
Delete	Deletes the selected buffer.
Fill Mode	**Continuous:** Fills the buffer continuously and overwrites old data when the buffer is full. **Once:** Stops collecting data when the buffer is full (no data is overwritten).
Make Active	Makes the selected buffer the active reading buffer.
Save to USB	Saves the data in the buffer to a CSV file, which can be opened by a spreadsheet program. A USB flash drive must be present in the front-panel USB port before you select Save to USB.
Style	Defines the amount and type of data the buffer stores. Only available when creating a new buffer. **Standard:** Store readings with full accuracy with formatting. **Compact:** Store readings with reduced accuracy (6.5 digits) with no formatting information, 1 µs accurate timestamp. **Full:** Store the same information as standard, plus additional information.

NOTE

The maximum readings represent the highest possible limits and may vary depending on memory usage, reading buffer style, or other reading buffers.

Views menu

The menus under Views in the main menu allow you to select, configure, and view data from measure operations on the 2470. The following topics describe the settings that are available on these screens.

Views Graph menu



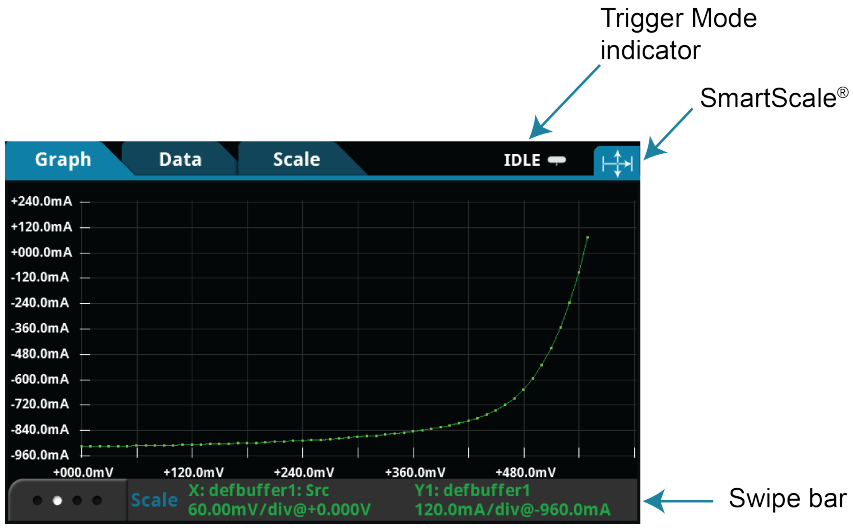
The **Graph** menu opens a screen that displays a graph of the measurements as traces in selected reading buffers. It also contains tabs that you use to customize the graph display.

You can also select the trigger mode and initiate the trigger model from this screen. Select the measurement method indicator in the upper right corner of the screen and select the measurement method. Refer to Measurement method indicator (on page 3-12) for details.

Graph tab

The Graph tab graphs readings as they are made by the instrument. Settings you make on the Data, Scale, and Trigger tabs affect how readings appear on this screen. You can also select the number of traces that are displayed.

Figure 47: Graph tab



You can pinch to zoom in the graph view. You can also swipe the graph to the left or right. When you adjust the view, the SmartScale® feature is turned off. To turn SmartScale back on, select the icon in the upper right of the Graph tab. When SmartScale is on, the instrument determines the best way to scale data based on the data and the instrument configuration (such as the measure count).

The values on the x and y axes show the values set in the Scale tab. The information at the bottom of the Graph tab contains a legend of the active axis and scale settings for the graph. You can swipe the legend to view buffer statistics and readings at the cursors. If the buffer type is set to full and contains extra values the statistics are shown as Not Applicable.

Data tab

The Data tab allows you to select the reading buffer that provides the data that is displayed on the Graph tab. You can select up to four buffers. The data from each buffer is shown as a separate trace on the Graph tab. You can also select the type of drawing style that is used on the graph.

Setting	Description
Add	Selects the reading buffers that supply the data for the traces on the Graph tab. You can select up to 20 reading buffers and specify the buffer element for the y-axis for each. Colors are automatically assigned to the traces and cannot be changed.
Clear Buffer	Clears data from the selected buffer. To clear the active buffer, you can press the MENU and EXIT keys simultaneously.
Draw Style	The drawing style determines how data is represented when there are many data points. You can select Line, Marker, or Both. When Line is selected, the data points are connected with solid lines. When Marker is selected, the individual data points are shown with no connecting lines. When both are selected, the individual data points are shown and the points are connected with solid lines.
Graph Type	Sets the data to be plotted on the x-axis. You can select Scatter/IV or Time.
Remove	Removes the trace that is selected in the Traces list. This removes the trace from the display.
Traces	Displays the names of the reading buffers that contain the data for the traces that are displayed on the Graph tab. If no buffer is selected, the active buffer is used. You can select up to 20 buffers. The data from each buffer is displayed as a separate trace on the graph. The active buffer contains the data that is displayed on the home screen and where readings are stored when Continuous Measurement is selected or a manual trigger is generated. When an active buffer is selected on the Data tab, that trace tracks the active buffer instead of a specific buffer. If the active buffer changes, the data that is displayed changes to match the new active buffer. To remove the active buffer from the list of traces, select it and select **Remove Trace**. This does not affect the active buffer that is selected on the home screen. To restore the active buffer to the list of traces, select **Add** and select the trace labeled **Active**.

Scale tab

The Scale tab contains settings that allow you to fine-tune the output on the Graph tab.

Setting	Description
Trace	When multiple traces are selected, toggles between the available traces. Information specific to the trace is shown in the same color as the trace.
X-Axis and Y-Axis Minimum Position	Sets the first value that is visible on the graph for the selected trace.
X-Axis and Y-Axis Scale	Sets the reading value scale for each division.
X-Axis Method	The method determines how data is scaled and tracked on the Graph tab. You can select: ■ **SmartScale®:** When the SmartScale feature is selected, the instrument scales the graph automatically, determining the best scaling and tracking method based on the data, reading groups, number of traces, and instrument configuration. The scale is set to show the most relevant portion of the data that is in the selected reading buffer. ■ **Show New Readings:** The graph always displays the latest data on a fixed scale. ■ **Show Group of Readings:** The graph displays a group of readings. A group is automatically created when the measure count is set to more than 1. ■ **Show All Readings:** All data in the buffer is displayed on the graph. ■ **Off:** The graph is not automatically adjusted. You can adjust the data manually by swiping, pinching, and zooming. You can also set the Scale and Minimum Position on the Scale tab.
Y-Axis Method	The scale method determines how data is scaled on the Graph tab. If you are graphing one trace, you can select: ■ **SmartScale®:** The instrument scales the graph automatically. The scale is set to fit all the data that is in the selected reading buffer onto the screen. The instrument determines the best scale based on the data. ■ **Autoscale Always:** Continuously scales the y-axis of the trace so it fits the entire height of the screen. ■ **Autoscale Once:** Scales the y-axis of the trace once. ■ **Off:** No automatic scaling. If you are graphing multiple traces, you can select: ■ **SmartScale®:** The instrument scales the graph automatically. The instrument determines the best scale and tracking method based on the data, reading groups, number of traces, and instrument configuration. ■ **Independent Autoscale:** Scales the y-axis of the trace so it fits the entire height of the screen. ■ **Swim Lanes:** Scales the y-axis of the traces in equal, non-overlapping portions of the height of the screen. ■ **Shared Autoscale:** Accommodates the minimum and the maximum of all traces. ■ **Off:** No automatic scaling.
Y-Axis Scale Format	Sets the scale format that is used on the graph. Select Linear to increase the step size in even increments. Select Log to increase the step size exponentially.

Trigger tab

The Views Graph Trigger tab contains settings that define the trigger mode.

Setting	Description
Source Event	Determines the event that is used to trigger measurements. You can select: ■ **Display TRIGGER Key:** The trigger occurs when you press the TRIGGER key. ■ **Digital Input Line:** The trigger occurs when a pulse is detected from the digital input line. ■ **TSP-Link Input:** The trigger occurs when a pulse is detected from a TSP-Link input line. ■ **Source Limit:** The trigger occurs when the source attempts to exceed the source limit. ■ **None:** No trigger event.
Delay	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay.
Position	The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled.
Trigger Clear	Specifies whether previously detected trigger events are cleared. You can select: ■ **Enter:** Previously detected trigger events are cleared. ■ **Never:** Any previously detected triggers are acted on immediately and not cleared.
Edge	When the source is set to Digital, TSP-Link, or External, this sets the type of edge that generates a trigger. You can set it to Rising, Falling, or Either.
Digital In Line	When the source is set to Digital, this selects the digital input line that generates the trigger (1 to 6).
TSP-Link Line	When the source is set to TSP-Link, this selects the TSP-Link input line that generates the trigger (1 to 3).

Views Histogram menu

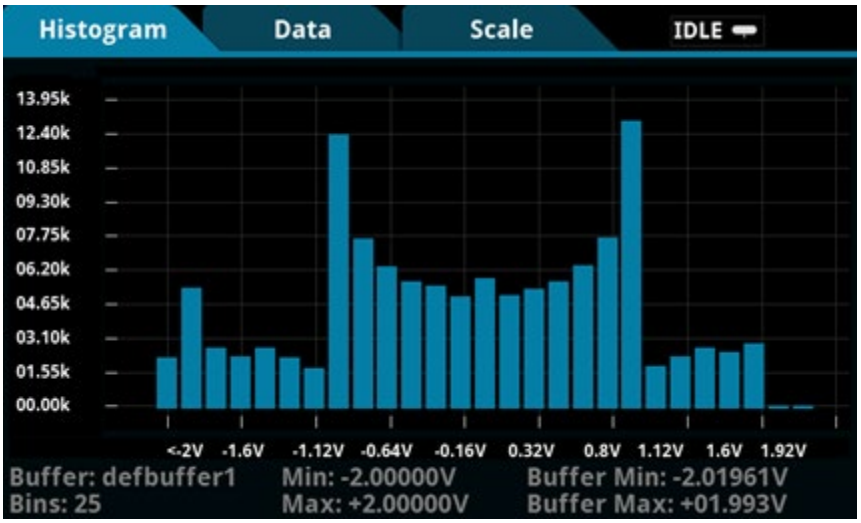


The **Histogram** menu allows you to graph the distribution of measurement data in the selected reading buffer. It also contains tabs that you use to customize the histogram.

Histogram tab

The Histogram tab graphs readings as a bar graph of the data distribution into bins. Settings you make on the Data and Scale tabs affect which data are used and how data distributions appear on this screen. You can change the scale of either axis on the screen by dragging or pinching the screen.

Figure 48: 2470 Histogram



Data tab

The Data tab allows you to select which reading buffer provides the data that is binned on the Histogram tab. You can also clear the data from the selected buffer.

Setting	Description
Add Trace	Selects the reading buffers that supply the data for the traces on the Data tab.
Clear Buffer	Clears data from the selected buffer. To clear data from the active buffer, you can press the MENU and EXIT keys.

Scale tab

The Scale tab allows you to set up boundaries, number of bins, and type of scaling used for the histogram.

Setting	Description
Maximum Boundary	The highest value of the data that is binned in the histogram. Data that is above this level is binned in the high outlier bin.
Method	The method of autoscaling to use: ▪ **SmartScale®:** Automatically select the most appropriate scaling method. ▪ **Auto Bin:** Redistribute the data evenly in the bins based on the present minimum and maximum boundaries. ▪ **Fit:** Adjust the y-axis scale so that the tops of all bins are visible ▪ **Off:** Turn off autoscaling.
Minimum Boundary	The lowest value of the data that is binned in the histogram. Data that is below this level is binned in the low outlier bin.
Number of Bins	The number of bins in the histogram. The histogram will create two outlier bins in addition to the bins you define. These bins are used to collect data that is below or above the defined minimum and maximum boundaries.

Views Reading Table menu



This menu allows you to view data in the selected reading buffer.

Setting	Description
Buffer	Selects the reading buffer that contains the data you want to view. If Active is selected, the data from the reading buffer that is presently storing readings is displayed.
Reading Details	Select a data point to open the Reading Details window for the selected data point. The details describe the instrument settings when the data point was read.
Reading Preview Graph	Shows a small graph view of the data in the reading table. Touch a data point in the graph to jump to that data point in the table.
Table	Displays the data in the selected reading buffer. You can select a data point to display additional detail about that data point.

Trigger menu

The menus under Trigger in the main menu allow you to configure triggering operations from the 2470 front panel. The following topics describe the settings that are available on these screens.

Trigger Templates menu



The **Templates** menu allows you to choose from one of several preprogrammed trigger models. When you select a template, settings you can specify for that template are shown in the lower part of the screen.

You can also customize the templates from the front panel using the Configure menu under Trigger on the main menu screen. For details, see Trigger Configure menu (on page 3-32).

The table below describes the trigger-model templates and available user-specified settings and their default settings.

Setting	Description
ConfigList	Creates a trigger model that loads a source and measure configuration list. The lists are iterated until every index in the configuration list with fewer indexes has been loaded. For example, if the measure list has seven indexes and the source configuration list has 10, it will iterate through seven indexes of the source list and all of the measure list. However, if the source list has three indexes and the measure list has five, it will iterate through three indexes of measure list and all of the indexes in the source list. At each index when the output is turned on, a measurement is made and the output is turned off. Settings that you can change before generating the trigger model: ■ Measure Config List ■ Source Config List ■ Delay (default 0.001 s) ■ Reading Buffer (default `defbuffer1`)

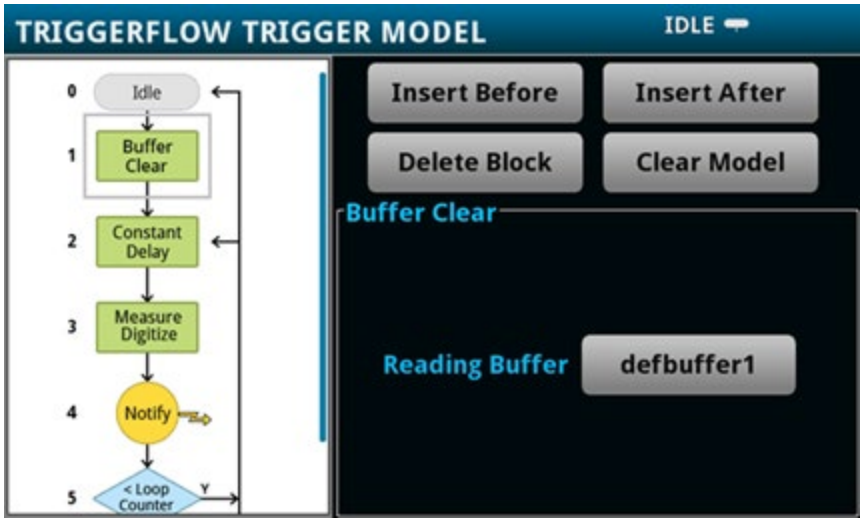
Setting	Description
DurationLoop	Creates a trigger model that makes continuous measurements for a specified amount of time. Settings that you can change before generating the trigger model: ■ Duration (default 1.00 s) ■ Delay (default 0.001 s) ■ Reading Buffer (default `defbuffer1`)
Empty	Clears the present trigger model.
GradeBinning	Creates a trigger model that successively measures components and compares their readings to high or low limits to grade components.
LogicTrigger	Creates a trigger model that waits on an input line, delays, makes a measurement, and sends out a trigger on the output line a specified number of times. Settings that you can change before generating the trigger model: ■ Input Line (default 1) ■ Output Line (default 2) ■ Count (default 10) ■ Delay (default 0.001 s) ■ Trigger Clear (default Never) ■ Reading Buffer (default `defbuffer1`)
LoopUntilEvent	Creates a trigger model that makes continuous measurements until a specified event occurs. Settings that you can change before generating the trigger model: ■ Source Event (default Trigger Key) ■ Position (default 50%) ■ Delay (default 0.001 s) ■ Reading Buffer (default `defbuffer1`) ■ Trigger Clear (default Enter)
SimpleLoop	Creates a trigger model that sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter. Settings that you can change before generating the trigger model: ■ Count (default 10) ■ Delay (default 0.001 s) ■ Reading Buffer (default `defbuffer1`)
SortBinning	Creates a trigger model that successively measures components and compares their readings to high or low limits to sort components.

Trigger Configure menu



The **Configure** menu allows you to view and modify the structure and parameters of a trigger model. You can also monitor trigger model operation.

Figure 49: TRIGGERFLOW TRIGGER MODEL screen



To see the parameters that you can change from the front panel, select a block in the trigger model diagram. The available options change depending on the type of block you select.

From this screen, you can:

- Insert a new trigger block before or after the selected block
- Choose among several block types to add
- Edit an existing block
- Delete an existing block
- Remove all trigger blocks by selecting Clear Model

When you finish your changes to the trigger model, you can initiate the trigger model by pressing the front-panel TRIGGER key.

For detailed information on the trigger model, refer to Trigger model (on page 8-25).

Scripts menu

The menus under Scripts in the main menu allow you to configure, run, and manage scripting operations from the 2470 front panel. Scripts are blocks of commands that the instrument can run as a group. The following topics describe the settings that are available on these screens.

Scripts are presented in alphabetic order. Scripts that are on a USB flash drive are presented after scripts that are loaded on the instrument.

Scripts Run menu



The **Run** menu contains a list of scripts that you can select to run immediately. You can also copy a script to a script that runs each time the instrument power is turned on. You can access scripts that are in the instrument or on a USB flash drive.

Setting	Description
Available Scripts	Displays a list of available scripts that you can select. All scripts that are saved on the 2470 or are on a USB flash drive inserted into the instrument are listed.
Copy to Power Up	Saves the selected script to a script that runs automatically when the instrument is turned on. The script is saved with the script name `autoexec`.
Run Selected	Runs the selected script immediately.

Scripts Manage menu

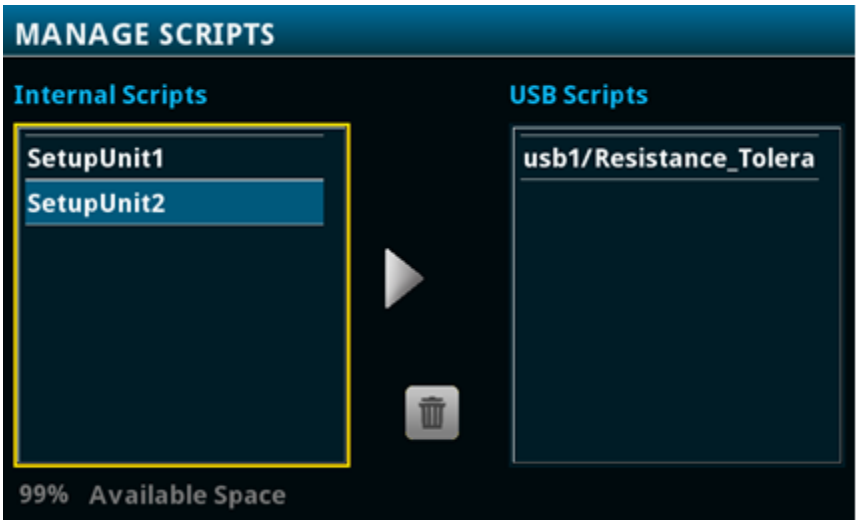


The **Manage** menu allows you to copy scripts to and from the instrument and the USB flash drive. You can also delete scripts from the instrument or USB flash drive.

Setting	Description
>	Copies a script from the instrument to a USB script. A USB flash drive must be inserted before you select this option.
<	Copies a script from a USB flash drive to the instrument. A USB flash drive must be inserted before you select this option.
Delete	Deletes the script that is selected.

For more information about using scripts with the 2470, see Fundamentals of scripting for TSP (on page 13-4).

Figure 50: 2470 MANAGE SCRIPTS menu



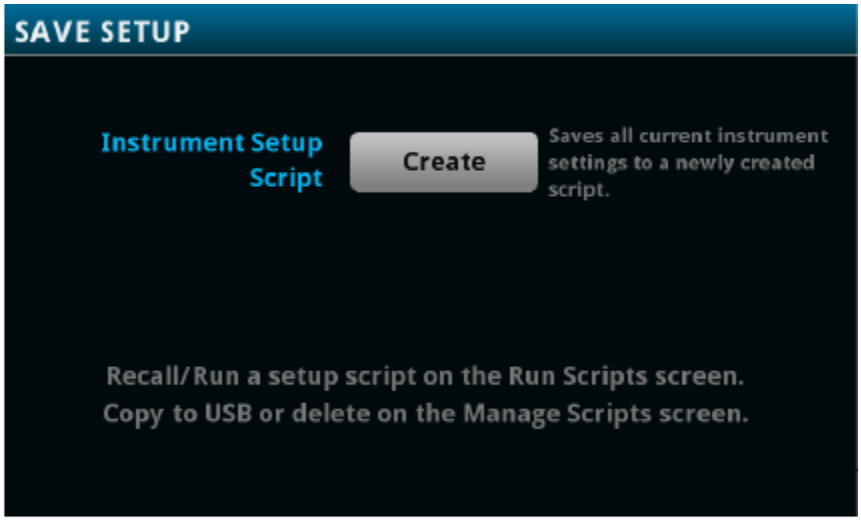
Scripts Save Setup menu



The Save Setup menu allows you to save the present settings and configuration lists of the instrument into a configuration script. You can use this script to recall the settings. Graph settings are not saved.

For more information about user configuration scripts and setups, see Saving setups (on page 3-45).

Figure 51: 2470 SAVE SETUP menu



Scripts Record menu



The options in the **Record** menu allow you to record your actions and store them in a macro script. The script can be run and managed like any other script using the options in the Scripts menu or remote commands. Note that only settings are stored; no key presses or front-panel only options are stored (including pressing the OUTPUT ON/OFF switch).

Setting	Description
Cancel Macro	Stop recording without saving.
Start Macro	Begin recording your selections.
Stop Macro	Stop recording. You are prompted to enter a Macro Script Name.

Scripts Apps menu



The Apps menu opens the Apps Manager. Apps Manager is used to manage prebuilt TSP® applications.

TSP applications are Keithley-developed programs that enable the 2470 to use specialized functions, test automation, and visualize information on the user interface. TSP applications are available when the instrument is used in the TSP or SCPI command set.

The Local tab shows apps that are installed on the 2470. The USB tab shows apps that are on an installed USB drive.

After selecting an application, select **Run** to run the application. To stop running an application, select **End App** from the top right of the screen.

System menu

The menus under System in the main menu allow you to configure general instrument settings from the 2470 front panel. Among these settings are the event log, communications, backlight, time, and password settings.

System Event Log menu



The **Event Log** menu allows you to view and clear event log entries. You can also adjust which events are displayed or logged.

The System Events tab view shows event log entries in a table. Select a line in the table to open a dialog box that contains more detailed information about the event. The event log entries are one of the following types:

- **Error:** An error occurred. When an error occurs, the requested change is not implemented.
- **Warning:** This message indicates that a change occurred that could affect operation.
- **Information:** The message is for information only. This indicates status changes or information that may be helpful. If the Log Command option is on, it also includes commands.

The Log Settings tab view contains settings that affect what data displays on the System Events tab. The following table describes these settings.

Setting	Description
Clear Log	Clears all entries from the event log.
Log Command	Turns the logging of commands on or off. When logging is turned on, the instrument records the commands that are sent to the instrument. It records commands sent from any interface (the front panel or a remote interface).
Log Information	Turns the logging of information messages on or off. If this is turned off, the instrument does not log or display popups for information messages.
Log Warning	Turns the logging of warnings on or off. If this is turned off, the instrument does not log or display popups for warning messages.
Popups	Chooses what type and whether to display popup messages on the front panel. You can choose to display error messages, error and warning messages, or no messages in popups. Messages continue to be saved in the event log when popups are turned off.
Reset Popups	Restores the popups setting to show errors and warnings.
Save to USB	Saves the event log to a CSV file on the USB flash drive. The file name is `eventlog.csv`.
Show Information	Turns the display of information messages on or off. If you turn this off, the instrument continues to record information messages and display popup messages, but does not display them on the System Events tab.
Show Warning	Turns the display of warnings on or off. If you turn this off, the instrument continues to record warnings and display warning popup messages, but does not display them on the System Events tab.

System Communication menu



The **Communication** menu opens a set of tabs that contain information about the communications settings. Most of the tabs contain settings that you can change.

GPIB tab setting	Description
Address	The default GPIB address is 18. You can set the address to any address from 1 to 30 if it is unique in the system. This address cannot conflict with an address that is assigned to another instrument or to the GPIB controller.
GPIB resource string	The VISA instrument connection string is displayed in the lower right.

LAN tab settings	Description
Apply Settings	To save any changes you made on the LAN tab, select **Apply Settings**.
Gateway	Displays the present gateway address. When TCP/IP Mode is set to Manual, you can set the gateway address.
IP Address	Displays the present IP address. When TCP/IP Mode is set to Manual, you can set the IP address.
LXI LAN Reset	Sets the TCP/IP Mode to Auto and clears the IP address, gateway, and subnet mask. Resets the LAN password.
LXI LAN indicator	The LXI LAN indicator illuminates when the connection is established.
MAC Address	Read-only text that shows the present media access control (MAC) address of the instrument.
Subnet	Displays the present subnet mask address. When TCP/IP Mode is set to Manual, you can set the subnet mask address.
TCPIP resource strings	The VISA instrument connection strings are displayed in the lower right.
TCP/IP Mode	Select **Auto** to set the instrument to automatically obtain an IP address. Select **Manual** to manually set the IP address, gateway, and subnet mask values.

TSP-Link tab settings	Description
Node	Sets the TSP-Link node number for the instrument (1 to 63). Each instrument or enclosure attached to the TSP-Link expansion interface is called a node. Each node must be identified with a unique node number. This identification is called a TSP-Link node number.
Initialize	Select Initialize to have the 2470 find all connected TSP-Link instruments and form a network.

USB tab setting	Description
USB resource string	The VISA instrument connection string is displayed in the lower right.

System Settings menu



The **Settings** menu contains general instrument settings.

Setting	Description
Audible Errors	Turns the beeper on or off. When the beeper is on, the beeper sounds when an event or error occurs. The audible error setting is not affected by instrument reset or power cycle. For more information, see Instrument sounds (on page 3-44).
Backlight Brightness	Adjusts the brightness of the front-panel display. The sliding adjustment scale adjusts the brightness level.
Backlight Dimmer	Sets the front-panel display to dim after a period (1, 4, or 8 hours) or never.
Command Set	Specifies the type of commands to use when controlling the instrument from a remote interface (SCPI or TSP).
Interface Access	Specifies the request access for control interface before taking control of the instrument: Full, Exclusive, Protected, or Lockout. For details, see Interface access (on page 2-34).
Key Click	Turns the sound that occurs when you press a front-panel key On or Off. The key-click setting is not affected by instrument reset or power cycle.
Line Frequency	Displays the line frequency detected by the instrument. The line frequency is automatically detected and cannot be changed.
Password	Contains the password if the instrument is set to use an access mode that requires a password. The 2470 is programmed with a default user name and password (case-sensitive): ■ User name: `admin` ■ Password: `admin` You can change the password. See Instrument access (on page 2-34) for more information about controlling access to the instrument.
Reading Format	Sets the format of the front-panel readings to Prefix (adds a prefix to the units symbol, such as k, m, or µ) or Exponent.
Time and Date	Sets the instrument month, day, year, hour, and minute.

System Calibration Menu



This read-only screen shows factory adjustment and verification dates.

Information	Description
Adjust Date	The date when the instrument was adjusted through a factory calibration.
Adjust Count	The adjustment count is the number of times the instrument has been factory calibrated.
Calibration Date	The date when the instrument calibration was last verified.

System Info/Manage menu



The **Info/Manage** menu gives you access to version and serial number information and settings for instrument firmware and reset functions.

Setting	Description
Downgrade to Older	Returns the 2470 to a previous version of the firmware from a file on a USB flash drive.
Password Reset	Resets the access password to the default value.
Product Demo	Configures a brief demonstration of the graphing capability of the 2470. To get correct results, you must have the appropriate demonstration fixture connected to the inputs. For more information, contact your local Keithley office, sales partner, or distributor.
Serial Number	Displays the serial number of the instrument.
System Reset	Resets many of the instrument settings to their default values. For more information about what settings get reset, see Reset default values (on page 5-28).
Upgrade to New	Initiates a firmware upgrade from a file on a USB flash drive.
Version	Displays the version of firmware that is installed in the instrument.

APPS Manager

TSP® applications are Keithley-developed programs that enable the 2470 to use specialized functions, test automation, and visualize information on the user interface. TSP applications are available when the instrument is using either the TSP or SCPI command set. Applications may be preinstalled on your 2470.

To access the APPS Manager:

1. Press the **MENU** key.
2. Select **Apps** in the Scripts column. The APPS Manager screen is displayed.

Selecting either the Local or USB tabs on the APPS Manager screen shows the applications that are installed on the 2470 or on an installed USB drive. After selecting an application from the APPS Manager, select Run to run the application. To stop running an application, select End App from the top right of the screen.

Download and run TSP applications

If an application is removed from your 2470 or a new application is made available, you can download the application and install it on your 2470.

To download and run TSP applications from your computer:

1. Download the TSP® application from tek.com/keithley.
2. Save and unzip the file onto the root directory of a USB drive.
3. Insert the USB drive into the front panel of the instrument.
4. Press the **MENU** key.
5. Select **Apps** in the Scripts column.
6. Select the **USB** tab in the APPS MANAGER.
7. Select an application. A brief description of the application, including the name, function, and instrument compatibility, is displayed.
8. To run the application, select **Run**.

To save the application to the internal memory as a local application, select the applications and select **Save**. The application is now available on the Local tab.

To delete the application, select the application and select **Delete**.

Display features

You can set the front-panel display to display the units of measure, number of digits, and customized text messages for your applications.

Setting the number of displayed digits

You can change the number of digits that are displayed for measurement readings on the front panel. You can display 3½, 4½, 5½, or 6½ digits. The default is 5½.

The number of displayed digits does not affect accuracy or speed of the measurements. It also does not affect the format of readings that are returned from a remote command.

From the front panel:

1. Set TERMINALS to **FRONT**.
2. Press **MENU**.
3. Under Measure, select **Settings**.
4. Set **Display Digits**.

This setting takes effect the next time you make a measurement.

From a remote interface:

- SCPI commands: Refer to :DISPlay:<function>:DIGits (on page 12-29).
- TSP commands: For measure functions, refer to smu.measure.displaydigits (on page 14-138).

Setting the display format

You can set the format of units that are displayed for measurement readings on the front panel. The formats are:

- **Prefix:** Add a prefix to the units symbol, such as k, m, or µ.
- **Exponent:** Replace the units symbol with exponents.

See the following figures for examples of each display format.

Figure 52: 2470 prefix display format

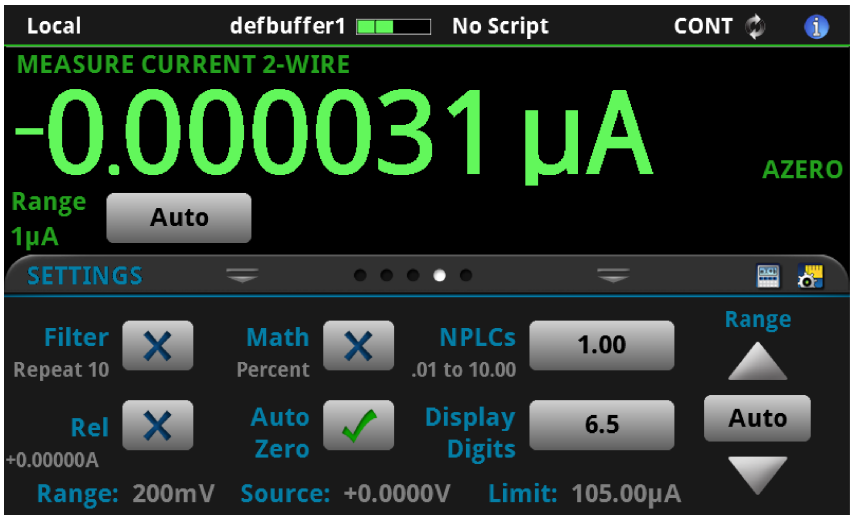
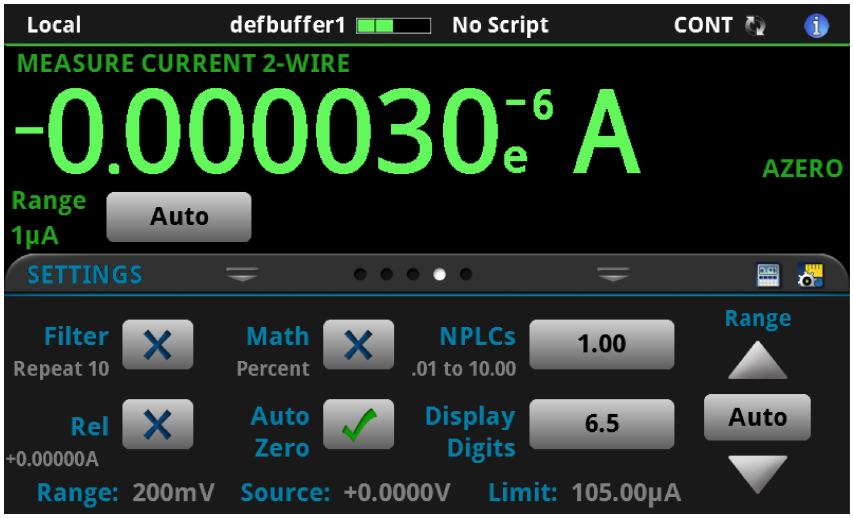


Figure 53: Exponent display format



From the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Reading Format.
4. Select the reading format (**Prefix** or **Exponent**).

This setting takes effect immediately.

Over a remote interface:

- **SCPI commands:** Refer to :DISPlay:READing:FORMat (on page 12-31)
- **TSP commands:** Refer to display.readingformat (on page 14-74)

Customizing a message for the USER swipe screen

You can customize the message that is displayed on the USER swipe screen.

You must use a remote interface to customize the USER swipe screen.

Creating a message

When you create the message, you can send text that will be used on the top and bottom lines of the USER swipe screen. The top line allows up to 20 characters and the bottom line allows up to 32 characters.

The examples shown here switch the display to the USER swipe screen, set the first line to read `Test in process` and the second line to display `Do not disturb`.

Using SCPI commands:

Send the commands:

```
DISPlay:SCReen SWIPE_USER
DISPlay:USER1:TEXT "Test in process"
DISPlay:USER2:TEXT "Do not disturb"
```

Using TSP commands:

Send the commands:

```
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, "Test in process")
display.settext(display.TEXT2, "Do not disturb")
```

Clearing the USER swipe screen

You can clear the message that is displayed on the USER swipe screen.

To clear the message using SCPI commands, send the command:

```
:DISPlay:CLEar
```

To clear the message using TSP commands, send the command:

```
display.clear()
```

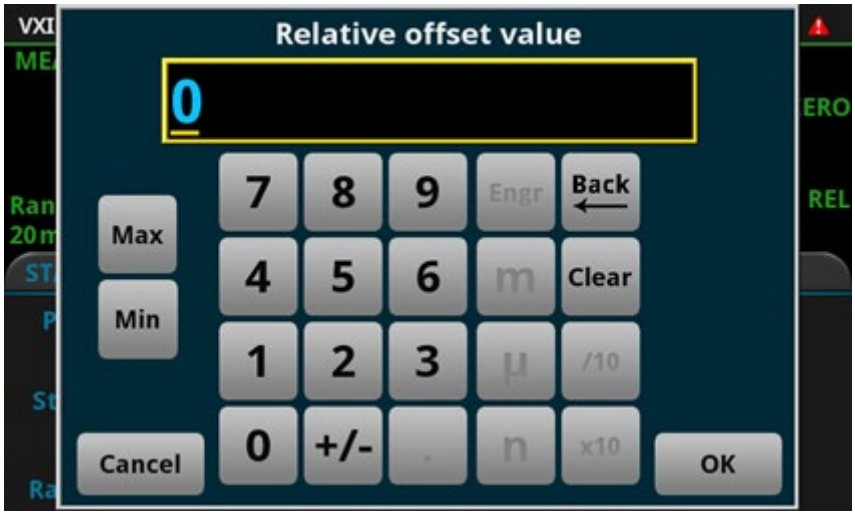
Creating messages for interactive prompts

If you are using the TSP command language and scripts, you can set up scripts that prompt the operator to enter information from the front-panel display of the instrument.

The options that you can define include:

- Display a number pad so that operator can enter a value.
- Display a custom button that the operator can press.
- Display a message and a predefined set of buttons that the operator can respond to.
- Display a keypad so that the operator can enter information, as shown in the example below.

Figure 54: Input number example



For more information on creating the interactive prompts, see the following command descriptions:

- display.input.number() (on page 14-66)
- display.input.option() (on page 14-68)
- display.input.prompt() (on page 14-69)
- display.input.string() (on page 14-70)

Save screen captures to a USB flash drive

You can save a screen capture of the front-panel display to a graphic file on a USB flash drive. The instrument saves the graphic file in PNG file format.

To save a screen capture:

1. Insert a USB flash drive into the USB port on the front panel of the instrument.
2. Navigate to the screen you want to capture.
3. Press the **HOME** and **ENTER** keys. The instrument displays `Saving screen capture`.
4. Release the keys.

Instrument sounds

The instrument can emit a beep when a front-panel key is pressed or when a system event occurs. You can turn these beeps on or off.

Through the remote interface, you can generate a beep with a defined length and tone. This is typically used as part of code to indicate that something has occurred.

To change the beeps when system events occur (setting is only available from the front panel):

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Set Audible Errors to **On** or **Off**.

To turn the key clicks on or off (setting is only available from the front panel):

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Set Key Click to **On** or **Off**.

To generate an audible tone from the SCPI remote interface, send the command:

```
:SYSTem:BEEPer <frequency>, <duration>
```

Where `frequency` is the frequency of the sound in Hertz (20 to 20000) and `duration` is the length of the sound in seconds.

To generate an audible tone from the TSP command interface, send the command:

```
beeper.beep(duration, frequency)
```

Where *`duration`* is the length of the sound in seconds and *`frequency`* is the frequency of the sound in Hertz (20 to 20000).

Saving setups

You can save the present settings and any configuration lists that you have defined for the 2470 to internal memory or an external USB flash drive.

After the settings are saved, you can recall the settings. You can also set them to be the default settings when the instrument is powered on.

If you are using TSP commands, saved setups are scripts and can be added, modified, and deleted like any other script. See Introduction to TSP operation (on page 13-1) for additional information about working with scripts.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press HOME and ENTER to save an image of the settings with the screen capture feature. Refer to Save screen captures to a USB flash drive (on page 3-44) for additional information.

Save a user setup to internal memory

From the front panel:

1. Configure the 2470 to the settings that you want to save.
2. Press the **MENU** key.
3. Under Scripts, select **Create Setup**.
4. Select **Create**. A keyboard is displayed.
5. Use the keyboard to enter the name of the script.
6. Select the **OK** button on the displayed keyboard. The script is added to internal memory.

Using SCPI commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
   `*SAV <n>`
   Where <n> is an integer from 0 to 4.

NOTE

In the front-panel script menus, the setups saved with the `*SAV` command have the name `Setup0`*x*, where *x* is the value you set for <n>.

Using TSP commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
   `createconfigscript("setupName")`
   Where *`setupName`* is the name of the setup script that is created.

Save a user setup to a USB flash drive

NOTE

You cannot save to the flash drive using SCPI commands.

From the front panel:

1. Save the user setup to internal memory, as described in Save a user setup to internal memory (on page 3-45).
2. Insert the USB flash drive into the USB port on the front panel.
3. Press the **MENU** key.
4. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
5. In the Internal Scripts list, select the script you want to copy to the USB flash drive.
6. Select **>**. The file is transferred to the USB flash drive, and the corresponding file name is displayed in the USB Scripts box.

Using TSP commands:

1. Save the user setup to internal memory, as described in Save a user setup to internal memory (on page 3-45).
2. Insert the USB flash drive into the USB port on the front panel.
3. Send the command:

```
setupName.save("/usb1/USBSetupName")
```

   Where *`setupName`* is the name of the user setup and *`USBSetupName`* is the name of the file on the USB flash drive. You can use the same name for *`setupName`* and *`USBSetupName`*.

Copy a user setup

To copy a user setup from an external USB flash drive to the instrument from the front panel:

1. Insert the USB flash drive into the USB port on the front panel.
2. Press the **MENU** key.
3. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
4. In the USB Scripts list, select the script you want to copy from the USB flash drive.
5. Select **<**. The file is transferred to the instrument, and the corresponding file name is displayed in the Internal Scripts box.

Delete a user setup

To remove a user setup from internal memory or the USB flash drive from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
3. Under Internal Scripts or USB Scripts, select the name of the script.
4. Select **Delete**. A confirmation message is displayed.
5. Select **OK**.

To delete a user setup from internal memory using SCPI commands:

You must overwrite an existing setup with the new setup. See Save a user setup to internal memory (on page 3-45).

To delete a user setup from internal memory using TSP commands, send the command:

```
script.delete("setupName")
```

Where *setupName* is the name of the script that will be deleted.

Recall a user setup

You can recall setups from internal nonvolatile memory or a USB flash drive. When you recall a setup, you run a script that restores the instrument to the settings that are saved in that script.

To recall a saved setup from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. In the Available Scripts list, select the script you want to recall. USB scripts have the prefix `usb1/`.
4. Select **Run Selected**.

To recall a user setup from internal memory using SCPI commands, send the command:

```
*RCL <n>
```

Where <n> is an integer from 0 to 4 that represents the saved script.

To recall a saved setup using TSP commands, send the command:

```
setupName()
```

Where *setupName* is the name of the script that contains the setup that was saved with `createconfigscript()`.

Define the setup used when power is turned on

You can select a configuration to be used when power is turned on.

From the front panel:

1. Set the instrument to the settings that you want it to have each time the power is turned on.
2. Press the **MENU** key to open the main menu. Under Scripts, select **Create Setup**.
3. Select **Create**. A keyboard is displayed.
4. Enter the name of the new script, and then select **ENTER** on the keyboard to save it. The instrument saves all present system settings to the script and displays a confirmation message.
5. Select **OK**.
6. Press the **EXIT** key to return to the main menu.
7. Under Scripts, select **Run**. The RUN SCRIPTS window opens.
8. Select the script you created.
9. Select **Copy to Power Up**.
10. Select **OK** on the confirmation message.

Using a SCPI command, send the command:

```
:SYSTem:POSetup <name>
```

Where `<name>` is:

- `RST`: Use the *RST defaults.
- `SAV0`: Use the setup stored at memory location 0
- `SAV1`: Use the setup stored at memory location 1
- `SAV2`: Use the setup stored at memory location 2
- `SAV3`: Use the setup stored at memory location 3
- `SAV4`: Use the setup stored at memory location 4

Using a TSP command:

Save the script that you want to use as the power-on default to be `autoexec`. For example, to save the commands that are presently in the instrument to be the power-on defaults, send the command:

```
createconfigscript("autoexec")
```

NOTE

If an `autoexec` script already exists, you must delete it by sending the `script.delete("autoexec")` command. Performing a system reset does not delete the `autoexec` script.

Resets

There are several types of resets in the 2470.

In general, the terms "reset," "instrument reset," and "system reset" refer to the reset that is performed when you send the `*RST` or `reset()` command, or when you select **MENU > System > Info/Manage > System Reset** from the front panel. It resets most commands to their default values. For more information about the settings that get reset, refer to Reset default values (on page 5-28).

The instrument also responds to other types of resets. These resets include:

- **SMU reset:** This reset is only available if you are using the TSP command set. The `smu.reset()` function turns off the output and resets any commands that begin with `smu.` to their default values. Refer to smu.reset() (on page 14-169).
- **Password reset:** This resets the instrument password to its default value. You can reset the password by pressing the **MENU** key, selecting **Info/Manage** (under System), and selecting **Password Reset**. When you do this, the password returns to the default setting. Refer to Instrument access (on page 2-34).
- **Digital line reset:** This resets digital I/O line values to their factory defaults if you are using the TSP command set. If you are using SCPI, the lines are reset when the system is reset.
- **LAN reset:** This resets the LAN settings and the instrument password to the system default values. To do this reset, insert a straightened paper clip into hole below LAN RESET on the rear panel. For the location of LAN RESET, refer to Rear panel overview (on page 3-3).
- **Status preset:** This resets all bits in the status model. If you are using the SCPI command set, refer to :STATus:PRESet (on page 12-100). If you are using the TSP command set, refer to status.preset() (on page 14-208).
- **Trigger blender reset:** This reset is only available if you are using the TSP command set. Resets some of the trigger blender settings to their factory defaults. Refer to trigger.blender[N].reset() (on page 14-219).
- **Trigger timer reset:** This reset is only available if you are using the TSP command set. Resets trigger timer settings to their default values. Refer to trigger.timer[N].reset() (on page 14-283).
- **TSP-Link line reset:** This reset is only applicable if you are using TSP-Link. Resets some of the TSP-Link trigger attributes to their defaults. Refer to tsplink.line[N].reset() (on page 14-298).
- **TSP-Net reset:** This reset is only applicable if you are using TSP-NET. Disconnects all TSP-Net sessions. Refer to tspnet.reset() (on page 14-309).

Reset the instrument

You can reset many of the instrument settings to their default values. For detail on what gets reset, see Reset default values (on page 5-28). Default values are also listed in the command descriptions.

The reading buffer is reset to defbuffer1 when the instrument is reset. Configuration lists are removed when a system reset occurs.

NOTE

If you are connected to a TSP-Link system, resetting the instrument resets all TSP-Link-enabled instruments on the TSP-Link system. If you are using TSP commands and you want to reset only the local instrument, send `localnode.reset()` instead of `reset()`.

NOTE

Reset restores swipe screens to the factory defaults and removes any user-created swipe screens.

Using the front panel:

1. Press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **System Reset**.

The commands are reset and a confirmation message is displayed.

Using SCPI commands, send the command:

```
*RST
```

Using TSP commands, send the command:

```
reset()
```

Using the event log

The event log records events, which can be errors, warnings, and information reported by the instrument. Through the Event Log menu, you can view these events. You can also specify which events are shown in the event log, which ones are logged, and which ones generate popup messages.

Information provided for each event log entry

Each event log entry includes the following information:

- The date and time when the event occurred in 24-hour time format (MM/DD HH:MM)
- The code number of the event; if you are using a remote interface, you can use this number with the status model to map events to bits in the event registers
- The type of event (separate icons for informational, error, or warning)
- The description of the event

To access an event log listing from the front panel:

1. Press the **MENU** key.
2. Under System, select **Event Log**.
3. Select the **System Events** tab. A list of events is displayed.
4. If the events fill the page, you can scroll down to see additional events.
5. To view additional detail about an event, select the event. A dialog box with additional detail is displayed.

Event log settings

You can set which events you can see in the instrument event log, and which events cause a status message indicator to be displayed on the front panel of the instrument. You can also choose whether or not to log all commands the instrument receives in the event log, which can be useful for troubleshooting problems. You can save the contents of the event log to a USB flash drive. You can clear the event log.

To access event log settings from the front panel:

1. Press the **MENU** key.
2. Under System, select **Event Log**.
3. Select the **Log Settings** tab. A list of settings is displayed.
4. Make the settings as needed.

The options available on this tab are described in the table below.

Settings tab settings	Description
Show Warning	Turns the display of warnings on or off. If you turn this off, the instrument continues to record warnings and display warning popup messages, but does not display them on the System Events tab.
Show Information	Turns the display of information messages on or off. If you turn this off, the instrument continues to record information messages and display popup messages, but does not display them on the System Events tab.
Log Warning	Turns the logging of warnings on or off. If this is turned off, the instrument does not log or display popups for warning messages.
Log Information	Turns the logging of information messages on or off. If this is turned off, the instrument does not log or display popups for information messages.
Log Command	Turns the logging of commands on or off. When logging is turned on, the instrument records the commands that are sent to the instrument. It records commands sent from any interface (the front panel or a remote interface).
Popups	Turns the display of popups on or off. Options are: ■ Errors: Turn off the display of error popups. ■ Errors and Warnings: Turn off the display of error and warning popups. ■ None: Turn off the display of all popups.
Reset Popups	Restores the popups setting to show errors and warnings.

Settings tab settings	Description
Save to USB	Saves the event log to a CSV file on the USB flash drive. The file name is `eventlog.csv`.
Clear Log	Clears all entries from the event log.

Effects of errors on scripts

Most errors will not abort a running script. The only time a script is aborted is when a Lua run-time (event code -286, "TSP runtime error") is detected. Run-time events are caused by actions such as trying to index into a variable that is not a table.

Syntax errors (event code -285, "Program syntax") in a script or command will prevent execution of the script or command.

Saving front-panel settings into a macro script

You can save some settings made through the front panel into a macro script that you can run later.

The settings that are saved include any settings made through:

- QuickSet menu (except the Performance slider, which cannot be used when recording a macro)
- Source menu: Settings, Sweep, and Config Lists
- Measure menu: Settings, Calculations, Config Lists, and Reading Buffers
- Trigger menu: Templates and Configure
- System Communication
- Time and date

NOTE

Only settings are stored; no front-panel only options or key presses (including the OUTPUT ON/OFF switch) are stored.

It also saves the reading format, interface access, and system reset settings.

Macro scripts are limited to 10 kB for each script.

Recording a macro script

To record a macro script:

1. Press the **MENU** key.
2. Under Scripts, select **Record**.
3. Select the **Start Macro** button.
4. Make the settings that you want to record.
5. Press the **MENU** key.
6. Under Scripts, select **Record**.
7. Select the **Stop Macro** button. The Macro Script Name dialog box is displayed.
8. Enter a name for the script.
9. Select the **OK** button.

NOTE

You can also stop or cancel recording from the home screen. Select the **Recording** indicator in the indicator bar.

After you create a macro script, you can use the other Scripts menu options to run and manage scripts. Refer to Scripts menu (on page 3-33) for information on the options.

Running a macro script

You can run a macro script from the front panel or from a remote interface.

To run a macro script from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. Select the macro script to run.
4. Select **Run Selected**.

Using SCPI commands:

```
SCRipt:RUN "scriptName"
```

Where *scriptName* is the name of the macro script to run.

Using TSP commands:

```
scriptVar.run()
```

Where *scriptVar* is the name of the macro script to run.

Front-panel macro recording limitations

When you are recording a macro script from the front panel, the settings you make are recorded at the speed at which you make them. However, when the macro you created is run, it runs at remote command processing speed. This can be a problem when working with trigger models and other features that require time to finish processing before remaining commands can process.

For example, if you record a macro that includes a trigger model that you initiate followed by other settings changes or additional trigger initiate actions, an error message is generated. This is because the trigger model takes time to complete, but the macro recording from the front panel does not add `waitcomplete()` commands or other delay settings to the script that allow the trigger model to finish before processing the other commands.

Section 4

Sourcing and measuring

In this section:

Test connections 4-1
Source-measure overview 4-20
Protection 4-35
Ranges 4-39
Automatic reference measurements 4-44
Source readback 4-45
Source delay 4-46
Relative offset 4-47
Calculations that you can apply to measurements 4-50
Sweep operation 4-54
Limit testing and binning 4-66
Configuration lists 4-82

Test connections

WARNING

To prevent electric shock, test connections must be configured such that the user cannot come in contact with conductors or any device under test (DUT) that is in contact with the conductors. It is good practice to disconnect power before connecting DUTs. Safe installation requires proper shields, barriers, and grounding to prevent contact with conductors.

There is no internal connection between protective earth (safety ground) and the LO terminals of the 2470. Therefore, hazardous voltages (more than 30 V_{RMS}) can appear on LO terminals. This can occur when the instrument is operating in any mode. To prevent hazardous voltage from appearing on the LO terminals, connect the LO terminal to protective earth (safety ground) if your application allows it. You can connect the LO terminal to the chassis ground terminal on the front panel or the chassis ground screw terminal on the rear panel. Note that the front-panel terminals are isolated from the rear-panel terminals. Therefore, if you are using the front-panel terminals, ground to the front-panel LO terminal. If using the rear-panel terminals, ground to the rear panel LO terminal. Failure to follow these guidelines can result in injury, death, or instrument damage.

NOTE

On some sensitive or easily damaged devices under test (DUTs), the instrument power-up and power-down sequence can apply transient signals to the DUT that may affect or damage it. When testing this type of DUT, do not make final connections to it until the instrument has completed its power-up sequence and is in a known operating state. When testing this type of DUT, disconnect it from the instrument before turning the instrument off.

To prevent any human contact with a live conductor, connections to the DUT must be fully insulated and the final connections to the DUT must only use safety-rated safety jack socket connectors that do not allow bodily contact.

You can make test connections to the 2470 from the rear or front panel of the instrument.

Basic connections

WARNING

The front and rear terminals of the instrument are rated for connection to circuits rated Measurement Category O only, with transients rated less than 1500 V_{PEAK} above the maximum rated input. Do not connect the instrument terminals to CAT II, CAT III, or CAT IV circuits. Connection of the instrument terminals to circuits higher than CAT O can cause damage to the equipment and expose the operator to hazardous voltage.

Do not exceed the maximum allowable voltage differentials. Exceeding the voltage differentials can result in electric shock and damage to the equipment.

Common mode voltage must be externally limited to 250 V DC, 1.05 A maximum. Failure to limit the common mode voltage can result in electric shock and damage to the equipment.

You can access the FORCE HI, FORCE LO, SENSE LO, and SENSE HI connections from the front or rear panel of the instrument. The front panel has banana jack connections and the rear panel has triaxial connections.

The front panel of the instrument shows the maximum allowable voltage differentials between terminals. The maximum common mode voltage is the voltage between FORCE LO and chassis ground. You must limit the current from an external common mode voltage source. You can use protective impedance or a fuse to limit the current.

The guard connections are only available from the rear panel of the instrument.

To remove a triaxial cable from the rear panel, turn the cable connector counterclockwise and pull it off the rear panel.

When making or breaking connections, follow these guidelines:

- Power off the 2470 and all other instruments.
- Disconnect any devices that may deliver energy.
- Make connections to the device under test through a test fixture or other safe enclosure.
- Make sure the 2470 is properly connected to protective earth (safety ground).
- If the test fixture is conductive, make sure the test fixture is properly connected to protective earth (safety ground).
- Make sure the test fixture provides proper protection.
- Properly make interlock connections between the 2470, the test fixture, and any other instruments.
- Make sure to follow all warnings and cautions and to take adequate safety precautions for each set of connections.
- Properly terminate any cables. All unterminated cable ends must be in a safe enclosure.
- See Two-wire local sense connections (on page 4-8) and Four-wire remote sense connections (on page 4-10) for examples of connections.
- For information about the interlock, see Using the interlock (on page 4-3).

Using the interlock

The instrument provides an interlock circuit on the rear panel. You must enable this circuit in order for the instrument to set source voltages greater than ±42 V DC.

When the safety interlock signal is asserted, the following actions occur:

- All voltage ranges of the instrument are available.
- The green front-panel INTERLOCK indicator is on.

The action when the interlock signal is not asserted depends on the Interlock setting. If Interlock is set to Off, if the safety interlock signal is not asserted, the following occurs:

- The nominal output is limited to less than ±42 V.
- The front-panel INTERLOCK indicator is not illuminated.
- You can output voltages less than ±42.

If Interlock is set to On, when the safety interlock signal is not asserted, the following occurs:

- You cannot turn on the source output.
- The front-panel INTERLOCK indicator is not illuminated.
- Whenever the interlock changes state (from asserted to not asserted or vice versa), the output is turned off.

To change the Interlock setting:

- From the front panel, select **MENU**. Select **Source Settings** and set **Interlock** to **ON** or **OFF**.
- Using SCPI commands, refer to :OUTPut[1]:INTerlock:STATe (on page 12-39).
- Using TSP commands, refer to smu.interlock.enable (on page 14-120).

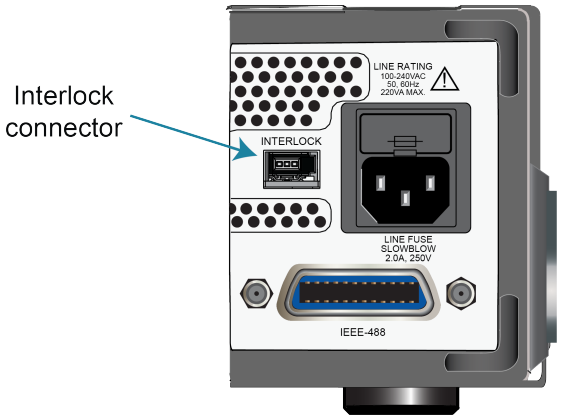
If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, you see event code 5074, “Output voltage limited by interlock.” Note that the SOURCE swipe screen displays the value that is selected for the voltage source.

⚠ WARNING

The 2470 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

Location of the interlock connection

Figure 55: Rear-panel interlock location



Interlock connector pins

An interlock circuit is provided on the rear panel of the instrument. This circuit must be closed to enable the 2470 to produce voltages greater than ±42 VDC.

The interlock is intended for use through a normally open switch, which may be installed on the lid of a test fixture, on the enclosure of a semiconductor prober or device handler, or on the door or doors of a test equipment rack. The circuit opens when an access door is opened and closes when the door is closed.

When the interlock is asserted, the FORCE and GUARD terminals should be considered hazardous voltages, even if they are programmed to a non-hazardous voltage or current.

WARNING

Potentially hazardous voltages of up to approximately 1350 V may be present at the High Force, High Sense, and Guard terminals when the interlock circuit is closed.
To prevent electrical shock, do not expose these lines.

You can use the Keithley Instruments connector CS-1616-3 Safety Interlock Mating Connector, supplied with the 2470, to make the interlock connection to the rear panel. You must supply connection wire. The recommended wire is:

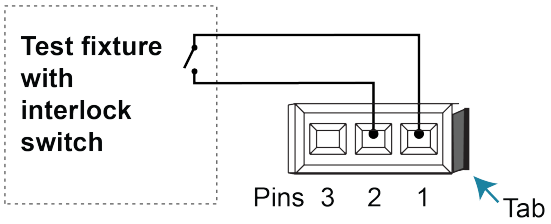
- 20 AWG to 24 AWG copper alloy
- 7 to 19 bare and tinned strands
- 0.20 mm^2 to 0.50 mm^2
- Flexible vinyl, semi-flexible vinyl, polyethylene, x-linked polyethylene, or PTFE

To ensure proper interlock operation, the combined resistance of the external interlock switch and connection wires must be less than 10 Ω when the switch is closed.

The interlock pin locations and connections are shown in the following figure. The pins are:

- Pin 3: Earth and chassis ground
- Pin 2: Interlock
- Pin 1 (next to tab): +6 VDC out (current limited)

Figure 56: 2470 interlock pins



To assemble the interlock:

1. Insert the wire into CS-1616-3.
2. Use a pair of pliers to squeeze the connector sections together.

You cannot disassemble connector and reuse it.

Front- or rear-panel test connections

You can use either the front-panel or the rear-panel terminals to make connections to the device under test (DUT). The instrument must be set to use either the front or the rear terminals.

NOTE

You cannot make some connections to the front-panel terminals and some to the rear-panel terminals for the same test setup. All connections for the same test must be made to either the front-panel or the rear-panel terminals.

WARNING

Be aware that hazardous voltages can appear on the LO terminals even if the terminals are not presently selected. The TERMINALS switch selects the active terminals for the measurement. It does not disconnect the terminals.

Determining whether to use front or rear terminals

The terminals on the front panel are banana jack connectors, and the rear-panel terminals are triaxial connectors. Depending on your test setup, the test environment, and the precision of your measurements, you may see different results between measurements made from the front and rear terminals.

For the most precise measurements, use the rear-panel triaxial terminals. You will get the best results with rear-panel connections if you are making measurements or sourcing current in the 10 nA and 100 nA ranges or when making bipolar junction transistor (BJT) measurements.

You may also want to use the rear-panel terminals if the test environment is electrically noisy. The shielding on the triaxial cables will prevent environmental noise from affecting measurements.

Setting the instrument to use the front or rear terminals

NOTE

If the output is on when you change the settings for the terminals that are used, the output turns off.

Using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measure combination
3. Press the **TERMINALS FRONT/REAR** switch.

When F is lit, the instrument is using the front-panel terminals. When R is lit, the instrument is using rear-panel terminals.

Using SCPI commands:

To change to the front-panel terminals for current measurements, send the command:

```
ROUTe:TERMinals FRONt
```

To change to the rear-panel terminals for current measurements, send the command:

```
ROUTe:TERMinals REAR
```

Using TSP commands:

To change to the front-panel terminals, send the command:

```
smu.terminals = smu.TERMINALS_FRONT
```

To change to the rear-panel terminals, send the command:

```
smu.terminals = smu.TERMINALS_REAR
```

Two-wire compared to four-wire measurements

You can use 2-wire or 4-wire measurement techniques with the 2470.

You should use 4-wire, or remote sense, measurement techniques for the following conditions:

- Low-impedance applications
- When sourcing high current
- When sourcing low voltage
- When sourcing higher currents and measuring low voltages
- When enforcing voltage limits directly at the device under test (DUT)
- When sourcing or measuring voltage in low-impedance (less than 100 Ω) test circuits
- When optimizing the accuracy for low resistance, voltage source, or voltage measurements

Use 4-wire connections when you are concerned about voltage drops because of lead or contact resistance that could affect measurement accuracy. This can occur on low-impedance devices when you are sourcing or measuring voltage, especially in semiconductor device testing. For example, when testing low-impedance devices (less than 100 Ω), usually a higher current is sourced and small voltages are measured.

Sourcing current and measuring voltage drops in a 4-wire configuration is used when measuring resistivity of a material using a 4-point collinear probe.

It is sometimes necessary to use a 4-wire configuration when sourcing small voltages (less than 1 V) and measuring current. This is true when performing I-V tests on semiconductor devices such as diodes.

When you source or measure voltage in a low-impedance test circuit, there can be errors because of lead resistance. Use 4-wire remote sensing to eliminate these errors. If you use 4-wire remote sensing when you source voltage, the programmed voltage is delivered to the DUT. If you use 4-wire remote sensing when you measure voltage, only the voltage drop across the DUT is measured.

The maximum voltage drop between the force and sense leads is 5 V.

NOTE

When the output is off, the remote sense lines are disconnected and 4-wire sensing is disabled. When the output is off, the instrument uses 2-wire sense, regardless of the sense setting. When the output is on, the selected sense setting is used.

You can use 2-wire, or local sensing, measurement techniques for the following source-measure conditions:

- Sourcing and measuring low current.
- Sourcing and measuring voltage in high impedance (more than 1 kΩ) test circuits.
- Measure-only operation (voltage or current).

When you use 2-wire sensing, voltage is sensed at the output connectors.

You should only use 2-wire connections if the error contributed by test-lead IR drop is acceptable.

Two-wire local sense connections

Two-wire connections are shown in the following figures.

If your application results in impedances above 1 GΩ, you may also need to use guarding. This prevents leakage current from affecting measurement accuracy. For information, see Guarding (on page 5-15).

To use 2-wire connections, you must set the sense mode of the instrument to 2-wire, as described in the following topics.

Figure 57: 2470 2-wire front-panel connections

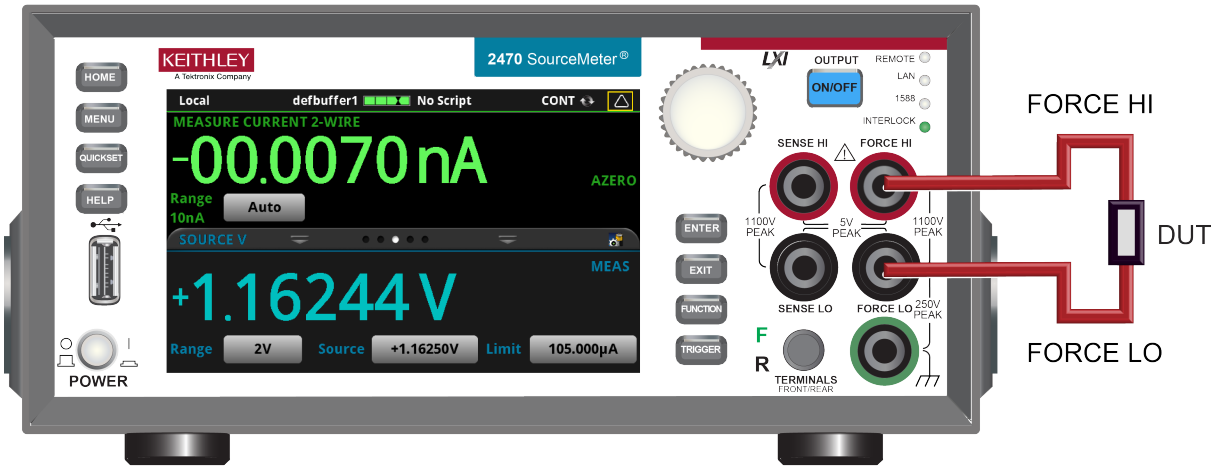
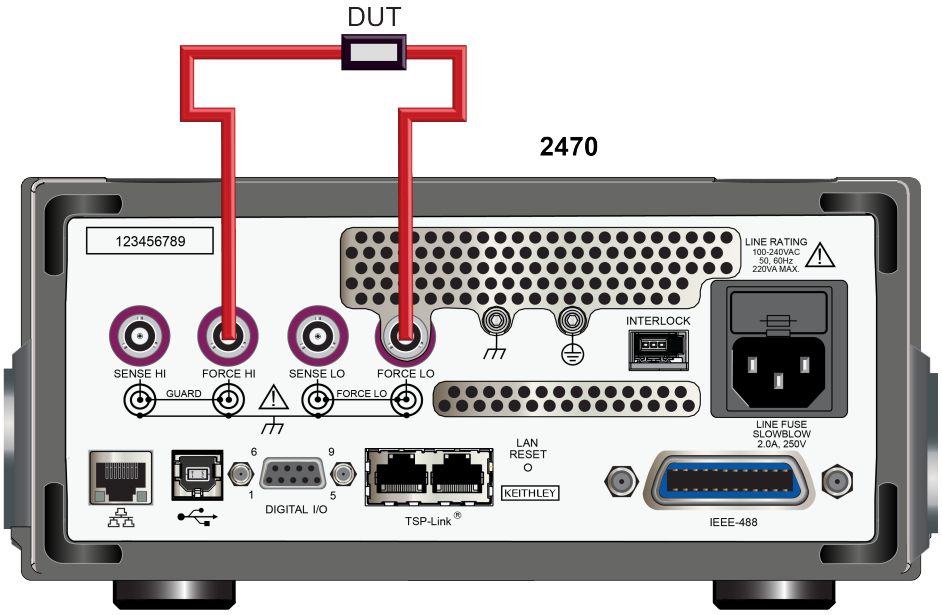


Figure 58: 2470 2-wire rear-panel connections



Using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measure combination.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. Select the button next to Sense and select **2-Wire Sense**.
6. Press the **HOME** key to return to the operating display.

Using SCPI commands:

To change to 2-wire sensing for current measurements, send the command:

```
:SENSe:CURRent:RSENse OFF
```

To change to 2-wire sensing for voltage, replace `CURRent` with `VOLTage`. For resistance, replace `CURRent` with `RESistance`.

Using TSP commands:

For voltage measurements, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.sense = smu.SENSE_2WIRE
```

To assign a different measure function, replace `smu.FUNC_DC_VOLTAGE` with one of the following:

- For current measurements: `smu.FUNC_DC_CURRENT`
- For resistance measurements: `smu.FUNC_RESISTANCE`

Four-wire remote sense connections

Using 4-wire remote sense connections provides the most accurate low-resistance, voltage source, and measurement accuracy. Specified accuracies for instrument source and measurement capabilities are only guaranteed when you use 4-wire remote sensing.

By default, the 2470 instruments are configured to use 2-wire or local voltage sensing. If you choose to enable 4-wire or remote voltage sensing, it is critical that you establish and maintain the proper Kelvin connections between the corresponding force and sense leads to ensure the proper operation of the instrument and to make accurate voltage measurements. Sense HI must be connected to Force HI, and Sense LO must be connected to Force LO.

When sourcing voltage with remote sense, the instrument relies on the voltage detected with the sense lines to provide the proper closed-loop control of its output voltage. If a sense line becomes disconnected from its corresponding force line, an erroneous voltage is sensed and the output voltage may be adjusted to a level that is radically different than the programmed voltage level (possibly to hazardous levels, depending on the model).

When sourcing current with remote sense, the instrument relies on the voltage detected with the sense lines to properly limit the voltage across the device-under-test. If a sense line becomes disconnected from its corresponding force line, an erroneous voltage is sensed and the voltage across the device may exceed the programmed source limit voltage, possibly causing damage to the device or test fixture.

In both cases, the voltage is not measured correctly if a sense lead becomes disconnected from its corresponding force lead.

WARNING

Even with the overvoltage protection set to the lowest value (20 V), never touch anything connected to the terminals of the 2470 when the instrument is on. Always assume that a hazardous voltage (greater than 30 V_{RMS}) is present when the instrument is on. To prevent damage to the device under test or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

To prevent unexpected voltages caused by a sense lead that becomes disconnected from its corresponding force lead during remote sense operation, you can enable overvoltage protection (OVP) to restrict the maximum output voltage level of the instrument when either voltage or current is sourced. With OVP enabled, the voltage between the Force HI and the Force LO terminals will not exceed the specified overvoltage limit value regardless of the voltage present on the sense leads. Refer to Overvoltage protection (on page 4-35) for additional information.

To make 4-wire measurements, you must set the sense mode of the instrument to 4-wire, as described in the following topics.

NOTE

When you are sourcing voltage in 4-wire remote sense, connect the sense leads to the DUT. If a sense lead is disconnected, the instrument senses 0 V, which causes it to increase the output voltage to compensate. To further protect against overvoltage situations, you can set overvoltage protection. See Overvoltage protection (on page 4-35) for more information.

Always connect the sense lines as close as possible to the device under test.

Figure 59: 2470 rear-panel 4-wire remote sense connections

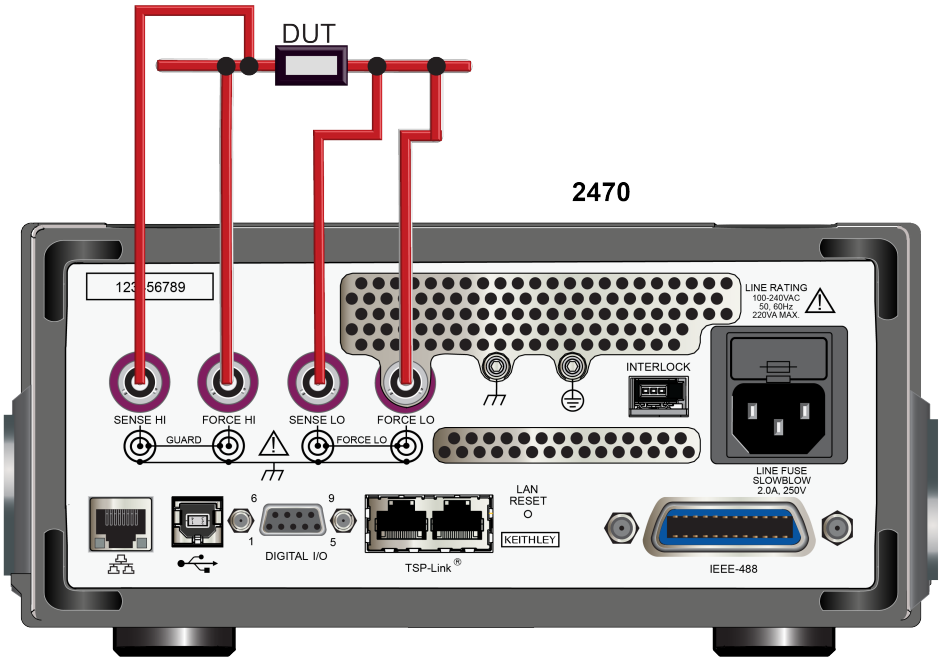
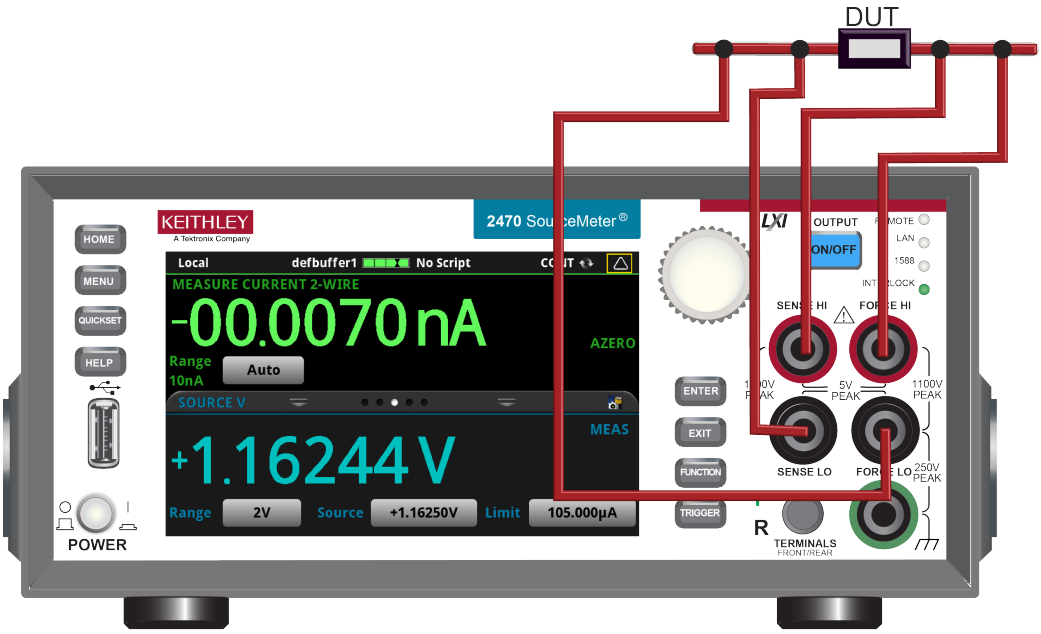


Figure 60: 2470 4-wire sense front-panel connection



Set the instrument to 4-wire sense

To use 4-wire connections, you must set the instrument to 4-wire sense.

When 4-wire sense is selected and the output is turned off, the sense lines are internally disconnected. The sense lines are automatically reconnected when the output is turned on.

NOTE

When you change the sense setting, the output is automatically turned off.

Using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measure combination.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. Set Sense to **4-Wire Sense**.
6. Select **HOME** to return to the operating display.

Using SCPI commands:

To change to 4-wire sensing for current measurements, send the command:

```
:SENSe:CURRent:RSENse ON
```

To change to 4-wire sensing for voltage, replace `CURRent` with `VOLTage`. For resistance, replace `CURRent` with `RESistance`.

Using TSP commands:

To change to 4-wire sensing for voltage measurements, send these commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.sense = smu.SENSE_4WIRE
```

To assign a different measure function, replace `smu.FUNC_DC_VOLTAGE` with one of the following:

- For current measurements: `smu.FUNC_DC_CURRENT`
- For resistance measurements: `smu.FUNC_RESISTANCE`

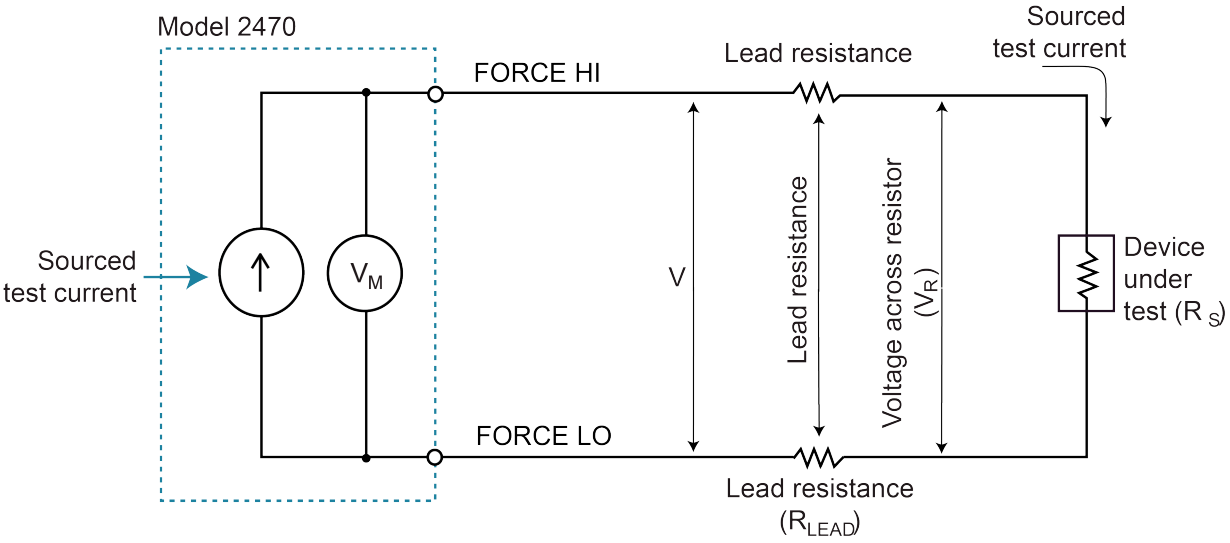
Ohms measurements

You can make ohms measurements using either 2-wire or 4-wire sensing.

Accuracy of 2-wire resistance measurements

The 2-wire sensing method has the advantage of requiring only two test leads. However, as shown in the following figure, the total lead resistance is added to the measurement. This can seriously affect the accuracy of 2-wire resistance measurements, particularly with low resistance values.

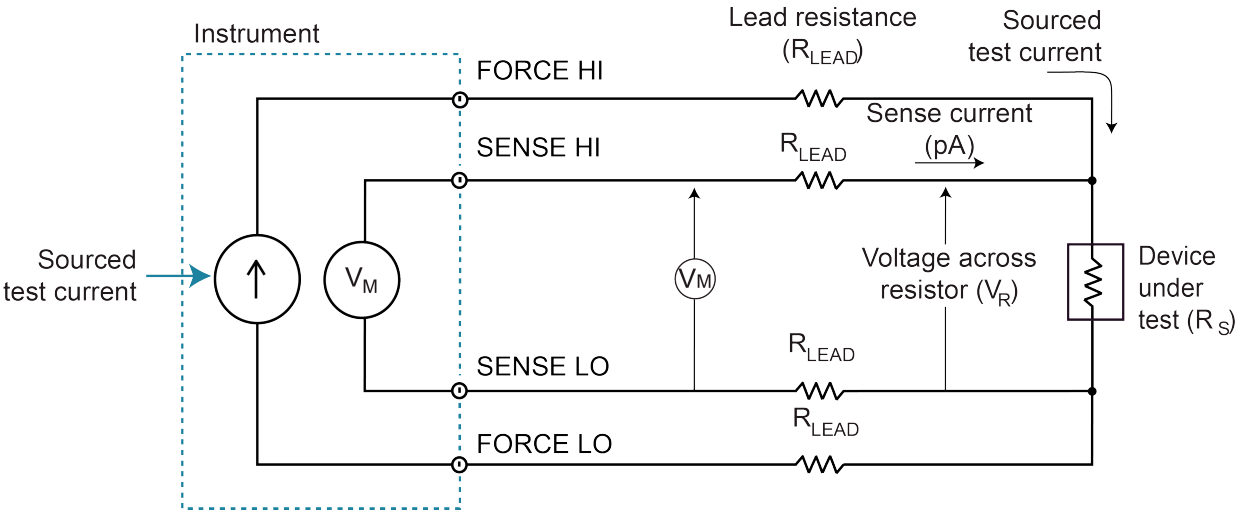
Figure 61: Two-wire resistance sensing for a high-impedance DUT



Minimizing the effect of lead resistance with 4-wire testing

The 4-wire sensing method, shown in the following figure, minimizes or eliminates the effects of lead resistance. The effects of lead resistance are minimized by measuring the voltage across the resistor under test with a second set of test leads. The current through the sense leads is negligible, and the measured voltage is essentially the same as the voltage across the resistor under test. The voltage-sensing leads should be connected as close to the resistor under test as possible to avoid including the resistance of the test leads in the measurement.

Figure 62: 2470 4-wire resistance sensing



Sense current is negligible, therefore $V_M = V_R$

Measure resistance is $\frac{V_M}{I} = \frac{V_R}{I} = R_s$

Test fixtures

A test fixture can be used to house a device or test circuit. The test fixture can be a metal or nonconductive enclosure and is typically equipped with a lid. When the test fixture is correctly connected using the interlock, the output of the 2470 will be less than ±42 V_{PEAK} when the lid of the test fixture is opened.

You mount the test circuit inside the test fixture.

WARNING

To provide protection from shock hazards, an enclosure should be provided that surrounds all live parts.

Nonconductive enclosures must be constructed of materials that are suitably rated for flammability and the voltage and temperature requirements of the test circuit. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

For metallic enclosures, the test fixture chassis must be properly connected to protective earth (safety ground). A grounding wire (16 AWG or larger) must be attached securely to the test fixture at a screw terminal designed for safety grounding. The other end of the ground wire must be attached to a known protective earth (safety ground).

When hazardous voltages (>30 V_{RMS}, 42 V_{PEAK}) will be present, the test fixture must meet the following safety requirements:

- **Construction material**: A metal test fixture must be connected to a known protective earth (safety ground) as described in the above warning. A nonconductive test fixture must be constructed of materials that are suitable for flammability, voltage, and temperature conditions that may exist in the test circuit. The construction requirements for a nonconductive enclosure are also described in the warning above.
- **Test circuit isolation**: With the lid closed, the test fixture must completely surround the test circuit. A metal test fixture must be electrically isolated from the test circuit. Although the outer layer on a high voltage triaxial cable must be connected to the test fixture's metal chassis, the inner two layers of the cable (input/output connectors) must be isolated from the test fixture. Internally, Teflon standoffs are typically used to insulate the internal printed circuit board or guard plate for the test circuit from a metal test fixture.
- **Interlock switch**: The test fixture must have a normally-open interlock switch. The interlock switch must be installed so that when the lid of the test fixture is opened, the switch will open, and when the lid is closed, the switch will close. The 2470 includes an interlock connector on the rear panel of the instrument. When properly connected to a test fixture, the output of the 2470 is limited to ±42 V when the lid of the test fixture is open.

Output-off state

CAUTION

Carefully consider and configure the appropriate output-off state, source, and limits before connecting the 2470 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

When the source of the instrument is turned off, it may not completely isolate the instrument from the external circuit. You can use the Output Off setting to place the 2470 in a known, noninteractive state during idle periods, such as when you are changing the device under test. The output-off states that can be selected for a 2470 are normal, high-impedance, zero, or guard.

When you change the output-off state, the selected output-off state is set immediately. When the instrument is powered on, the instrument is momentarily in the high-impedance output-off state before going to the default output-off state of normal or a setting defined in a power up script.

Regardless of the selected output-off state, if an overtemperature condition occurs, the instrument goes into the high-impedance output-off state.

When the output is off, the SOURCE area of the home screen shows the source value that is set, not the value that is presently being output.

Normal output-off state

When the 2470 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10% of the full scale of the present measurement function autorange value
- If source readback is off, Output Off is displayed in the home screen Source area
- If source readback is on, the actual measurement is displayed in the home screen Source area
- If measurement is set to resistance, dashes (--.----) are shown in the home screen Source area
- The Source button on the home screen shows the value that will be sourced when the output is turned on again

Even though the voltage source is set to zero, the source value may not be exactly at zero and the instrument may source or sink a small amount of power. In most cases, this source or sink power level is insignificant. For passive devices such as resistors, normal mode should be sufficient to protect your device. However, because the limit current is 10% of range, it may take time to remove charge from large energy storage devices such as capacitors.

For sources, such as batteries, the normal output-off state limits the current to 10% of range. This may be acceptable for devices such as another source-measure instrument or a power supply. However, for devices such as small cell batteries, it can drain the batteries. In situations such as this, use the high-impedance output-off state.

High-impedance output-off state

When the high-impedance output-off state is selected and the output is turned off:

- The measurement sense is set to 2-wire
- The output relay opens, disconnecting the instrument as a load

Opening the relay disconnects external circuitry from the inputs and outputs of the instrument. To prevent excessive wear on the output relay, do not use this output-off state for tests that turn the output off and on frequently.

The high-impedance output-off state should be used when the instrument is connected to a power source or another source-measure instrument. In some cases, it may also be appropriate for devices such as capacitors.

When the output is turned on again, the relay has a settling time of approximately 15 ms.

When the high-impedance output-off state is selected, you cannot make measurements using 2-wire connections.

Zero output-off state

When the zero output-off state is selected and you turn off the output:

- The measurement sense is changed to 2-wire
- The voltage source is selected and set to 0 V
- The range is set to the presently selected range (turn off autorange)
- If the source is voltage, the current limit is not changed
- If the source is current, the current limit is set to the programmed source current value or to 10% full scale of the present current range, whichever is greater

When the zero output-off state is selected, you can use the instrument as an ammeter because it is outputting 0 V.

The zero mode is ideal for passive devices such as resistors. In most cases, it can also be used with energy storage devices such as capacitors and inductors. This mode will discharge capacitors under test and remove the charge from semiconductor junctions.

Guard output-off state

Use the guard output-off state when you are measuring a load that uses an active source.

When the guard output-off state is selected and the output is turned off, the following actions occur:

- The measurement sense is changed to 2-wire
- The current source is selected and set to 0 A if the source is set to current (amps); otherwise, the output remains a voltage source when the output is turned off
- The voltage limit is set to 10% full scale of the present voltage range

Output-off states and inductive loads

To protect the instrument from inductive energy, you may need to install a spark gap across the HI and LO terminals. The instrument does not have internal spark gap protection.

Setting the output-off state

Before setting the output-off state, set the source function. The output-off state is stored with the source function. If you change the source function, the output-off state changes to the last state you set for that function.

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Set **Output Off** to the appropriate setting for your application.
4. Select **HOME** to return to the operating display.

Using SCPI commands:

To set the output-off state to normal, send the command:

```
:OUTPut:SMODe NORMal
```

To set the output-off state to zero, send the command:

```
:OUTPut:SMODe ZERO
```

To set the output-off state to high impedance, send the command:

```
:OUTPut:SMODe HIMPedance
```

To set the output-off state to guard, send the command:

```
:OUTPut:SMODe GUARd
```

Using TSP commands:

To set the output-off state to normal, send the command:

```
smu.source.offmode = smu.OFFMODE_NORMAL
```

To set the output-off state to zero, send the command:

```
smu.source.offmode = smu.OFFMODE_ZERO
```

To set the output-off state to high impedance, send the command:

```
smu.source.offmode = smu.OFFMODE_HIGHZ
```

To set the output-off state to guard, send the command:

```
smu.source.offmode = smu.OFFMODE_GUARD
```

Source-measure overview

Using the 2470, you can perform the following operations:

- Source voltage and measure current, voltage, resistance, or power
- Source current and measure voltage, current, resistance, or power
- Measure voltage, current, resistance, or power

NOTE

Make sure you select functions before you make changes to other instrument settings. The options that you have for settings depend on the functions that are active when you make the changes. If you make a change that is not compatible with the active functions, you may get unexpected results or you may receive an event message. Also note that when you select a different function, the instrument clears the buffer.

When you are making measurements, you can set the measurement range to a specific range or to automatic ranging. If you select a specific range, for the best accuracy, use the lowest possible range. When Auto is selected, the instrument selects the most sensitive range to make a measurement or will change automatically as the measured value of the device changes.

The operating boundaries for the source-measure operations are provided in Operating boundaries (on page 5-4).

WARNING

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the 2470 while the instrument is powered on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the 2470 before handling cables. Putting the equipment into an output-off state does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Source and measure order

When you are using a remote interface, you should set the measure function first, then set the source function, because setting the measure function may change the source function. This is not necessary when you use the front-panel FUNCTION key to set the source and measure functions.

Once you have set the source and measure functions, you can change other measure and source settings as needed.

When setting range, you should first set the limit (compliance) to a value higher than the measure range you intend to set.

Source and measure through the front panel

You can source and measure through the options available on the front panel.

Using Quick Setups

Quick Setups allow you to select predefined setups. The options include measurement choices that make measurements while sourcing a 0 value. The power supply option sources a voltage without making a measurement.

Quick Setup options that are available include Voltmeter, Ammeter, Ohmmeter, and Power Supply.

For detail on the values that are set when you select these options, see QuickSet menu (on page 3-18).

CAUTION

When you select a Quick Setup, the instrument turns the output on. Carefully consider and configure the appropriate output-off state, source, and limits before connecting the 2470 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

Making a measurement with the QuickSet functions

The measurement-only functions available through the QuickSet option include Voltmeter, Ammeter, Ohmmeter, and Power Supply.

Using the front panel:

1. Make connections to the device under test before running the Quick Setup. The Voltmeter and Ammeter options use 2-wire connections. For the Ohmmeter option, you can select 2-wire or 4-wire connections. For information about making connections, see Test connections (on page 4-1).
2. Press **QUICKSET**.
3. Under Quick Setups, select the type of measurement.
4. **Ohmmeter only:** Select 2-wire or 4-wire sense.
5. A message may be displayed. If the conditions of your test setup permit it, select **OK**.
6. The output turns on and the instrument begins making measurements.
7. Observe the readings.
8. You can adjust the source and measure settings while the instrument makes measurements.
9. When finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Sourcing a voltage with the QuickSet functions

The source-only function available through the QuickSet option is named Power Supply.

Using the front panel:

1. Make connections to the device under test before running the Quick Setup. You can select 2-wire or 4-wire connections. For information about making connections, see Test connections (on page 4-1).
2. Press **QUICKSET**.
3. Under Quick Setups, select **Power Supply**.
4. Select the voltage level to be sourced. Select **OK**.
5. Select the maximum allowed source current. Select **OK**.
6. Select 2-wire or 4-wire sense.
7. The output turns on and the instrument begins sourcing voltage.
8. You can adjust the source and measure settings while the instrument makes measurements.
9. When finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Using the Performance slider

Use the Performance slider to adjust for performance (resolution versus speed).

When you adjust the Performance slider, the instrument changes settings based on where you position the slider. As you increase reading speed, you lower the amount of resolution. As you increase resolution, you decrease the speed. These settings take effect the next time the output is turned on and measurements are made.

NOTE

To see which settings are adjusted, you can set the Command setting of the Event Log to On. When command logging is on, each setting made by the Performance slider is listed as an Information event in the Event Log.

The settings that the instrument adjusts include autozero, autodelay and filter settings, display digits, and integration rate.

Source voltage and make measurements

When the 2470 is sourcing voltage, you can make current, voltage, resistance, or power measurements.

Using the front panel:

1. Connect the device under test (DUT) as described in Test connections (on page 4-1).
2. Set the function for your measurement. Press **FUNCTION**. Under Source Voltage and Measure, select the type of measurement you want to make.
3. Select the source voltage range. Under SOURCE V on the home screen, set the **Range**..
4. Set the source voltage. Under SOURCE V on the home screen, select **Source**. Use the displayed number pad to set the value. Select **OK**.
5. Set current limits for the source. Under SOURCE V on the home screen, set the **Limit**.
6. Select the measurement range. In the MEASURE area of the home screen, set the **Range**.
7. Turn on the output by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns on.
8. Hold the **TRIGGER** key for three seconds to change the measurement method. Select one of the following options:
   - **Continuous Measurement:** The instrument makes continuous measurements.
   - **Manual Trigger Mode:** The instrument makes measurements when you press the TRIGGER key.
9. Observe the readings.
10. You can adjust the settings while the instrument makes measurements.
11. When you are finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Source current and make measurements

When the 2470 is sourcing current, you can make current, voltage, resistance, or power measurements.

Using the front panel:

1. Connect the device under test (DUT) as described in Test connections (on page 4-1).
2. Set the function for your measurement. Press **FUNCTION**. Under Source Current and Measure, select the type of measurement you want to make.
3. Select the source current range. Under SOURCE I on the home screen, set the **Range**.
4. Set the source current level. Under SOURCE I on the home screen, select **Source**. Use the displayed number pad to set the value. Select **OK**.
5. Set voltage limits for the source. Under SOURCE I on the home screen, set the **Limit**.
6. Select the measurement range. In the MEASURE area of the home screen, set the **Range**.
7. Turn on the output by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns on.
8. Hold the **TRIGGER** key for three seconds to change the measurement method. Select one of the following options:
   - **Continuous Measurement:** The instrument makes continuous measurements.
   - **Manual Trigger Mode:** The instrument makes measurements when you press the TRIGGER key.
9. Observe the readings.
10. When you are finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Source values

When you edit the source value, the source is updated immediately. This allows you to adjust the source value while the output is on.

In certain situations, you cannot change the source value immediately. These situations include:

- While the instrument is performing a sweep
- If offset compensation is enabled (ohms measurements only)
- If the Quick Setup Ohmmeter option is enabled

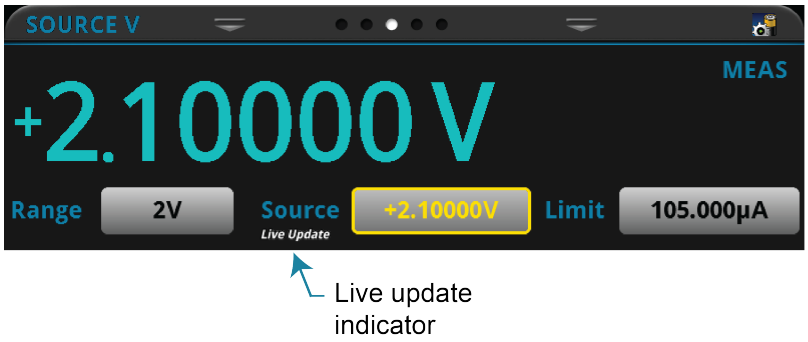
Adjusting source and limit values using live update

When you enable live update on the instrument, you can update source range and source limit values instantly using the navigation control.

To enter live update:

1. Press the **HOME** key.
2. Use the navigation control to select Source or Limit.
3. Hold down the **ENTER** key for a few seconds to enable live update. The instrument displays the Live Update indicator below the value you selected to update.
4. You can turn live update mode on for both Source and Limit at the same time, as shown in the following figure, by repeating steps 1 through 3. Yellow indicates the value that you selected to update.

Figure 63: Live update



5. Select the value you want to update.
6. Press the navigation control to access the value. The instrument indicates each selected character by underlining it.
7. Turn the navigation control to select the character you want to update. As you scroll, the underline moves from one character to the next.
8. Press the navigation control to enable updating.
9. Turn the navigation control to increase or decrease the value of the underlined character. Press the navigation control when the value is correct.
10. Repeat steps 7 through 9 for each character you want to update.

NOTE

As you change each character, the updates happen in real time if the output is on (press the **OUTPUT ON/OFF** switch to turn the source on). For example, if you want to change from 15 V to 7 V, after changing the 5 to a 7, the instrument outputs 17 V until you change the 1 to a 0.

11. Press the navigation control twice to leave the edit mode.

To exit live update mode:

Hold down the **ENTER** key for a few seconds to exit live update.

Store settings for functions regardless of active state

When you are using the front panel or TSP commands, changes to settings affect the function that is presently selected.

If you need to set up functions that are not selected, you can use the `smu.measure.setattribute` command. This command applies settings to a specific function, whether or not the function is selected. If you are changing functions during a test and want to improve the speed of the test, this eliminates the time needed to change the settings for each function during the test.

For example, the following set of commands sets up the Current function. When you select Measure by to be Current, these settings are active immediately.

```
-- Active measure function is DC Voltage.
-- Configure DC Current settings without changing the active function.
smu.measure.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_MEAS_RANGE, 35e-6)
smu.measure.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_MEAS_DIGITS, smu.DIGITS_5_5)
smu.measure.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_MEAS_NPLC, 0.5)
```

Making resistance measurements

When you make resistance measurements, the resistance is calculated by either sourcing current and measuring voltage or by sourcing voltage and measuring current.

When source readback is on, the instrument measures both voltage and current and uses these values in the ohms calculations. When source readback is off, the instrument uses the programmed source value and the measured value in the ohms calculations. Note that the measured source value is more accurate than the programmed source value, so measurements made with source readback on are more accurate.

When you are measuring resistance, you can set the offset-compensated ohms option.

Resistance measurement methods

From the front panel, you can use one of the following methods to measure resistance with the 2470:

- Press **FUNCTION** and select source current and measure resistance.
- Press **FUNCTION** and select source voltage and measure resistance.
- Press **QUICKSET** and select **Ohmmeter**. When Ohmmeter is selected, the source current and source limit are set automatically.

From a remote interface, you can use one of following methods to measure resistance with the 2470:

- Source voltage, measure current, and set measure units to ohms.
- Source current, measure voltage, and set measure units to ohms.
- Set the measure function to resistance. When the measure function is set to resistance, the instrument sets the source current and source limit automatically.

Each of these methods is described in the following topics.

Source voltage, measure current, and read ohms

If you want to make resistance readings by sourcing voltage and measuring current, you can use this method.

The examples below use a 100 kΩ device under test. The code:

- Makes five readings by sourcing 5 V
- Measures current with autorange enabled
- Sets the measure units to ohms
- Uses offset compensation
- Retrieves the source and measure values

Even though the measurement units are in ohms, the measurement range is 10 µA.

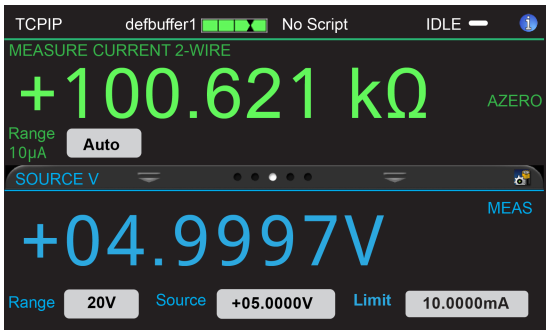
Using SCPI:

Send the following code:

```
*RST
SENSe:FUNCtion "CURR"
SENSe:CURRent:RANGe:AUTO ON
SENSe:CURRent:UNIT OHM
SENSe:CURRent:OCOM ON
SOURce:FUNCtion VOLT
SOURce:VOLT 5
SOURce:VOLT:ILIM 0.01
SENSe:COUNT 5
OUTPut ON
TRACe:TRIGger "defbuffer1"
TRACe:DATA? 1, 5, "defbuffer1", SOUR, READ
OUTPut OFF
```

The front-panel display will look similar to the following example.

Figure 64: Resistance measurement using SVMI and reading ohms



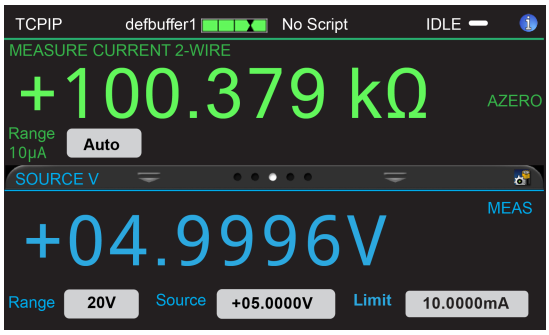
Using TSP commands:

Send the following code:

```
reset()
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.ON
smu.measure.unit = smu.UNIT_OHM
smu.measure.count = 5
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.level = 5
smu.source.ilimit.level = 0.01
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
for i = 1, defbuffer1.n do
   print(defbuffer1.relativetimestamps[i], defbuffer1[i])
end
smu.source.output = smu.OFF
```

The front-panel display will look similar to the following example.

Figure 65: Resistance measurement SVMI and reading ohms



Source current, measure voltage, and set measure units to ohms

If you want to make resistance readings by sourcing current and measuring voltage, you can use this method.

The examples below use a 100 kΩ device under test. The code:

- Makes five readings by sourcing 5e-6 A
- Measures voltage with autorange enabled
- Sets the measure units to ohms
- Uses offset compensation
- Retrieves the source and measure values

Even though the measurement units are in ohms, the measurement range is 2 V.

Using SCPI:

Send the following code:

```
*RST
SENSe:FUNCtion "VOLT"
SENSe:VOLTage:RANGe:AUTO ON
SENSe:VOLTage:UNIT OHM
SENSe:VOLTage:OCOM ON
SOURce:FUNCtion CURR
SOURce:CURRent 5e-6
SOURce:CURRent:VLIM 10
SENSe:COUNT 5
OUTPut ON
TRACe:TRIGger "defbuffer1"
TRACe:DATA? 1, 5, "defbuffer1", SOUR, READ
OUTPut OFF
```

The front-panel display will look similar to the following example.

Figure 66: Resistance measurement SIMV SCPI example



Using TSP commands:

Send the following code:

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
smu.measure.unit = smu.UNIT_OHM
smu.measure.count = 5
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.level = 5e-6
smu.source.vlimit.level = 10
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
for i = 1, defbuffer1.n do
   print(defbuffer1.relativetimestamps[i], defbuffer1[i])
end
smu.source.output = smu.OFF
```

The front-panel display will look similar to the following example.

Figure 67: Resistance measurement SIMV and read ohms



Measure resistance using the resistance function

When the measurement function is set to resistance, the 2470 measures resistances by sourcing current. The instrument automatically sets the magnitude of the current source, voltage limit, and the measure range.

This mode is the same as the Ohmmeter Quick Setup, which is available by pressing the QUICKSET key.

The examples below use a 100 kΩ device under test. The code makes five readings. Note that the measurement range is 200 kΩ.

Using SCPI:

Send the following code:

```
*RST
SENSe:FUNCtion "RES"
SENSe:RESistance:RANGe:AUTO ON
SENSe:RESistance:OCOMpensated ON
SENSe:COUNt 5
OUTPut ON
TRACe:TRIGger "defbuffer1"
TRACe:DATA? 1, 5, "defbuffer1", SOUR, READ
OUTPut OFF
```

The front-panel display will look similar to the following example.

Figure 68: Resistance measurement using resistance function



Using TSP:

Send the following code:

```
reset()
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.autorange = smu.ON
smu.measure.count = 5
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
for i = 1, defbuffer1.n do
   print(defbuffer1.relativetimestamps[i], defbuffer1[i])
end
smu.source.output = smu.OFF
```

The front-panel display will look similar to the following example.

Figure 69: Resistance measurement with automatic settings



Offset-compensated ohms

The voltage offsets caused by the presence of thermoelectric EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

See Offset-compensated ohm calculations (on page 5-22) for additional detail on calculating offset-compensated ohms.

Setting offset-compensated ohms

Using the front panel:

NOTE

This setting is only available from the front panel when the resistance measurement function is selected.

1. Press the **MENU** key.
2. Under Measure, select **Settings**.
3. Set Offset Compensation to **On**.
4. Select **HOME** to return to the operating display.

Using SCPI commands:

To enable offset-compensated ohms, send the command:

```
SENSe:RESistance:OCOMpensated ON
```

Using TSP commands:

To enable offset-compensated ohms, send the commands:

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.offsetcompensation = smu.ON
```

Source and measure using SCPI commands

The SCPI commands that set up the source functions are in the SOURce subsystem.

The source commands are specific to each source function (voltage or current). For example, to set the range to 100 mA for the current function, you would send:

```
:SOURce:FUNCtion CURRent
:SOURce:CURRent:RANGe 1e-01
```

To set the range to 20 V for the voltage function, you would send:

```
:SOURce:FUNCtion VOLTage
:SOURce:VOLTage:RANGe 20
```

The SCPI commands that set up the measurement functions are in the SENSe subsystem.

The sense commands are also specific to each measure function (voltage, current, or resistance). For example, to set the NPLC cycles to 0.5 for the current measurement function, you send:

```
:SENSe:CURRent:NPLCycles 0.5
```

For the voltage measurement function, you send:

```
:SENSe:VOLTage:NPLCycles 0.5
```

For the resistance measurement function, you send:

```
:SENSe:RESistance:NPLCycles 0.5
```

To make a measurement, you send the `MEASure:<function>?` command. For example, to make a current measurement, send the command:

```
:MEASure:CURRent?
```

To make a voltage measurement, send the command:

```
:MEASure:VOLTage?
```

To make a resistance measurement, send the command:

```
:MEASure:RESistance?
```

For detailed application examples that use the SCPI command set, see the *Model 2470 User's Manual*.

Command descriptions are provided in the SCPI command reference (on page 12-1).

Source and measure using TSP commands

The TSP commands that set up the source functions begin with `smu.source`.

The source commands are specific to each source function (voltage or current). For example, to set the range to 100 mA for the current source function, you would send:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 0.1
```

To set the range to 20 V for the voltage function, you would send:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
```

The TSP commands that set up the measurement functions begin with `smu.measure`.

The sense commands are also specific to each measure function (voltage, current, or resistance). For example, to set the NPLC cycles to 0.5 for the current measurement function, you send:

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.nplc = 0.5
```

For the voltage measurement function, you send:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.measure.nplc = 0.5
```

For the resistance measurement function, you send:

```
smu.source.func = smu.FUNC_RESISTANCE
smu.measure.nplc = 0.5
```

To make a measurement, you set the measurement function and then send the `smu.measure.read()` command. For example, to make a current measurement, send the commands:

```
smu.measure.func = smu.FUNC_CURRENT
print(smu.measure.read())
```

To make a voltage measurement, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
print(smu.measure.read())
```

To make a resistance measurement, send the commands:

```
smu.measure.func = smu.FUNC_RESISTANCE
print(smu.measure.read())
```

For detailed application examples that use the TSP command language, see the *Model 2470 User's Manual*.

Command descriptions are provided in the TSP command reference (on page 14-1).

Protection

The 2470 provides several methods for ensuring that the source remains within certain values. This helps to protect the device under test (DUT) from damage.

The protections that affect the source are the:

- **Overvoltage protection.** This is the voltage at the instrument terminals.
- **Source limits.** This is the sourced value at the device.

Overvoltage protection

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced. This protects the device under test (DUT) from high voltage levels.

For example, if a sense lead is disconnected or broken during a 4-wire sense measurement, the instrument can interpret the missing sense lead as a decrease in voltage and respond by increasing the source output. If overvoltage protection is set, the sourced output is not allowed to exceed the overvoltage protection limit.

The value set for overvoltage protection takes precedence over the source limit settings. When it is enabled, it is always in effect.

When overvoltage protection is set and the sourced voltage exceeds the setting:

- The output is clamped at the overvoltage protection value
- On the front panel, an indicator to the right of the voltage displays OVP

When overvoltage protection is used in a test sequence, it should be set before turning the source on.

NOTE

Even when overvoltage protection is set to None, overvoltage protection is active to 1165 V. If the instrument indicates that OVP is active, check the sense connections and setttings to make sure the connections and settings are correct and that there are no broken leads.

WARNING

Even with the overvoltage protection set to the lowest value (20 V), never touch anything connected to the terminals of the 2470 when the instrument is on. Always assume that a hazardous voltage (greater than 30 V_{RMS}) is present when the instrument is on. To prevent damage to the device under test or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

Setting overvoltage protection levels

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Select **Overvoltage Protection Limit**.
4. Select the limit.
5. Select **HOME** to return to the operating display.

Using SCPI commands:

Send the `:SOURce:VOLTage:PROTection` command with the value of the limit. For example, to set the overvoltage limit for the voltage source to 20 V, send the command:

```
SOURce:VOLTage:PROTection PROT20
```

See the command description for :SOURce[1]:<function>:PROTection[:LEVel] (on page 12-81) for the full list of options.

Using TSP commands:

Set the source function and send the `smu.source.protect.level` command with the value of the limit. For example, to set the overvoltage limit to 20 V for the voltage source function, send the commands:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.protect.level = smu.PROTECT_20V
```

See the command description for smu.source.protect.level (on page 14-186) for the full list of options.

Source limits

The source limits (also known as compliance) prevent the instrument from sourcing a voltage or current over a set value. This helps prevent damage to the device under test (DUT).

The values that can be set for the limits must be below the setting for the overvoltage protection limit.

This limit can also be restricted by the measurement range. If a specific measurement range is set, the limit must be more than 0.1 percent of the measurement range. If not, an event is generated and the limit is automatically changed to an appropriate value for the selected range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

If you attempt to change the source limit to a value that is not appropriate for the selected source range, the source limit is not changed and a warning is generated. You must change the source range before you can select the new limit.

The lowest allowable limit is based on the load and the source value. For example, if you are sourcing 1 V to a 1 kΩ resistor, the lowest allowable current limit is 1 mA (1 V/1 kΩ = 1 mA). Setting a limit lower than 1 mA limits the source.

The effective source limit is the lesser of either the programmed source limit or 105% of the active measure range. If you use fixed measure ranges, the instrument prevents you from selecting different limit and measure ranges. However, if measure autorange is selected, it is possible for the autorange process to cause the ranges to differ because the instrument may go down to a range that is lower than the one on which the source limit is programmed. This causes the effective source limit to drop to 105% of the newly selected measure range. For example, the output may be limited if you make a measurement that causes the range to be lowered and then increase the source level or make a change to the device under test or an external source. In this situation, the source voltage or current is limited to conform with the lower measurement range until another measurement causes the instrument to select a higher range to accommodate the new source value. If no further measurement is made, the source may remain limited to the present measurement range indefinitely.

To prevent the source output from being limited below the source limit setting, you can enable the autorange rebound feature. When autorange rebound is enabled, after an autoranged measurement is made, the measure range is restored to match the limit range when the autoranged measurement is complete. This ensures that the source does not limit at less than the full limit setting.

For example, assume the following conditions:

- Current limit of 10 mA
- Instrument sources a voltage of 10 V
- DUT resistance of 10 Ω

With a source voltage of 10 V and a DUT resistance of 10 Ω, the current through the DUT should be: 10 V / 10 Ω = 1 A. However, because the limit is set to 10 mA, the current will not exceed 10 mA, and the voltage across the resistance is limited to 100 mV. In effect, the 10 V voltage source is transformed into a 10 mA current source.

In steady-state conditions, the set limit restricts the instrument output unless there are fast transient load conditions.

If the source output exceeds the source limit:

- On the home screen, LIMIT is displayed to the right of the source voltage.
- The Source value changes to yellow.

The source is clamped at the maximum limit value.

For additional details on using limits, see Operating boundaries (on page 5-4).

Setting source limits

Using the front panel:

1. Press **FUNCTION** and select the source and measurement combination.
2. On the home screen, select **Limit**.
3. Set the value.
4. Select **OK**.

Using SCPI commands:

To set the limit when sourcing current, send the command:

```
SOURCe:CURRent:VLIMit <n>
```

Where <n> is the current limit value.

To set the limit when sourcing voltage, send the command:

```
SOURCe:VOLTage:ILIMit <n>
```

Where <n> is the voltage limit value.

Using TSP commands:

To set the limit when sourcing current, send the commands:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.vlimit.level = limitValue
```

To set the limit when sourcing voltage, send the commands:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = limitValue
```

Where *limitValue* is the limit value.

Setting autorange rebound

Using the front panel:

1. Press **FUNCTION** and select the source and measurement combination.
2. Select **MENU**.
3. Under Measure, select **Settings**.
4. Set Auto Range Rebound to **On**.

Using SCPI commands:

To enable autorange rebound when measuring current, send the command:

```
SENSe:CURRent:RANGe:AUTO:REBound ON
```

To enable autorange rebound when measuring voltage, send the command:

```
SENSe:VOLTage:RANGe:AUTO:REBound ON
```

Using TSP commands:

To enable autorange rebound:

```
smu.measure.autorangerebound = smu.ON
```

Ranges

You can set ranges for the source and measurement values. You can set specific ranges or allow the instrument to choose the ranges automatically.

The source range determines how accurately the source output can be set.

The measurement range determines the full-scale input for the measurement. The measurement range also affects the accuracy of the measurements and the maximum signal that can be measured.

The highest available range is determined by the limit setting for the function that is being sourced or measured.

The range defaults to autorange for all functions. If you switch from a fixed range to autorange, autorange is set to off. The range remains at the fixed range until a measurement is made or source level is changed, at which time the range is set to accommodate the new measurement or source level.

Source range

For most applications, you will select the automatic source range. This causes the instrument to set the source range to the best range for the present settings.

In most cases, you select a specific source range to verify that a sweep uses the same range to source all the sweep points or to reduce test time. Selecting a specific range eliminates the time that the instrument needs to select the range automatically.

If you set the source range manually through either the front panel or a remote command, the setting for automatic source range is set to disabled.

The selected source level must be lower than the overvoltage protection limit. When the overvoltage protection level is exceeded, OVP is displayed in the SOURCE area of the home screen.

Selecting a specific source range

To select the range, specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 3 V, set the source level to 3. The 2470 selects the 20 V range.

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 200 mV, 2 V, 20 V, 200 V, and 1000 V.

From the front panel:

1. Press the **HOME** key.
2. Under SOURCE, select **Range**. The Select Range dialog box is displayed.
3. Select the range.

Over a remote interface:

SCPI commands: Refer to :SOURce[1]:<function>:RANGe (on page 12-82).

TSP commands: Refer to smu.source.range (on page 14-187).

Selecting the automatic source range

When the automatic range is selected, the source-measure cycle is repeated to determine the correct range. This means that any source delay is applied each time the instrument has to set the automatic range.

Using the front panel:

1. Press the **HOME** key.
2. Under SOURCE, select **Range**. The Select Range dialog box is displayed.
3. Select **Auto**.

Over a remote interface:

SCPI commands: Refer to :SOURce[1]:<function>:RANGe:AUTO (on page 12-84).

TSP commands: Refer to smu.source.autorange (on page 14-170).

Measurement range

The measure range determines the full-scale measurement span that is applied to the signal. Therefore, it affects both the accuracy of the measurements and the maximum signal that can be measured.

A change to the measure range can cause a change to the related current or voltage limit. When this occurs, an event message is generated.

The combination of voltage and current ranges cannot exceed the power limit of the instrument. For example, with a 200 V source range, the highest current measure range is 100 mA (200 V * 100 mA = 20 W). Refer to Operating boundaries (on page 5-4) for other ranges.

Whether or not you can select a measure range is affected by other settings on the instrument. You can only select a measure range if you are sourcing one type of measurement and measuring another. For example, you can select a measure range if you are sourcing voltage and measuring current. However, if you are sourcing voltage and measuring voltage, the measure range is the same as the source range and cannot be changed.

For information on how source limits and automatic measurement ranges interact, refer to Source limits.

You can only select a measure range for ohms if you are using the instrument as an ohmmeter.

NOTE

You need to set the measure function before you can set the measure range.

Maximum limits for 2470

Measure range	Maximum source limit
200 mV	±210 mV
2 V	±2.1 V
20 V	±21 V
200 V	±210 V
1000 V	±1100 V
10 nA	±10.5 nA
100 nA	±105 nA
1 µA	±1.05 µA
10 µA	±10.5 µA
100 µA	±105 µA
1 mA	±1.05 mA
10 mA	±10.5 mA
100 mA	±105 mA
1 A	±1.05 A

Selecting a specific measure range

When selecting a measure range, to ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow event.

NOTE

If you set the measure range to a specific value, the setting for automatic measure range is set to disabled.

Selecting a specific measure range

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. On the home screen, select **Range** in the measurement view area. The Measure Range dialog box is displayed.
3. Select the range. The selected value is displayed.

If the instrument displays an overflow message, select a higher range.

Using a remote interface:

- SCPI commands: Refer to [:SENSe[1]]:<function>:RANGe[:UPPer] (on page 12-55).
- TSP commands: Refer to smu.measure.range (on page 14-158).

Selecting the automatic measurement range

When automatic measure range is selected, the instrument automatically selects the best range to measure the signal. When the automatic range is selected, the source-measure cycle is repeated to determine the correct range. This means that any source delay is applied each time the instrument has to set the automatic range. For example, if you program a one-second source delay, the instrument could take two or more seconds to complete a reading if it must change ranges.

The instrument changes ranges as follows:

1. If the reading reaches 100% of the present range, the instrument goes up three ranges or to the highest range possible.
2. The instrument makes another reading.
3. The instrument uses this reading to determine whether it needs to continue going up in range or if it picks the range based on the reading. If the reading is less than 10%, 1%, or 0.1% of the present range, it will go down by 1, 2, or 3 ranges, respectively.

If you enable the automatic measurement range, the measurement range is changed when a measurement is made. To read the measurement range that the instrument chose, you must query the range after a measurement is made.

For current and voltage measurements, the upper limit is controlled by the current or voltage limit.

For resistance measurements, you can set the upper limit to be used when automatic range selection is enabled to put an upper bound on the range that is used for resistance measurements.

If you set the low limits for the measurement when the automatic range is disabled, the limits are not used until the limit is changed. When changing the lower limit, if the measure range is lower than the limit, the instrument goes up to the limit range.

NOTE

If you set the measure range manually for a function, automatic measure range is automatically turned off for that function and remains off until you re-enable it.

Using the front panel:

1. Press the **FUNCTION** key and select the function.
2. On the measure area of the home screen, select **Range**. The Measure Range dialog box is displayed.
3. Select **Auto**. The actual range is displayed to the left of the button.

Using a remote interface:

- SCPI commands: Refer to [:SENSe[1]]:<function>:RANGe:AUTO (on page 12-51).
- TSP commands: Refer to smu.measure.autorange (on page 14-122).

Selecting low limits when automatic measurement range is used

You can set the low limit for the measurement range that is selected when the measurement range is set automatically.

Choose the lower limits for the automatic measurement range using the front panel

Using the front panel:

1. Press **FUNCTION** and select the measurement function.
2. Press **MENU**.
3. Under Measure, select **Settings**. The MEASURE SETTINGS screen is displayed.
4. Set **Auto Range Low Limit** to an appropriate value. The MEASURE SETTINGS screen is displayed again.
5. Press **HOME** to return to the home screen.

Using a remote interface:

SCPI commands: Refer to [:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 12-52).

TSP commands: Refer to smu.measure.autorangelow (on page 14-124).

Determining upper limits when automatic measurement range is used

For resistance measurements, you can define the upper limit that can be selected when the measurement range is set automatically. For voltage and current measurements, you can retrieve the value of the upper limit.

Using a remote interface:

SCPI commands: Refer to [:SENSe[1]]:<function>:RANGe:AUTO:ULIMit (on page 12-54).

TSP commands: Refer to smu.measure.autorangehigh (on page 14-123).

Automatic reference measurements

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The 2470 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks the reference measurements after every signal measurement. This can cause some measurements to take longer than normal. Source readback is considered to be part of a measurement, so only one reference measurement is made for the measure and source readback measurements.

This additional time can cause problems in sweeps and other test sequences in which measurement timing is critical. To avoid the time that is needed for the reference measurements, you can disable the automatic reference measurements.

When automatic reference measurements are turned off, the instrument may gradually drift out of specification. To prevent inaccurate readings, you can use autozero once to update the autozero information.

Setting autozero

You can enable or disable automatic referencing. You can also request a one-time refresh of the reference values.

The reference setting is stored with the selected measure function.

To set autozero using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measure combination.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. For Auto Zero, select **On** or **Off**.
6. If Off is selected, you can select the **Once** option to send a one-time refresh.
7. Select **HOME** to return to the operating display.

To set autozero using SCPI commands, refer to the following commands:

- [:SENSe[1]]:<function>:AZERo[:STATe] (on page 12-47)
- [:SENSe[1]]:AZERo:ONCE (on page 12-60)

To set autozero using TSP commands, refer to the following commands:

- smu.measure.autozero.enable (on page 14-126)
- smu.measure.autozero.once() (on page 14-127)

Source readback

You can set the instrument to record and display the voltage or current of the configured source value or the actual source value. When you use the configured source value, the instrument records and displays the value that was configured. When you use the actual source value, the instrument measures the actual source value immediately before making the measurement of the device under test.

Using source readback results in more accurate measurements, at the cost of a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the measurement of the device under test. For example, if you have the source set to 60 V, you see something like `+059.998 V` in the SOURCE VOLTAGE area of the home screen.

When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the measurement of the device under test. For example, if the source is set to 60 V, you see something like `+060.000 V` in the SOURCE VOLTAGE area of the home screen.

Setting source readback

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Set Source Readback to **On** or **Off**.
4. Select **HOME** to return to the operating display.

Using a remote interface:

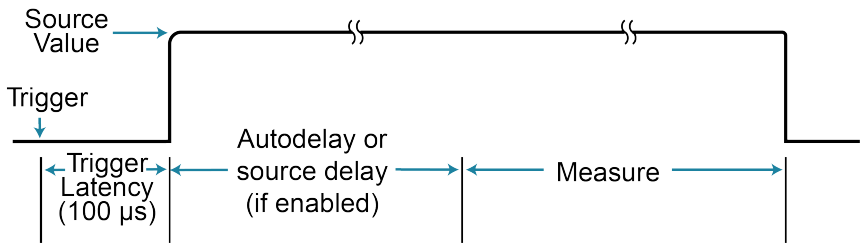
- SCPI commands: Refer to :SOURce[1]:<function>:READ:BACK (on page 12-85).
- TSP commands: Refer to smu.source.readback (on page 14-189).

Source delay

When you use the instrument to source and measure, there is a delay between when the source is turned on and when the measurement is made. The delay provides a settling time for the source.

The amount of delay time depends on the settings that are made for the source delay. You can set a manual value or use the autodelay setting.

Figure 70: 2470 settling and delay times



You can increase the amount of delay time by either setting a fixed amount of time or setting the instrument to include an automatic delay. The delay can be from 0 to 10,000 seconds.

If you select an automatic source delay, the delay time depends on the selected current range. Values for the delay times for each range are shown in the following table. The delay times shown in the table are with source readback off and autozero off.

Current range	Voltage source autodelay (ms)	With high capacitance (ms)	Current source autodelay (ms)	With high capacitance (ms)
10 nA	150	300	150	300
100 nA	100	200	100	200
1 µA	3	20	3	20
10 µA	2	10	2	10
100 µA	1	10	1	10
1 mA	1	10	1	10
10 mA	1	5	1	5
100 mA	1	5	1	5
1 A	1	5	2	5

When either a source delay or autodelay is set, the delay is applied to the first source output and then only when the magnitude of the source changes.

For high impedance and high capacitive loads, more settling time is required for the source. The actual delay period you need can be calculated or determined by trial and error. For purely resistive loads and at higher current levels, the programmable delay can be set to 0 ms.

The measure time depends on the selected measurement speed. For example, if the speed is set at 0.01 PLC, the measure time would be 167 ms for 60 Hz operation (0.01/60).

Setting the source delay

Using the front panel:

1. Press **FUNCTION** and set the source and measurement function.
2. Press the **MENU** key.
3. Under Source, select **Settings**.
4. Set Source Delay to **Auto Delay** or specify a delay and enter a value.
5. Select **OK**.

Using a remote interface:

- SCPI commands: Refer to :SOURce[1]:<function>:DELay (on page 12-74).
- TSP commands: Refer to smu.source.delay (on page 14-179).

Relative offset

When making measurements, you may want to subtract an offset value from a measurement.

The relative offset feature subtracts a set value or a baseline reading from measurement readings. When you enable relative offset, all measurements are recorded as the difference between the actual measured value and the relative offset value. The formula to calculate the offset value is:

$$Displayed\ value = Actual\ measured\ value - Relative\ offset\ value$$

When a relative offset value is established for a measure function, the value is the same for all ranges for that measure function. For example, if 5 V is set as the relative offset value on the 20 V range, the relative offset value is also 5 V on the 2 V and 200 mV ranges.

On the front panel, when relative offset is enabled, the REL indicator to the right of the measured value is displayed.

A relative offset value is saved for each function. If you change the measure function, the relative offset value is changed to the setting for that measure function.

The relative offset is applied to the measurement before any math and limit test functions. For more information on the order in which operations are performed, see Order of operations (on page 5-27).

NOTE

You can perform the equivalent of relative offset manually by using the mx+b (on page 4-50) math function. Set m to 1 and b to the value of the offset.

Establishing a relative offset value

You can use the 2470 to automatically determine the relative offset, or you can assign a specific relative offset value.

Automatically acquiring a relative offset value

When you automatically acquire a relative offset value, the 2470:

- Makes a new measurement.
- Stores the measurement as the new relative offset level.

Before acquiring the offset, apply the signal that you want to offset the measurement by.

Using the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Select **Calculations**.
4. For Rel, select **Acquire**. The relative offset value is displayed to the right.

When you select **Acquire** from the front panel, Rel is automatically set to On unless an overflow reading is detected.

NOTE

You can also enable or disable the relative offset feature through the SETTINGS swipe screen Rel option.

Using a remote interface:

- SCPI commands: Refer to [:SENSe[1]]:<function>:RELative:ACQuire (on page 12-57) and [:SENSe[1]]:<function>:RELative:STATe (on page 12-58).
- TSP commands: Refer to smu.measure.rel.acquire() (on page 14-161) and smu.measure.rel.enable (on page 14-162).

Setting a relative offset value

You can set a specific relative offset value using the front panel or remote commands.

Using the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Select **Calculations**.
4. For Rel, select **On**.
5. Select **Rel Value**.
6. Enter the value and select **OK**.

Using SCPI commands, send the commands:

```
:SENSe:FUNCtion "VOLTage"
:SENSe:VOLTage:RELative <n>
:SENSe:VOLTage:RELative:STATe ON
```

Where <n> is the amount of the offset.

To set the relative offset for another function, replace `VOLTage` with `CURRent` or `RESistance`. Refer to [:SENSe[1]]:<function>:RELative (on page 12-56) and [:SENSe[1]]:<function>:RELative:STATe (on page 12-58) for additional information.

Using TSP commands, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.rel.level = relValue
smu.measure.rel.enable = smu.ON
```

Where *relValue* is the relative offset value.

To set the relative offset for another function, replace `smu.FUNC_DC_VOLTAGE` with `smu.FUNC_DC_CURRENT` or `smu.FUNC_RESISTANCE`. Refer to smu.measure.rel.level (on page 14-163) and smu.measure.rel.enable (on page 14-162) for additional information.

Disabling the relative offset

Using the front panel:

1. Select the measure function to which the relative offset is applied.
2. On the SETTINGS swipe screen, select **Rel**. An X should be displayed and the REL annunciator to the right of the measurement is no longer displayed.

NOTE

You can also disable the relative offset feature by selecting MENU > Measure > Calculations > Rel and selecting Off.

Using SCPI commands:

Send the command:

```
:SENSe:VOLTage:RELative OFF
```

To set the relative offset for another function, replace `VOLTage` with `CURRent` or `RESistance`.

Using TSP commands:

Send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.rel.enable = smu.OFF
```

To set the relative offset for another function, replace `smu.FUNC_DC_VOLTAGE` with `smu.FUNC_DC_CURRENT` or `smu.FUNC_RESISTANCE`.

Calculations that you can apply to measurements

The 2470 allows you to apply the following math operations to the measurement:

- mx+b
- percent
- reciprocal (1/X)

Math calculations are applied to the input signal after relative offset and before limit tests. For more detail on the order of operations, see Order of operations (on page 5-27).

Math operations apply to the selected measure function. If you change the measure function, the math operation for that function becomes active.

NOTE

Changing math functions does not clear the reading buffer, which can result in mixed units in the reading buffer. If you are graphing, this can cause ? to be displayed on the y-axis. Clear the reading buffer to remove the mixed units.

mx+b

The mx+b math operation lets you manipulate normal display readings (x) mathematically based on the following calculation:

$mx + b = Y$

Where:

- **m** is a user-defined constant for the scale factor
- **x** is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- **b** is a user-defined constant for the offset factor
- **Y** is the displayed result

When the mx+b math operation is active, the unit of measure for the front-panel voltage and current readings is **X** and the MATH indicator is displayed to the right of the measurement. For resistance readings, the units of measure do not change. You cannot change this units designator.

Percent

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

$$\text{Percent} = \left(\frac{\text{input - reference}}{\text{reference}}\right) \times 100\%$$

Where:

- **Percent** = The result
- **Input** = The measurement (if relative offset is being used, this is the relative offset value)
- **Reference** = The user-specified constant

The result of the percent calculation is positive when the input is more than the reference. The result is negative when the input is less than the reference.

When the percent operation is active, the unit of measure for the front-panel voltage and current readings is **%** and the MATH indicator is displayed to the right of the measurement. For resistance readings, the units of measure do not change. You cannot change the unit designator.

Reciprocal (1/X)

You can set math operation to reciprocal to display the reciprocal of a reading.

The reciprocal is 1/X, where X is the reading. If relative offset is on, the 1/X calculation uses the input signal with the relative offset applied.

Example:

Assume the normal displayed reading is 002.5000 Ω. The reciprocal of resistance is conductance. When the reciprocal math function is enabled, the following conductance reading is displayed:

```
0.400000
```

When the reciprocal math operation is active, the unit of measure for the front-panel readings is 1/x and the MATH indicator is displayed to the right of the measurement. You cannot change this units designator.

Setting percent math operations

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Under Measure, select **Calculations**.
4. Set Math to **On**.
5. Select **Settings**.
6. For Math Format, select **Percent**.
7. For Zero Reference, select the percent reference.
8. Select **OK**.
9. Press the **HOME** key to view the measurement with the percent math format applied.

Using a remote interface:

- SCPI commands: Refer to :CALCulate[1]:<function>:MATH:FORMat (on page 12-10) and :CALCulate[1]:<function>:MATH:PERCent (on page 12-15).
- TSP commands: Refer to smu.measure.math.format (on page 14-152) and smu.measure.math.percent (on page 14-155).

Setting mx+b math operations

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Under Measure, select **Calculations**.
4. For Math, select **On**.
5. Select **Settings**.
6. For Math Format, select **mx+b**.
7. For m(Scalar), set the **m** value.
8. For b(Offset), set the **b** value.
9. Select **OK**.
10. Press the **HOME** key to view the measurement with the mx+b math format applied.

Using a remote interface:

- SCPI commands: Refer to :CALCulate[1]:<function>:MATH:FORMat (on page 12-10), :CALCulate[1]:<function>:MATH:MMFactor (on page 12-13), and :CALCulate[1]:<function>:MATH:MBFactor (on page 12-12).
- TSP commands: Refer to smu.measure.math.format (on page 14-152), smu.measure.math.mxb.bfactor (on page 14-153), and smu.measure.math.mxb.mfactor (on page 14-154).

Setting reciprocal math operations

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Under Measure, select **Calculations**.
4. For Math, select **On**.
5. Select **Settings**.
6. For Math Format, select **Reciprocal**
7. Select **OK**.
8. Press the **HOME** key to view the measurement with the reciprocal math format applied.

Using a remote interface:

- SCPI commands: Refer to :CALCulate[1]:<function>:MATH:FORMat (on page 12-10).
- TSP commands: Refer to smu.measure.math.format (on page 14-152) and smu.measure.math.enable (on page 14-151).

Switching math on the SETTINGS swipe screen

Once you set the math operations settings for a measure function, you can turn the math function on or off on the SETTINGS swipe screen.

From the front panel:

1. Select **HOME**.
2. On the SETTINGS swipe screen, select or clear **Math** to enable or disable the selected math operation.
3. To change other math settings, select the calculations settings icon on the right side of the settings swipe screen to open the CALCULATION SETTINGS screen.

Displayed measurements

When you make measurements, the instrument may perform operations on the measured values that affect what you see on the display and the measurements that are stored in the buffer.

The operations that can affect the measurement display are:

- Filtering
- Relative offset
- Math operations
- Limit tests

If none of these operations is set, the value that is displayed on the front panel is the actual measurement reading.

If any one of these operations is set, the value that is displayed is the measurement reading with these operations applied. The operations are applied in the order shown above.

For example, if you made a measurement and had a relative offset and limit tests active, the measured value would have the relative offset applied, then have limit test results applied.

For additional detail on the order of operations, see Order of operations (on page 5-27).

Sweep operation

Sweeps allow you to set up the instrument to source specific voltage or current values to a device under test (DUT). A measurement is made for each value.

The 2470 can generate linear staircase, logarithmic staircase, linear dual staircase, and logarithmic dual staircase sweeps from the front panel or from a remote interface. In addition to these sweeps, you can generate custom sweeps if you use remote commands.

When you generate the sweep, the 2470 creates a source configuration list and a trigger model that contain the settings you selected for the sweep. To run the sweep, press the TRIGGER key. You can also use an initiate command over the remote interface.

Linear staircase sweep

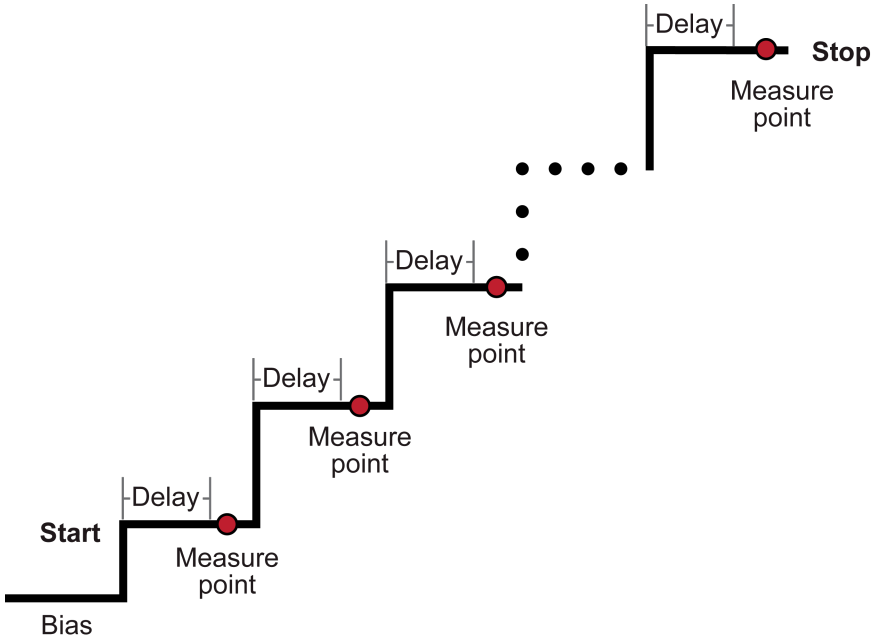
When you use a linear staircase sweep, the voltage or current source increases or decreases in fixed steps. Each source-measure point is equally spaced between the start and stop.

The 2470 sends a buffer clear command at the start of a sweep; if you do not want this action, you can change the sweep trigger model to eliminate the clear block.

The sweep begins with a start voltage or current and ends with a stop voltage or current. A measurement is made at each point after the delay. The figure below shows an increasing linear staircase sweep.

When a linear staircase sweep is triggered to start, the output goes from the bias level to the start source level. The output then changes in equal steps until the stop source level is reached. A measurement is performed at each source step (including the start and stop levels). With trigger delay set to zero, the time duration at each step is determined by the source delay and the time it takes to perform the measurement (the NPLC setting). Note that the delay is the same for all steps in the sweep.

Figure 71: 2470 sweep linear staircase

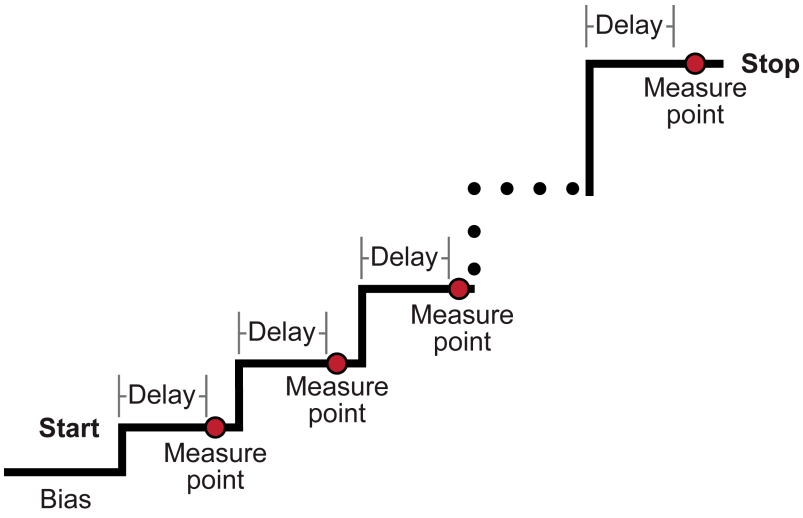


Logarithmic staircase sweep

A logarithmic staircase sweep is similar to a linear staircase sweep. The only difference is that the steps are scaled logarithmically.

The steps in a logarithmic staircase sweep increase or decrease geometrically, beginning with a start voltage or current and ending with a stop voltage or current. The figure below shows an increasing logarithmic staircase sweep.

Figure 72: Logarithmic staircase sweep



Setting up a sweep

NOTE

Defining and generating a sweep creates a new trigger model that will replace an existing trigger model. If you want to preserve the existing trigger model, save a user-saved setup. See Saving setups (on page 3-45) for information on saving an existing trigger model as part of a user-saved setup.

Before setting up the sweep, set up the instrument for the test you will run. Typical settings you can set for a sweep include:

- Source function
- Measure function
- Current or voltage limit
- Source readback
- Voltage protection limits
- 2-wire or 4-wire sense mode
- Front or rear terminal selection

If you change settings after you set up a sweep, those changes affect the sweep the next time it is initiated.

NOTE

You can get extra steps in your sweep if the instrument is set to autorange. If the instrument needs to change range during a step, it creates a new step to accommodate the value that required the range change. To avoid creation of additional steps, use a fixed range or make a measurement on the highest measurement range that is expected during the sweep before running the sweep.

Setting up a sweep from the front panel

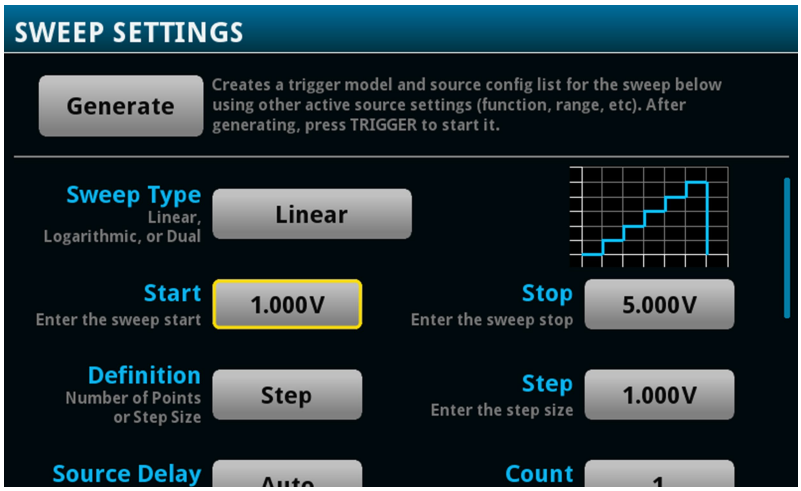
To set up a sweep from the front panel, you select options from the Sweep Settings screen.

The sweep stores results in the active reading buffer. Select the buffer you want to use before setting up the sweep. Note that you cannot use writable reading buffers to store sweep results. See Selecting a buffer (on page 6-14) for more information.

Set up the sweep from the front panel

1. Select **FUNCTION** and select the source and measure functions.
2. On the home screen, set the Source value.
3. Press the **Menu** key.
4. Under Source, select **Sweep**. The Sweep Settings screen is displayed.

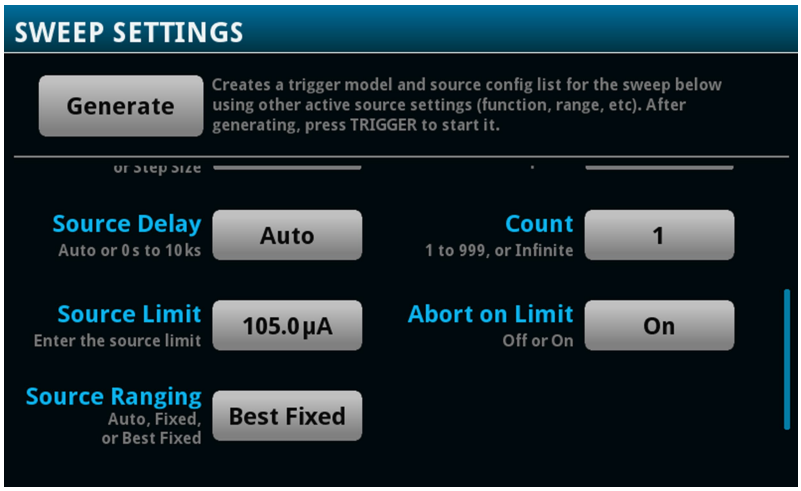
Figure 73: Sweep Settings screen - first page



5. Make selections appropriate to your sweep. See the table below for detail on the options.

6. Swipe down to see additional options.

Figure 74: Sweep Settings screen - second page



7. Make selections appropriate to your sweep. See the table below for detail on the options.
8. Select **Generate**.
9. To run the sweep, press the **TRIGGER** key.

Front-panel sweep options

Option	Description
Type	You can select one of the following options: ■ **Linear**: Sets up a linear staircase sweep. ■ **Logarithmic**: Sets up a logarithmic staircase sweep. ■ **Linear Dual**: Sets up a linear staircase sweep that runs from the start source value to the stop source value, then returns to the start value. ■ **Log Dual**: Sets up a logarithmic staircase sweep that runs from the start source value to the stop source value, then runs from the stop value to the start value.
Start	The voltage or current source level at which the sweep starts: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V To set up an increasing sweep, set start level to be less than the stop level. To set up a decreasing sweep, set the start level to be more than the stop level.
Stop	The voltage or current at which the sweep stops: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V

Front-panel sweep options

Option	Description
Definition	Determines if the sweep is set up for a certain number of points or by a specific step size. Select one of the following options: ■ **Number of Points:** When this option is selected, the instrument calculates the number of source-measure points in the sweep using the following formula: Points = [(Stop - Start) / Step] + 1 ■ **Step Size:** When this option is selected, the source level changes in equal steps from the start level to the stop level. A measurement is performed at each source step (including the start and stop levels). To calculate the number of source-measure points in a sweep, use one of the following formulas. Linear sweep: $step = \frac{stop - start}{points - 1}$ Logarithmic sweep: $log\ step\ size = \frac{log10(stop) - log10(start)}{points - 1}$
Step	Displayed if the sweep definition is set to Step Size. Set the size that each step should be.
Points	Displayed if the sweep definition is set to Number of Points. Select the number of points that you want to measure in the sweep.
Source Delay	Sets the delay (settling time) for the source function.
Count	How many times the sweep should repeat. You can select one of the following options: ■ **Finite**: Set a specific number of times to repeat. ■ **Infinite**: The sweep will repeat until is it aborted.
Source Limit	Sets the source limit for measurements. The instrument cannot source levels that exceed this limit.
Abort on Limit	Determines if the sweep is stopped immediately if a limit is exceeded. You can select one of the following options: ■ **ON**: Abort the sweep if a limit is exceeded. ■ **OFF**: Complete the sweep even if a limit is exceeded.
Source Ranging	The source range that is used for the sweep. You can select one of the following options: ■ **Best Fixed**: The instrument selects a single fixed source range that will accommodate all the source levels in the sweep. This avoids overshoots during sweeps. ■ **Auto:** The instrument selects the most sensitive source range for each source level in the sweep. ■ **Fixed**: The source remains on the range that is set when the sweep is started. If a sweep point exceeds the source range capability, the source will output the maximum level for that range.

Setting up a sweep using SCPI commands

To set up a sweep using SCPI commands, you send one of the following commands:

- `:SOURce[1]:SWEep:<function>:LINear`: Sets up a linear sweep for a fixed number of measurement points.
- `:SOURce[1]:SWEep:<function>:LINear:STEP`: Sets up a linear source sweep configuration list and trigger model with a fixed number of steps.
- `:SOURce[1]:SWEep:<function>:LIST`: Sets up a sweep based on a configuration list, which allows you to customize the sweep.
- `:SOURce[1]:SWEep:<function>:LOG`: Sets up a logarithmic sweep for a set number of measurement points.

To create a sweep:

1. Set the source function using `:SOURce[1]:FUNCtion[:MODE]`.
2. Set the source range using `:SOURce[1]:<function>:RANGe`.
3. Set any other source settings that apply to your sweep. You must set source settings before the sweep function is called.
4. If you are using `:SOURce[1]:SWEep:<function>:LIST`, set up the source configuration list for your sweep.
5. Set the parameters for the sweep command.
6. Set the measurement function using `[:SENSe[1]]:FUNCtion`.
7. Set the measurement range using `[:SENSe[1]]:<function>:RANGe[:UPPer]`.
8. Make any other settings appropriate to your sweep.
9. Send `:INITiate` to start the sweep.

NOTE

To save your settings, save them to a user-saved setup using the *SAV command.

For example sweeps, see Sweep programming examples (on page 4-63).

For detail on the commands and options listed above, see the following command descriptions:

- [:SENSe[1]]:FUNCtion[:ON] (on page 12-68)
- [:SENSe[1]]:<function>:RANGe[:UPPer] (on page 12-55)
- :SOURce[1]:FUNCtion[:MODE] (on page 12-80)
- :SOURce[1]:<function>:RANGe (on page 12-82)
- :SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)
- :SOURce[1]:CONFiguration:LIST:STORe (on page 12-73)
- :SOURce[1]:SWEep:<function>:LINear (on page 12-89)
- :SOURce[1]:SWEep:<function>:LINear:STEP (on page 12-91)
- :SOURce[1]:SWEep:<function>:LIST (on page 12-93)
- :SOURce[1]:SWEep:<function>:LOG (on page 12-95)

Setting up a sweep using TSP commands

To set up a sweep using TSP commands, you send one of the following commands:

- `smu.source.sweeplinear()`: Sets up a linear sweep for a fixed number of measurement points.
- `smu.source.sweeplinearstep()`: Sets up a linear source sweep configuration list and trigger model with a fixed number of steps.
- `smu.source.sweeplist()`: Sets up a sweep based on a configuration list, which allows you to customize the sweep.
- `smu.source.sweeplog()`: Sets up a logarithmic sweep for a set number of measurement points.

To create a sweep:

1. Set the source function using `smu.source.func`.
2. Set the source range using `smu.source.range`.
3. Set any other source settings that apply to your sweep. You must set source settings before the sweep function is called.
4. If you are using `smu.source.sweeplist()`, set up the source configuration list for your sweep.
5. Set the parameters for the sweep command.
6. Set the measurement function using `smu.measure.func`.
7. Set the measurement range using `smu.measure.range`.
8. Make any other settings appropriate to your sweep.
9. Send `trigger.model.initiate()` to start the sweep.

NOTE

To save your settings, save them to a configuration script using the `createconfigscript()` command.

For example sweeps, see Sweep programming examples (on page 4-63).

For detail on the commands and options listed above, see the following command descriptions:

- createconfigscript() (on page 14-52)
- smu.source.sweeplinear() (on page 14-191)
- smu.source.sweeplinearstep() (on page 14-193)
- smu.source.sweeplist() (on page 14-196)
- smu.source.sweeplog() (on page 14-198)
- smu.source.func (on page 14-180)
- smu.source.range (on page 14-187)
- smu.source.configlist.create() (on page 14-172)
- smu.source.configlist.store() (on page 14-177)
- smu.measure.func (on page 14-142)
- smu.measure.range (on page 14-158)

Aborting a sweep

Sweeps can be stopped for the following reasons:

- The limit set by the abort on limit setting was exceeded
- The trigger model is aborted

You can stop the sweep while it is in progress. When you stop the sweep, all sweep commands in the trigger model are terminated.

Using the front panel:

Press the front-panel **TRIGGER** key for two seconds and select **Abort Trigger Model**.

Using SCPI commands:

Send the command:

```
:ABORt
```

Using TSP commands:

Send the command:

```
trigger.model.abort()
```

Sweep programming examples

The following examples show programming examples of typical sweeps.

Linear sweep with a voltage source

The following examples perform a linear sweep that uses a voltage source. They perform the following actions:

- Reset the instrument to its defaults.
- Set the source function to voltage.
- Set the source range to 20 V.
- Set the source limit for measurements to 0.02 A
- Set the measure function to current.
- Set the current range to automatic.
- Set up a linear sweep that sweeps from 0 to 10 V in 21 steps with a source delay of 200 ms.
- In TSP only, name the configuration list that is created for this sweep RES.
- Start the sweeps.
- Wait until all commands are complete and then query the source value and measurement reading.

No buffer is defined, so the data is stored in `defbuffer1`. See Reading buffers (on page 6-1) for more information about reading buffers.

Using SCPI commands

```
*RST
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 0.02
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SOUR:SWE:VOLT:LIN 0, 10, 21, 200e-3
INIT
*WAI
TRAC:DATA? 1, 21, "defbuffer1", SOUR, READ
```

Using TSP commands

```
reset()
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.ON
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.source.ilimit.level = 0.02
smu.source.sweeplinear("RES", 0, 10, 21, 200e-3)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 21, defbuffer1.sourcevalues, defbuffer1.readings)
```

Logarithmic sweep with a current source

The following examples perform a logarithmic sweep using a current source. They perform the following actions:

- Reset the instrument to its defaults.
- Set the source function to current.
- Set the source range to 100 mA.
- Set up a logarithmic sweep from 100 µA to 100 mA in 10 steps with a source delay of 10 ms, a sweep count of 1, and a fixed source range. In TSP only, name the configuration list that is created for this sweep RES.
- Set the measure function to current.
- Set the current range to 100 µA.
- Start the sweep.

No buffer is defined, so the data is stored in `defbuffer1`. See Reading buffers (on page 6-1) for more information on reading buffers.

Using SCPI commands

```
*RST
SOUR:FUNC CURR
SOUR:CURR:RANG 100e-3
SOUR:CURR:VLIM 20
SENS:FUNC "VOLT"
SENS:VOLT:RANG 20
SOUR:SWE:CURR:LOG 100e-6, 100e-3, 10, 10e-3, 1, BEST, OFF
INIT
*WAI
TRAC:DATA? 1, 10, "defbuffer1", SOUR, READ
```

Using TSP commands

```
reset()
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 100e-3
smu.source.vlimit.level = 20
smu.source.sweeplog("RES", 100e-6, 100e-3, 10, 10e-3, 1, smu.RANGE_BEST, smu.OFF)
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
trigger.model.initiate()
waitcomplete()
printbuffer(1, 10, defbuffer1.sourcevalues, defbuffer1.readings)
```

Voltage sweep based on a configuration list

The following TSP example shows a voltage sweep that is based on a configuration list. It:

- Resets the instrument to its defaults.
- Sets a variable that contains the voltage levels 1 V, 10 V, 5 V, 7 V, 1 V, and 9 V.
- Creates a source configuration list called `VoltageListSweep`.
- Set the source function to DC voltage.
- Iterate through `voltageLevels` and store each voltage level in the `VoltageListSweep` configuration list.
- Set up a list sweep that uses the entries from the `VoltageListSweep` configuration list and starts at index 1 of the list.
- Set the measure function to DC voltage.
- Start the sweep.

No buffer is defined, so the data is stored in `defbuffer1`. See Reading buffers (on page 6-1) for more information on reading buffers.

Using TSP commands

```
reset()
local voltageLevels = { 1, 10, 5, 7, 1, 9 }
smu.source.configlist.create("VoltageListSweep")
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 10
for index = 1, table.getn(voltageLevels) do
  smu.source.level = voltageLevels[index]
  smu.source.configlist.store("VoltageListSweep")
end
smu.source.sweeplist("VoltageListSweep",  1, 0.001)
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.initiate()
waitcomplete()
```

Increasing the speed of sweeps

To increase the speed of sweeps:

- Reduce the NPLC.
- Turn autozero off. If autozero is on, the instrument takes new reference and zero values for every reading. This can slow down sweep operation. Be aware that if you disable autozero, measurements may drift and become erroneous. To minimize drift when autozero is disabled, use the autozero once feature. For more information on the autozero options, see Automatic reference measurements (on page 4-44).

Limit testing and binning

The 2470 can be set up for limit testing and binning. It can perform simple benchtop limit testing using the front panel or sophisticated limit and binning operations using the trigger model and digital I/O to control external component-handling devices.

Some typical forms of limit testing include:

- Simple pass-or-fail testing.
- Resistor grading: Inspect multiple limits until the first failure is received.
- Resistor sorting: Inspect multiple limits until the first pass is received.

For binning applications, you use limit testing to determine placement of tested parts. To set up the instrument to place the part in the correct bin, you do the following steps:

- Determine and record a bin number for later use.
- Output a digital bit pattern to physically place the tested device in a bin.
- If multiple tests are performed on the same part, determine when the part should be binned:
  - Bin the part as soon as it fails a test.
  - Bin the part after all parameters are measured; bin according to the first failure or a combination of failures.

Limit testing allows you to set high and low limit values. When the reading falls outside these limits, the instrument displays L1FAIL or L2FAIL.The low limit must be set to the low value and the high limit must be set to the high value to prevent an automatic limit-fail.

The limit values are stored in volatile memory.

Limits are tested after any selected filter, relative offset, and math functions have been applied to the measurement.

The 2470 provides two binning trigger-model templates to assist with setup, one for grading and one for sorting. Refer to Trigger-model templates (on page 8-45).

Limit testing using the front-panel interface

You can do pass or fail limit testing through the front panel. When limit testing and a test fails, the limit (1 or 2) that failed is shown on the home screen.

NOTE

The low value must be less than the high value. If it is not, an automatic limit fail will occur.

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Calculations**.
3. Set Limit 1 or Limit 2 to **On**.
4. Select **Settings**.
5. The Auto Clear setting automatically clears the limit fail indicator when a new passing measurement is made. To turn this feature off, select **Off**.
6. Set the **Low Value**. If the measurement is below the Low Value, the limit failure indicator is displayed.
7. Set the **High Value**. If the measurement is above the High Value, the limit failure indicator is displayed.
8. The Audible setting determines if a beeper sounds when a measurement passes or fails. Set as needed.
9. Select **HOME**.
10. Make a measurement. L1PASS is displayed if the measurement is in the limits; L1FAIL is displayed if the measurement is not in the limits.

An example of using limit testing to check resistors is described in the following topic.

Front-panel limit test to check resistors

This example is set up to test a box of 100 Ω ±1 percent and 100 Ω ±10 percent resistors that you need to separate manually. You can change values as needed to adapt the test to your needs.

To set up the test:

1. Press the **FUNCTION** key.
2. Under Source Current and Measure, select **Resistance**.
3. Press the **MENU** key.
4. Under Measure, select **Calculations**.
5. Set Limit 1 and Limit 2 to **On**.
6. Select **Settings** for Limit 1.
7. Set the High Value to **110 Ω**.
8. Set the Low Value to **90 Ω**.
9. Select **OK**.
10. Select **Settings** for Limit 2.
11. Set the High Value to **101 Ω**.
12. Set the Low Value to **99 Ω**.
13. Select **OK**.
14. Press the **MENU** key.
15. Under Settings, select **Measure**.
16. On the MEASURE SETTINGS screen, set Sense to **4-Wire**. Leave other settings at the default values.

Run the test:

1. Press the **HOME** key.
2. Use 4-wire connections to connect the first resistor to the instrument.
3. Press the **OUTPUT ON/OFF** switch to turn the source on.
4. Verify that the instrument is set to Continuous Measurement. If necessary, hold the **TRIGGER** key for 2 s and select **Continuous Measurement**.
5. Observe the measurements. If the resistor is inside the limits set for Limit 1, L1PASS is displayed. If the resistor is not within the limits, L1FAIL is displayed. If the resistor is in the limits set for Limit 2, L2PASS is displayed. If the resistor is not within the limits, L2FAIL is displayed. An example of a test that passed the L1 test but failed the L2 test is shown below.
6. Press the **OUTPUT ON/OFF** switch to turn the source off. Note that the limit indicators are displayed until you turn limit testing off.

Figure 75: Limit test front-panel indicators



Set up a limit test using the remote interface

You can set up limit testing through a remote interface. There are several methods you can use to set up the limit test:

- Scripting (available with TSP only): Allows the most flexibility; you can set up the limit test as needed.
- Trigger model: Provides the best speed and throughput, using predefined trigger blocks to simplify set up.

The following topics show examples using simple and complex limit and binning tests.

Resistor grading using limit testing

This limit test inspects multiple limits until the first failure is received. When a resistor fails, it is sorted into the appropriate bin.

This example grades resistors into tolerance levels (for example, 20%, 10%, 5%, and 1%). A single spot measurement is inspected against multiple limits, which tighten progressively around the same nominal value. Since there is no reason to continue limit checking once the appropriate tolerance level for a resistor-under-test is determined, this application will typically immediately bin the tested resistors. The bit patterns assigned to the limits determine into which bin a resistor is placed.

For this example, the same fail bit pattern is assigned to both the lower and upper bounds of the limits so that resistors with resistance values in the range R-P% to R go into the same bin as those with resistance values in the range R to R+P%. If you want to put parts in separate bins that correspond to R-P% to R and R to R+P%, you can do so by assigning different bit patterns for the upper and lower bounds of the limits.

Since the limits are inspected in ascending numeric order, the measured resistance is checked first against Limit 2, which is the 20% limit. If a resistor fails this limit inspection, its resistance value is outside of the 20% tolerance band and it is considered to be a bad part. The trigger model outputs the Limit 2 fail bit pattern, which causes the component handler to place the resistor in the Bad Part bin.

If a resistor passes the 20% limit test, the resistance value is checked against the 10% limit value. If the resistor fails this limit inspection, the resistance is outside of the 10% tolerance band, but in the 20% band. The trigger model outputs the Limit 3 fail bit pattern, which causes the component handler to place the resistor in the 20% tolerance part bin.

If a resistor passes the 10% limit test, the resistance value is checked against the 5% limit value, and so on. If a resistor passes all the limit tests, the trigger model outputs the overall pass bit pattern, which causes the component handler to place the resistor in the 1% tolerance part bin.

Resistor grading example

The following diagrams show the trigger model flow for the resistor grading example.

Figure 76: Resistor grading example blocks 1 to 6

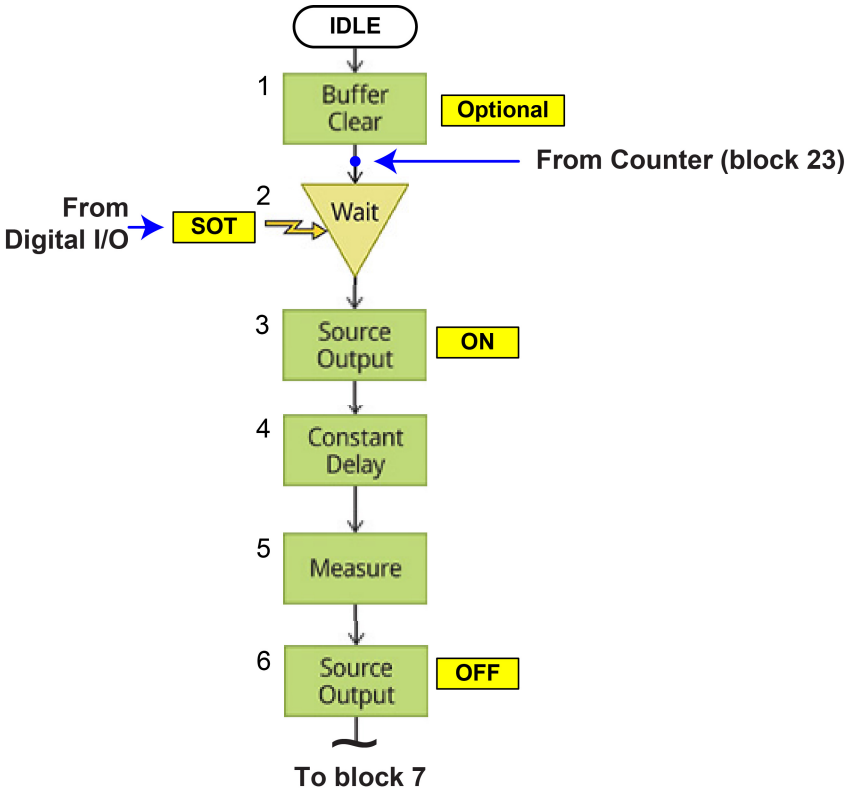


Figure 77: Resistor grading example blocks 7 to 18

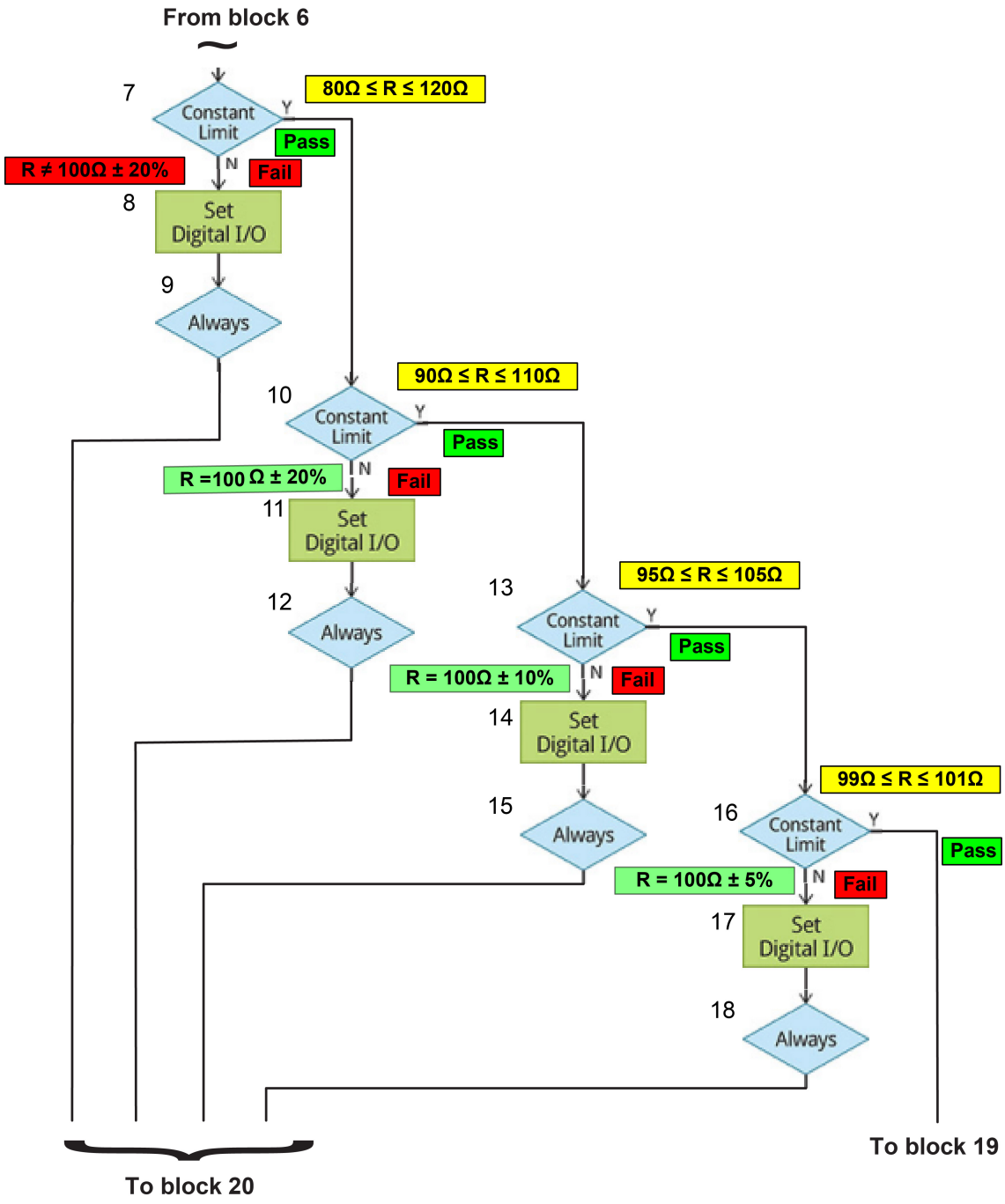
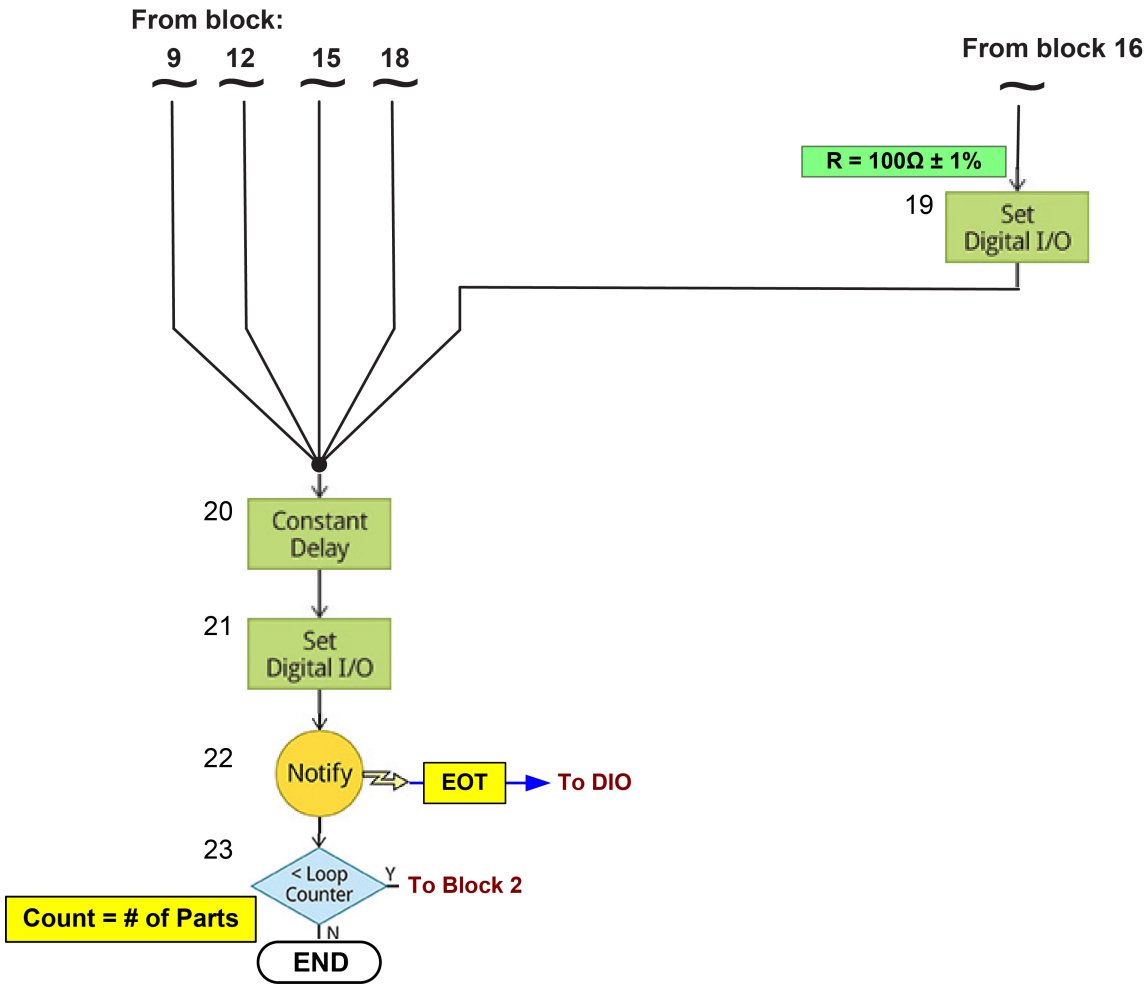


Figure 78: Resistor grading example blocks 19 to 23



Resistor grading SCPI code

Send the following commands for this example application:

Command	Description

`*RST`
 	
`SOURce:FUNCtion CURRent`
`SOURce:CURRent:RANGe 0.01`
`SOURce:CURRent 0.01`
`SOURce:CURRent:VLIM 2`
`SOURce:CURRent:READ:BACK ON`
 	
`SENSe:FUNCtion "VOLTage"`
`SENSe:VOLTage:UNIT OHM`
`SENSe:VOLTage:RSENse ON`
`SENSe:VOLTage:NPLC 1`
 	
`TRACe:POINts 10`
	Reset the 2470.
	
	Set the instrument to source current with a range of 10 mA and a voltage limit of 2 V. Turn source readback on.
	
	Set the instrument measure voltage in ohms and set 4-wire remote sensing on.
	Set the NPLCs to 1.
	
	Set the reading buffer size to 10.

`DIGital:LINE1:MODE DIG, OUT`
`DIG:LINE2:MODE DIG, OUT`
`DIG:LINE3:MODE DIG, OUT`
`DIG:LINE4:MODE DIG, OUT`
 	
`DIG:LINE5:MODE TRIG, IN`
`TRIGger:DIG5:IN:EDGE FALL`
 	
`DIG:LINE6:MODE TRIG, OUT`
`TRIGger:DIGital6:OUT:LOGic NEG`
`TRIG:DIG6:OUT:PULSewidth 10e-6`
`TRIG:DIG6:OUT:STIMulus NOT1`
	Set the digital I/O lines 1 to 4 to be digital lines that detect rising-edge or falling-edge triggers as input.
	
	Set digital I/O line 5 for trigger model control, detecting falling-edge triggers as input.
	
	Set digital I/O line 6 for trigger model control, detecting rising-edge or falling-edge triggers as output. Set the output trigger logic of the trigger event generator to negative. Set the length of time that the trigger line is asserted to 10e-6. Set the stimulus to create a notify event.

`TRIGger:LOAD:EMPTy`
 	
`TRIGger:BLOCk:BUFFer:CLE 1, "defbuffer1"`
`TRIG:BLOC:WAIT 2, DIG5`
`TRIG:BLOC:SOUR:STAT 3, ON`
`TRIG:BLOC:DEL:CONS 4, 0.001`
`TRIG:BLOC:MDIG 5, "defbuffer1"`
`TRIG:BLOC:SOUR:STAT 6, OFF`
`TRIG:BLOC:BRAN:LIM:CONS 7, IN, 80, 120, 10, 5`
`TRIG:BLOC:DIG:IO 8, 15, 15`
`TRIG:BLOC:BRAN:ALW 9, 20`
`TRIG:BLOC:BRAN:LIM:CONS 10, IN, 90, 110, 13, 5`
`TRIG:BLOC:DIG:IO 11, 1, 15`
`TRIG:BLOC:BRAN:ALW 12, 20`
`TRIG:BLOC:BRAN:LIM:CONS 13, IN, 95, 105, 16, 5`
`TRIG:BLOC:DIG:IO 14, 2, 15`
`TRIG:BLOC:BRAN:ALW 15, 20`
`TRIG:BLOC:BRAN:LIM:CONS 16, IN, 99, 101, 19, 5`
`TRIG:BLOC:DIG:IO 17, 3, 15`
`TRIG:BLOC:BRAN:ALW 18, 20`
`TRIG:BLOC:DIG:IO 19, 4, 15`
	Clear any existing trigger model commands from the instrument.
	Set up the trigger model:
	■ Block 1: Clear default buffer 1.
	■ Block 2: Set up a wait block to wait for digital line 5.
	■ Block 3: Turn the source output on.
	■ Bock 4: Set a constant delay of 0.001 s.
	■ Block 5: Make a measurement and store it in default buffer 1.
	■ Block 6: Turn the output off.
	■ Block 7: Set up the constant limits to perform the first test.
	■ Block 8: If block 7 fails, drive the digital I/O lines high.
	■ Blocks 9, 12, 15, and 18: Branch to block 20.

`TRIG:BLOC:DEL:CONS 20, 0.001`
`TRIG:BLOC:DIG:IO 21, 0, 15`
`TRIG:BLOC:NOT 22, 1`
`TRIG:BLOC:BRAN:COUN 23, 10, 2`
	■ Block 10: Set up the constant limits to perform the second test.
	■ Block 11: If block 10 fails, drive line 1 high.
	■ Block 13: Set up the constant limits to perform the third test.
	■ Block 14: If block 13 fails, drive line 2 high.
	■ Block 16: Set up the constant limits to perform the fourth test.
	■ Block 17: If block 10 fails, drive lines 1 and 2 high.
	■ Block 19: If block 10 fails, drive line 3 high.
	■ Block 20: Delay for 1 ms
	■ Block 21: Set all digital lines low.
	■ Block 22: Send out a notify event to indicate that the test has ended.
	■ Block 23: Return to block 2 23 times.

Resistor grading TSP code

```
local number_of_resistors = 100
-- Reset instrument to default settings.
reset()
-- Measure function must be first measure setting;
-- most other settings are tied to the function.
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.unit = smu.UNIT_OHM
-- Report Vmeasured/Isource.
-- Use 4-wire or "remote" voltage sensing.
smu.measure.sense = smu.SENSE_4WIRE
-- Measure the actual value of the source for higher accuracy.
smu.source.readback = smu.ON
-- Default setting; turn off for more speed, but reduced accuracy.
-- Set measurement integration time to 1 PLC (16.67 ms at 60 Hz).
-- Decrease to reduce test time; trade off accuracy for speed.
smu.measure.nplc = 1
-- Immediately update autozero reference measurements and then
-- disable the autozero function.
smu.measure.autozero.once()
-- Source function must be first source setting;
-- most other settings are tied to the function.
smu.source.func = smu.FUNC_DC_CURRENT

-- Set this after setting source function to current.
smu.measure.range = 2
-- This is a current range.
smu.source.range = 0.01
-- Set source level to 10 mA.
smu.source.level = 0.01
-- Set voltage limit of current source to 2V; set this after setting measure range.
smu.source.vlimit.level = 2
```

```
-- This example records the resistance measurements
-- for later statistical analysis.
-- Limit inspection and binning can be performed
-- without recording the measurements.
-- Set the buffer capacity equal to the number of resistors
-- to be tested.
defbuffer1.capacity = number_of_resistors
-- Configure digital I/O lines 1 through 4 as digital outputs.
-- These I/O lines are used to output binning code to component handler.
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[2].mode = digio.MODE_DIGITAL_OUT
digio.line[3].mode = digio.MODE_DIGITAL_OUT
digio.line[4].mode = digio.MODE_DIGITAL_OUT

-- Configure digital I/0 line 5 as a trigger input.
-- Used to detect start-of-test trigger from component handler.
digio.line[5].mode = digio.MODE_TRIGGER_IN
-- Set trigger detector to detect falling edge.
trigger.digin[5].edge = trigger.EDGE_FALLING

-- Configure digital I/0 line 6 as a trigger output.
-- Used to send end-of-test trigger to component handler.
digio.line[6].mode = digio.MODE_TRIGGER_OUT
-- Output a falling edge trigger.
trigger.digout[6].logic = trigger.LOGIC_NEGATIVE
-- Set width of output trigger pulse to 10 us.
trigger.digout[6].pulsewidth = 10E-6
-- Trigger pulse will be output when Notify Block generates an event.
trigger.digout[6].stimulus = trigger.EVENT_NOTIFY1
-- Reset existing trigger model settings.
trigger.model.load("Empty")
-- Configure the Trigger Model.
-- Block 1: Clear defbuffer1.
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
-- Block 2: Wait for start-of-test trigger on digital I/O line 5.
trigger.model.setblock(2, trigger.BLOCK_WAIT, trigger.EVENT_DIGIO5)
-- Block 3: Turn SMU output ON.
trigger.model.setblock(3, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
-- Block 4: Delay for 1ms to allow source to settle; adjust as appropriate.
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 5: Measure resistance and store result in defbuffer1.
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
-- Block 6: Turn SMU output OFF.
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Block 7: Check if 80<=R<=120; if yes, go to Block 10.
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_INSIDE,
   80, 120, 10, 5)
-- Block 8: Set digital I/O lines 1-4; output decimal 15 (binary 1111)
-- to component handler.
trigger.model.setblock(8, trigger.BLOCK_DIGITAL_IO, 15, 15)
-- Block 9: Go to Block 20.
trigger.model.setblock(9, trigger.BLOCK_BRANCH_ALWAYS, 20)
-- Block 10: Check if 90<=R<=110; if yes, go to Block 13.
trigger.model.setblock(10, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 90, 110, 13, 5)
```

```
-- Block 11: Set digital I/O lines 1-4; output decimal 1 (binary 0001)
-- to component handler.
trigger.model.setblock(11, trigger.BLOCK_DIGITAL_IO, 1, 15)
-- Block 12: Go to Block 20.
trigger.model.setblock(12, trigger.BLOCK_BRANCH_ALWAYS, 20)
-- Block 13: Check if 95<=R<=105; if yes, go to Block 16.
trigger.model.setblock(13, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 95, 105, 16, 5)
-- Block 14: Set digital I/O lines 1-4; output decimal 2 (binary 0010)
-- to component handler.
trigger.model.setblock(14, trigger.BLOCK_DIGITAL_IO, 2, 15)
-- Block 15: Go to Block 20.
trigger.model.setblock(15, trigger.BLOCK_BRANCH_ALWAYS, 20)

-- Block 16: Check if 99<=R<=101; if yes, go to Block 19.
trigger.model.setblock(16, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 99, 101, 19, 5)
-- Block 17: Set digital I/O lines 1-4; output decimal 3 (binary 0011)
-- to component handler.
trigger.model.setblock(17, trigger.BLOCK_DIGITAL_IO, 3, 15)
-- Block 18: Go to Block 20.
trigger.model.setblock(18, trigger.BLOCK_BRANCH_ALWAYS, 20)
-- Block 19: Set digital I/O lines 1-4; output decimal 4 (binary 0100)
-- to component handler.
trigger.model.setblock(19, trigger.BLOCK_DIGITAL_IO, 4, 15)
-- Block 20: Delay 1ms; controls duration of digital bit patterns;
-- adjust as appropriate.
trigger.model.setblock(20, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 21: Set digital I/O lines 1-4; output decimal 0 (binary 0000)
-- clear pattern to component handler.
trigger.model.setblock(21, trigger.BLOCK_DIGITAL_IO, 0, 15)
-- Block 22: Notify block generates event, which causes output
-- of a trigger pulse on digital I/O line 6.
trigger.model.setblock(22, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
-- Block 23: Loop back to Block 2; keep looping until
-- all resistors have been tested.
trigger.model.setblock(23, trigger.BLOCK_BRANCH_COUNTER, number_of_resistors, 2)
-- After executing all the above commands, the trigger model can be initiated
-- by executing trigger.model.initiate().
```

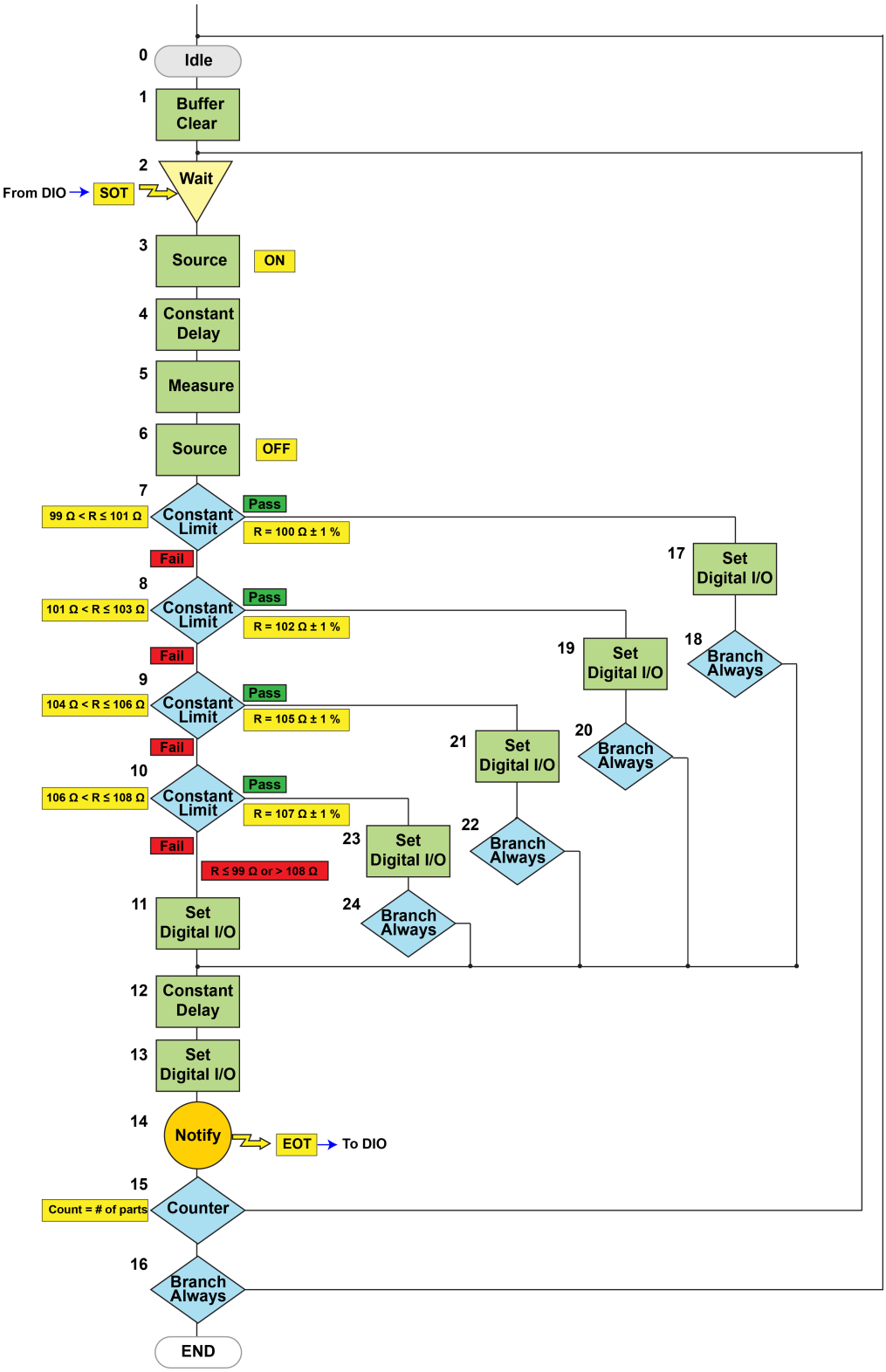
Resistor sorting using limit testing with multiple limits

This example inspects multiple resistors until it detects the first pass. This example uses a trigger model using constant limits.

This trigger model provides support for inspecting the output of a single test against multiple limits. The trigger model count block determines the number of devices that will be tested. A test refers to a single source or measure operation. After a measurement for a particular test is performed, it is checked against any enabled limits. The software limits are inspected in ascending numeric order until the first pass is detected. When a pass condition is detected, the pass bit pattern assigned to the limit that passed is output to the digital I/O port, and any subsequent limit inspections are aborted. If all limit inspections for the test fail, the overall fail bit pattern is output.

The following diagram shows the trigger model flow for the resistor sorting example.

Figure 79: Resistor sorting example trigger model



Resistor sorting SCPI code

Send the following commands for this example application:

Command	Description

`*RST`
 	
`SOUR:FUNC CURR`
`SOUR:CURR:RANG 0.01`
`SOUR:CURR 0.01`
`SOUR:CURR:VLIM 2`
`SOUR:CURR:READ:BACK ON`
 	
`SENS:FUNC "VOLTage"`
`SENS:VOLT:UNIT OHM`
`SENS:VOLT:RSEN ON`
`SENS:VOLT:NPLC 1`
 	
`TRAC:POIN 10`
	Reset the 2470.
	
	Set the instrument to source current with a range of 10 mA and a voltage limit of 2 V. Turn source readback on.
	
	Set the instrument measure voltage and set to display ohms. Set 4-wire remote sensing on.
	Set the NPLCs to 1.
	
	Set the reading buffer size to 10.

`DIG:LINE1:MODE DIG, OUT`
`DIG:LINE2:MODE DIG, OUT`
`DIG:LINE3:MODE DIG, OUT`
`DIG:LINE4:MODE DIG, OUT`
 	
`DIG:LINE5:MODE TRIG, IN`
`TRIGger:DIG5:IN:EDGE FALL`
 	
`DIG:LINE6:MODE TRIG, OUT`
`TRIG:DIG6:OUT:LOG NEG`
`TRIG:DIG6:OUT:PULS 10e-6`
`TRIG:DIG6:OUT:STIM NOT1`
	Set the digital I/O lines 1 to 4 to be digital output lines.
	
	Set digital I/O line 5 for trigger control, detecting falling-edge triggers as input.
	
	Set digital I/O line 6 as a trigger output line. Set the output trigger logic of the trigger event generator to negative. Set the length of time that the trigger line is asserted to 10e-6 s. Set the stimulus to create as Notify1.

`TRIG:LOAD "Empty"`
 	
`TRIG:BLOC:BUFF:CLE 1, "defbuffer1"`
 	
`TRIG:BLOC:WAIT 2, DIG5`
 	
`TRIG:BLOC:SOUR:STAT 3, ON`
 	
`TRIG:BLOC:DEL:CONS 4, 0.001`
 	
`TRIG:BLOC:MDIG 5, "defbuffer1"`
 	
`TRIG:BLOC:SOUR:STAT 6, OFF`
 	
`TRIG:BLOC:BRAN:LIM:CONS 7, IN, 99, 101, 17, 5`
 	
`TRIG:BLOC:BRAN:LIM:CONS 8, IN, 101, 103, 19, 5`
 	
`TRIG:BLOC:BRAN:LIM:CONS 9, IN, 104, 106, 21, 5`
 	
`TRIG:BLOC:BRAN:LIM:CONS 10, IN, 106, 108, 23, 5`
	Clear any existing trigger model commands from the instrument.
	Set up the trigger model:
	■ Block 1: Clear default buffer 1.
	■ Block 2: Set up a wait block to wait for digital line 5.
	■ Block 3: Turn the source output on.
	■ Bock 4: Set a constant delay of .001 s.
	■ Block 5: Make a measurement and store it in default buffer 1.
	■ Block 6: Turn the output off.
	■ Block 7: Check if 99 ≤ R ≤ 101; if yes, go to Block 17.
	■ Block 8: Check if 101 ≤ R ≤ 103; if yes, go to Block 19.
	■ Block 9: Check if 104 ≤ R ≤ 106; if yes, go to Block 21.
	■ Block 10: Check if 106 ≤ R ≤ 108; if

```
TRIG:BLOC:DIG:IO 11, 15, 15

TRIG:BLOC:DEL:CONS 12, 0.001

TRIG:BLOC:DIG:IO 13, 0, 15

TRIG:BLOC:NOT 14, 1

TRIG:BLOC:BRAN:COUN 15, 10, 2

TRIG:BLOC:BRAN:ALW 16, 0

TRIG:BLOC:DIG:IO 17, 1, 15

TRIG:BLOC:BRAN:ALW 18, 12

TRIG:BLOC:DIG:IO 19, 2, 15

TRIG:BLOC:BRAN:ALW 20, 12

TRIG:BLOC:DIG:IO 21, 3, 15

TRIG:BLOC:BRAN:ALW 22, 12

TRIG:BLOC:DIG:IO 23, 4, 15

TRIG:BLOC:BRAN:ALW 24, 12
```

yes, go to Block 23.

- Block 11: Set digital I/O lines 1 through 4; output decimal 15 (binary 1111) to component handler.
- Block 12: Delay 1 ms. Controls duration of digital bit patterns; adjust as appropriate.
- Block 13: Set digital I/O lines 1 through 4; output decimal 0 (binary 0000) clear pattern to component handler.
- Block 14: Notify block generates event, which causes output of a trigger pulse on digital I/O line 6.
- Block 15: Loop back to Block 2; keep looping until all resistors have been tested.
- Block 16: Go to Block 0 (go to Idle).
- Block 17: Set digital I/O lines 1 through 4; output decimal 1 (binary 0001) to component handler.
- Block 18: Go to Block 12.
- Block 19: Set digital I/O lines 1 through 4; output decimal 2 (binary 0010) to component handler.
- Block 20: Go to Block 12.
- Block 21: Set digital I/O lines 1 through 4; output decimal 3 (binary 0011) to component handler.
- Block 22: Go to Block 12.
- Block 23: Set digital I/O lines 1 through 4; output decimal 4 (binary 0100) to component handler.
- Block 24: Go to Block 12.

Resistor sorting TSP code

```
-- Create global variable to hold the number of resistors to be tested.
-- Can use local variable if all commands are sent to instrument as
-- a single script
number_of_resistors = 10
-- Reset instrument to default settings.
reset()
-- Clear the event log.
eventlog.clear()
-- Source function must be the first source setting;
-- most other settings are tied to the function.
smu.source.func = smu.FUNC_DC_CURRENT
-- Set the current range.
smu.source.range = 0.01
-- Set the source level to 10 mA.
smu.source.level = 0.01
```

```
-- The measure function must be first measure setting;
-- most other settings are tied to the function.
smu.measure.func = smu.FUNC_DC_VOLTAGE
-- Set the measure range after setting source function to current.
smu.measure.range = 2
-- Report Vmeasured/Isource.
smu.measure.unit = smu.UNIT_OHM
-- Use 4-wire or remote voltage sensing.
smu.measure.sense = smu.SENSE_4WIRE
-- Measure the actual value of the source for higher accuracy.
smu.source.readback = smu.ON
-- Set measurement integration time to 1 PLC (16.67 ms at 60 Hz).
-- Decrease to reduce test time; trade-off accuracy for speed.
smu.measure.nplc = 1
-- Immediately update autozero reference measurements and then
-- disable autozero function.
smu.measure.autozero.once()
-- Set voltage limit of current source to 2 V; set this after
-- setting measure range.
smu.source.vlimit.level = 2
-- This example records the resistance measurements for later
-- statistical analysis.
-- Limit inspection and binning can be performed without recording
-- the measurements.
-- Set default buffer equal to the number of resistors to be tested.
defbuffer1.capacity = number_of_resistors
-- Configure digital I/O lines 1 through 4 as digital outputs.
-- These I/O lines are used to output binning code to component handler.
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[2].mode = digio.MODE_DIGITAL_OUT
digio.line[3].mode = digio.MODE_DIGITAL_OUT
digio.line[4].mode = digio.MODE_DIGITAL_OUT
-- Configure digital I/0 line 5 as a trigger input.
-- Used to detect start-of-test trigger from component handler.
digio.line[5].mode = digio.MODE_TRIGGER_IN
-- Set trigger detector to detect falling edge.
trigger.digin[5].edge = trigger.EDGE_FALLING
-- Configure digital I/0 line 6 as a trigger output.
-- Used to send end-of-test trigger to component handler.
digio.line[6].mode = digio.MODE_TRIGGER_OUT
-- Output a falling edge trigger.
trigger.digout[6].logic = trigger.LOGIC_NEGATIVE
-- Set width of output trigger pulse to 10 us.
trigger.digout[6].pulsewidth = 10E-6
-- Trigger pulse will be output when Notify Block generates an event.
trigger.digout[6].stimulus = trigger.EVENT_NOTIFY1
-- Reset existing trigger model settings.
trigger.model.load("Empty")
-- Configure the trigger model.
-- Block 1: Clear defbuffer1.
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
-- Block 2: Wait for start-of-test trigger on digital I/O line 5.
trigger.model.setblock(2, trigger.BLOCK_WAIT, trigger.EVENT_DIGIO5)
-- Block 3: Turn source output on.
trigger.model.setblock(3, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
-- Block 4: Delay for 1 ms to allow source to settle; adjust as appropriate.
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 0.001)
```

```
-- Block 5: Measure resistance and store result in defbuffer1.
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
-- Block 6: Turn source output off.
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Block 7: Check if 99<=R<=101; if yes, go to Block 17.
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_INSIDE,
   99, 101, 17, 5)
-- Block 8: Check if 101<=R<=103; if yes, go to Block 19.
trigger.model.setblock(8, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_INSIDE,
   101, 103, 19, 5)
-- Block 9: Check if 104<=R<=106; if yes, go to Block 21.
trigger.model.setblock(9, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_INSIDE,
   104, 106, 21, 5)
-- Block 10: Check if 106<=R<=108; if yes, go to Block 23.
trigger.model.setblock(10, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 106, 108, 23, 5)
-- Block 11: Set digital I/O lines 1 through 4; output
-- decimal 15 (binary 1111) to component handler.
trigger.model.setblock(11, trigger.BLOCK_DIGITAL_IO, 15, 15)
-- Block 12: Delay 1 ms. Controls duration of digital bit patterns;
-- adjust as appropriate.
trigger.model.setblock(12, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 13: Set digital I/O lines 1 through 4; output
-- decimal 0 (binary 0000) clear pattern to component handler.
trigger.model.setblock(13, trigger.BLOCK_DIGITAL_IO, 0, 15)
-- Block 14: Notify block generates event, which causes output
-- of a trigger pulse on digital I/O line 6.
trigger.model.setblock(14, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
-- Block 15: Loop back to Block 2; keep looping until all resistors
-- have been tested.
trigger.model.setblock(15, trigger.BLOCK_BRANCH_COUNTER, number_of_resistors, 2)
-- Block 16: Go to Block 0 (go to Idle).
trigger.model.setblock(16, trigger.BLOCK_BRANCH_ALWAYS, 0)
-- Block 17: Set digital I/O lines 1 through 4; output
-- decimal 1 (binary 0001) to component handler.
trigger.model.setblock(17, trigger.BLOCK_DIGITAL_IO, 1, 15)
-- Block 18: Go to Block 12.
trigger.model.setblock(18, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- Block 19: Set digital I/O lines 1 through 4; output
-- decimal 2 (binary 0010) to component handler.
trigger.model.setblock(19, trigger.BLOCK_DIGITAL_IO, 2, 15)
-- Block 20: Go to Block 12.
trigger.model.setblock(20, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- Block 21: Set digital I/O lines 1 through 4; output
-- decimal 3 (binary 0011) to component handler.
trigger.model.setblock(21, trigger.BLOCK_DIGITAL_IO, 3, 15)
-- Block 22: Go to Block 12.
trigger.model.setblock(22, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- Block 23: Set digital I/O lines 1 through 4; output
-- decimal 4 (binary 0100) to component handler.
trigger.model.setblock(23, trigger.BLOCK_DIGITAL_IO, 4, 15)
-- Block 24: Go to Block 12.
trigger.model.setblock(24, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- After executing all of the above commands, the trigger model
-- can be initiated by executing the command trigger.model.initiate().
```

Configuration lists

A configuration list is a list of stored settings for the source or measure function. You can restore these settings to change the function and its settings that are used by the instrument.

Configuration lists allow you to store the function settings of the instrument and then return the instrument to those settings as needed.

The instrument also uses configuration lists to manage the settings for sweeps.

You can recall configuration lists from the front panel, using remote commands, or as part of a trigger model.

NOTE

Do not change the configuration lists that are created by the instrument directly. Use the front-panel options or remote commands to make changes to sweeps.

The 2470 supports source configuration lists and measure configuration lists. Source configuration lists contain the function setting and the settings for the source function, such as the source readback, protection limit, and source level settings. Measure configuration lists contain the measure function setting and the settings for the measure function, such as the NPLC, display digits, and math settings. If you are familiar with the Model 2400, using configuration lists is similar to using Source Memory.

Configuration indexes

A configuration index contains a copy of all instrument source or measure settings at a specific point. Configuration lists are typically made up of multiple indexes.

You can store a maximum of 300,000 indexes.

The first time you store a configuration index, the instrument stores the settings in configuration index 1. Subsequent indexes are numbered sequentially. You can use the index number to identify a specific configuration index and perform operations on it, such as when using the ConfigList trigger-model template.

The settings that are stored in configuration list indexes are listed in Settings stored in a measure configuration index (on page 4-83).

Settings stored in a source configuration index

The following source settings are stored in a source configuration index:

- Function
- Auto Delay
- Delay
- High Capacitance
- Limit Level
- Output Off State
- Overvoltage Protection Limit
- Source Autorange
- Source Level
- Source Range
- Source Readback
- User Delays 1 to 5

Settings stored in a measure configuration index

The following measure settings can be stored in a measure configuration index.

Function settings:

- Auto Zero
- Autorange
- Auto Range High Limit
- Auto Range Low Limit
- Auto Range Rebound
- Display Digits
- Function
- NPLC
- Offset Compensation
- Range
- Sense
- Unit
- User Delays 1 to 5

Filter settings:

- Enable
- Count
- Type

Limit 1 and Limit 2 settings:

- Enable
- Audible
- Auto Clear
- High Value
- Low Value

Math settings:

- Enable
- Format
- b(Offset)
- m(Scalar)
- Percent

Relative offset settings:

- Enable
- Level

Working with configuration lists and indexes

To create a configuration index, you need to:

- Create the configuration list
- Configure the instrument with the settings that you want to store in a configuration index
- Store the settings into a configuration index in the specified configuration list

After you store configuration indexes to a configuration list, you can do the following operations:

- Recall a configuration index and restore the instrument to the stored settings
- View the contents of a configuration index
- Delete a configuration index
- Delete the entire configuration list

You can work with configuration lists from the front panel or by using remote commands.

NOTE

Recall source configuration lists before measure configuration lists. This order ensures that dependencies between source and measure settings are properly handled.

Create a configuration list

This example creates a source configuration list named `MySourceList` and a measure configuration list named `MyMeasList`.

NOTE

Configuration list names must be unique. For example, the name of a source configuration list cannot be the same as the name of a measure configuration list.

To use the front panel to create a source configuration list:

1. Press the **MENU** key.
2. Under Source, select **Config Lists**. The SOURCE CONFIGURATION LISTS screen is displayed.
3. Select **Create New**. If a list already exists, choose Select and choose **Create New**. The keypad is displayed.
4. Enter a name for the configuration list you are creating. For this example, enter **MySourceList**.
5. Select the **OK** button on the displayed keyboard. The SOURCE CONFIGURATION LISTS screen is displayed.

To use the front panel to create a measure configuration list:

1. Press the **MENU** key.
2. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **Create New**. If a list already exists, choose **Select** and choose **Create New**. The keypad is displayed.
4. Enter a name for the configuration list you are creating. For this example, enter `MyMeasList`.
5. Select the **OK** button on the displayed keyboard. The MEASURE CONFIGURATION LISTS screen is displayed.

To use SCPI commands to create a source configuration list:

```
:SOURce:CONFiguration:LIST:CREate "MySourceList"
```

To use SCPI commands to create a measure configuration list:

```
:SENSe:CONFiguration:LIST:CREate "MyMeasList"
```

To use TSP commands to create a source configuration list:

```
smu.source.configlist.create("MySourceList")
```

To use TSP commands to create a measure configuration list:

```
smu.measure.configlist.create("MyMeasList")
```

Store settings into a configuration list index

This section describes how to store instrument settings to an index in a configuration list.

A configuration index contains a copy of the instrument source or measure settings for a function at a specific time. You can store up to 300,000 indexes.

The following examples make settings on the instrument and stores them in configuration lists `MySourceList` and `MyMeasList`.

Store settings using the front panel

To configure the instrument and store the settings into indexes:

1. Press the **FUNCTION** key.
2. Select **Source Voltage and Measure Current**.
3. Press the **MENU** key.
4. Under Source, select **Settings**. The SOURCE SETTINGS menu is displayed.
5. Set the Range to **20 mV**.
6. Press the **MENU** key.
7. Under Source, select **Config Lists.** The SOURCE CONFIGURATION LISTS screen is displayed.
8. Select **MySourceList**.
9. Select **Add Settings**. The configuration index is displayed in the list.
10. Press the **MENU** key.
11. Under Source, select **Settings**. The SOURCE SETTINGS menu is displayed.
12. Change the Range to 2 V.
13. Press the **MENU** key.
14. Under Source, select **Config Lists**.
15. Select **Add Settings.** The configuration index is displayed with the differences from the first index. If there are no differences, "No change" is displayed for that index.

To configure the instrument and store the measurement settings into indexes:

1. Press the **FUNCTION** key.
2. Select **Source Voltage and Measure Current**.
3. Press the **MENU** key.
4. Under Measure, select **Settings**. The MEASURE SETTINGS menu is displayed.
5. Set the Range to **10 mA**.
6. Set NPLC to **1.00**.
7. Press the **MENU** key.
8. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
9. Select **MyMeasList**.
10. Select **Add Settings**. The configuration index is displayed on the list.
11. Press the **MENU** key.
12. Under Measure, select **Settings**. The MEASURE SETTINGS menu is displayed.
13. Change NPLC to **2**.
14. Press the **MENU** key.
15. Under Measure, select **Config Lists**.
16. Select **Add Settings**. The configuration index is displayed on the list with the differences from the first index. If there are no differences, "No change" is displayed for that index.

Store settings using SCPI commands

This example:

- Sets the source function to voltage
- Sets the source range to 20 V
- Sets the source limit to 2.0 V
- Stores the settings to `MySourceList`
- Changes the source level to 3.0 V
- Stores the setting to `MySourceList`
- Sets the source level to 4.0 V
- Stores the setting to `MySourceList`
- Sets the source level to 5.0 V
- Stores the setting to `MySourceList`

Send the following SCPI commands:

```
:SOURce:FUNCtion VOLTage
:SOURce:VOLtage:RANGe 20
:SOURce:VOLTage:LEVel 2
:SOURce:CONF:LIST:STORe "MySourceList"
:SOURce:VOLTage:LEVel 3
:SOURce:CONF:LIST:STORe "MySourceList"
:SOURce:VOLTage:LEVel 4
:SOURce:CONF:LIST:STORe "MySourceList"
:SOURce:VOLTage:LEVel 5
:SOURce:CONF:LIST:STORe "MySourceList"
```

This example:

- Creates the measure configuration list `MyMeasList`.
- Sets the measure function to DC voltage.
- Sets the measure range to 100 V.
- Stores the settings to the configuration list `MyMeasList`.
- Sets the measure function to DC current.
- Sets the measure range to 100 mA.
- Stores the settings to the configuration list `MyMeasList`.

Send the following SCPI commands:

```
:SENSe:CONFiguration:LIST:CREate "MyMeasList"
:FUNCtion "VOLTage"
:SENSe:VOLTage:RANGe 100
:SENSe:CONFiguration:LIST:STORe "MyMeasList"
:FUNCtion "CURRent"
:SENSe:CURRent:RANGe 0.1
:SENSe:CONFiguration:LIST:STORe "MyMeasList"
```

Store settings for the active function using TSP commands

This example:

- Creates a source configuration list named `MySourceList`
- Sets the instrument source function to voltage
- Sets the instrument source range to 20 V
- Sets the instrument source limit to 2.0 V
- Stores the settings to `MySourceList`
- Changes the source level to 3.0 V
- Stores the setting to `MySourceList`
- Sets the source level to 4.0 V
- Stores the setting to `MySourceList`
- Sets the source level to 5.0 V
- Stores the setting to `MySourceList`

Send the following TSP commands:

```
smu.source.configlist.create("MySourceList")
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.source.level = 2
smu.source.configlist.store("MySourceList")
smu.source.level = 3
smu.source.configlist.store("MySourceList")
smu.source.level = 4
smu.source.configlist.store("MySourceList")
smu.source.level = 5
smu.source.configlist.store("MySourceList")
```

This example:

- Creates the measure configuration list `MyMeasList`.
- Sets the measure function to DC voltage.
- Sets the measure range to 100 V.
- Stores the settings to the configuration list `MyMeasList`.
- Sets the measure function to DC current.
- Sets the measure range to 100 mA.
- Stores the settings to the configuration list `MyMeasList`.

Send the following TSP commands:

```
smu.measure.configlist.create("MyMeasList")
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 100
smu.measure.configlist.store("MyMeasList")
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 0.1
smu.measure.configlist.store("MyMeasList")
```

Store settings for a function that is not active using TSP commands

You can set up a function that is not active and store it in a configuration list using the `smu.source.configlist.storefunc()` or `smu.measure.configlist.storefunc()` command and the `smu.source.setattribute()` or `smu.measure.setattribute` command. You can retrieve the settings for a function using the `smu.source.getattribute()` or `smu.measure.getattribute` command.

The store function command stores the settings for a specific function into a configuration list. The configuration list must be created before you use the store function command. Refer to smu.source.configlist.storefunc() (on page 14-178) or smu.measure.configlist.storefunc() (on page 14-134) for detail on using the command.

The set attribute command sets a single attribute for the specified function. For details of the command and listings of the parameters that can be set, refer to smu.source.setattribute() (on page 14-190) or smu.measure.setattribute() (on page 14-165).

The following examples demonstrate how to use the store function and set attribute commands. This example:

- Creates a source configuration list named `MySourceList`.
- Sets the instrument source range to 20 V for DC voltage.
- Sets the instrument source limit to 2.0 V for DC voltage.
- Stores the settings to `MySourceList`.
- Changes the source level to 3.0 V for DC voltage.
- Stores the setting to `MySourceList`.
- Sets the source level to 4.0 V for DC voltage.
- Stores the setting to `MySourceList`.
- Sets the source level to 5.0 V for DC voltage.
- Stores the setting to `MySourceList`.

Send the following TSP commands:

```
smu.source.configlist.create("MySourceList")
smu.source.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_SRC_RANGE, 20)
smu.source.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_SRC_LEVEL, 2)
smu.source.configlist.storefunc("MySourceList", smu.FUNC_DC_VOLTAGE)
smu.source.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_SRC_LEVEL, 3)
smu.source.configlist.storefunc("MySourceList", smu.FUNC_DC_VOLTAGE)
smu.source.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_SRC_LEVEL, 4)
smu.source.configlist.storefunc("MySourceList", smu.FUNC_DC_VOLTAGE)
smu.source.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_SRC_LEVEL, 5)
smu.source.configlist.storefunc("MySourceList", smu.FUNC_DC_VOLTAGE)
```

This example:

- Creates the measure configuration list `MyMeasList`.
- Sets the measure range to 100 V for the DC voltage function.
- Stores the settings to the configuration list `MyMeasList`.
- Sets the measure range to 100 mA for the DC current function
- Stores the settings to the configuration list `MyMeasList`.

Send the following TSP commands:

```
smu.measure.configlist.create("MyMeasList")
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_RANGE, 100)
smu.measure.configlist.storefunc("MyMeasList", smu.FUNC_DC_VOLTAGE)
smu.measure.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_MEAS_RANGE, 0.1)
smu.measure.configlist.storefunc("MyMeasList", smu.FUNC_DC_CURRENT)
```

Recall a configuration index

You can recall the settings stored in a specific configuration index in a configuration list.

If you are using remote commands, you can recall an index from a source configuration list and a measure configuration list in a single command.

This example recalls configuration index 2 from `MySourceList` and `MyMeasList`.

NOTE

Recall source configuration lists before measure configuration lists. This order ensures that dependencies between source and measure settings are properly handled.

Using the front panel to recall a source configuration index:

1. Press the **MENU** key.
2. Under Source, select **Config Lists**. The SOURCE CONFIGURATION LISTS screen is displayed.
3. Select **MySourceList**. The configuration indexes in the list are displayed.
4. Select the second configuration index.
5. Select **Recall Index**.

The instrument returns to the settings stored in this configuration list index.

Using the front panel to recall a measure configuration index:

1. Press the **MENU** key.
2. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Choose **Select**. A menu of available configuration lists is displayed.
4. Select **MyMeasList**. The configuration indexes in the list display.
5. Select the second configuration index.
6. Select **Recall Index**.

Using SCPI commands:

To recall index 2 from a source configuration list:

```
:SOURce:CONFiguration:LIST:RECall "MySourceList", 2
```

To recall index 2 from a measure configuration list:

```
:SENSe:CONFiguration:LIST:RECall "MyMeasList", 2
```

To recall index 2 from a source configuration list and a measure configuration list using a single command:

```
:SOURce:CONFiguration:LIST:RECall "MySourceList", 2, "MyMeasList", 2
```

Using TSP commands:

To recall index 2 from a source configuration list:

```
smu.source.configlist.recall("MySourceList", 2)
```

To recall index 2 from a measure configuration list:

```
smu.measure.configlist.recall("MyMeasList", 2)
```

To recall index 2 from a source configuration list and a measure configuration list using a single command:

```
smu.source.configlist.recall("MySourceList", 3, "MyMeasList", 5)
```

View configuration list contents

You can display or print the contents of a specific configuration index. The contents that are returned include all the active settings that the instrument saved when you stored the configuration index.

The following examples demonstrate how to view configuration index 2 from source configuration list `MySourceList` and measure configuration list `MyMeasList`.

View configuration list contents using the front panel

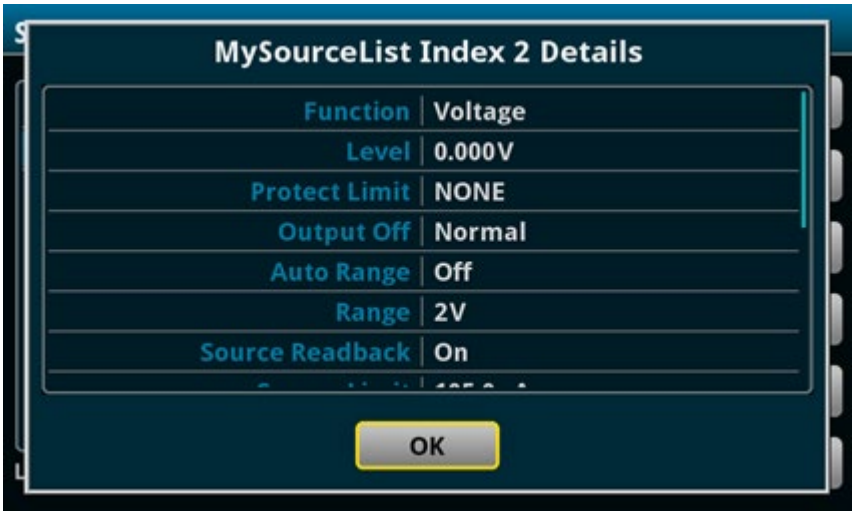
Use the following procedure to view configuration index 2 from source configuration list `MySourceList` and measure configuration list `MyMeasList`.

You can use the Jump to Index option to go to an index in the list.

Use the front panel to view the contents of a source configuration list:

1. Press the **MENU** key.
2. Under **Source**, select **Config Lists**. The SOURCE CONFIGURATION LISTS screen is displayed.
3. Select **MySourceList**. The configuration indexes are displayed.
4. Select the second configuration index.
5. Select **Index Details**. The stored settings are displayed. You can scroll in the dialog box to view additional settings.

Figure 80: Source configuration list index details



Use the front panel to view the contents of a measure configuration list:

1. Press the **MENU** key.
2. Under **Measure**, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **MyMeasList**. The configuration indexes are displayed.
4. Select the second configuration index.
5. Select **Index Details**. The stored settings are displayed. You can scroll in the dialog box to view additional settings.

Figure 81: Measure configuration list index details



6. When you are finished, select **OK**.

Using SCPI commands:

The SCPI configuration list query command returns a list of TSP commands that could be used to set the parameters stored in the specified configuration index.

To view a list of commands in configuration index 2 in a source configuration list named `MySourceList`, send the command:

```
:SOURce:CONFiguration:LIST:QUERy? "MySourceList", 2
```

To view a list of commands in configuration index 2 in a measure configuration list named `MyMeasList`, send the command:

```
:SENSe:CONFiguration:LIST:QUERy? "MyMeasList", 2
```

Using TSP commands:

The TSP configuration list query commands return a list of TSP commands that set the settings stored in the specified configuration index.

To print a list of commands in configuration index 2 in a source configuration list named `MySourceList`, send the command:

```
print(smu.source.configlist.query("MySourceList", 2))
```

To print a list of commands in configuration index 2 in a measure configuration list named `MyMeasList`, send the command:

```
print(smu.measure.configlist.query("MyMeasList", 2))
```

Delete a configuration index or list

You can delete individual configuration indexes from a configuration list using the front panel or remote commands.

When a configuration list index is deleted, the following indexes are renumbered so that the indexes are numbered sequentially.

NOTE

You cannot delete configuration lists from the front panel.

Use the front panel to delete a source configuration index:

1. Press the **MENU** key.
2. Under Source, select **Config Lists**. The SOURCE CONFIGURATION LISTS screen is displayed.
3. Select **MyMeasList**. The configuration indexes are displayed.
4. Select the index.
5. Select **Remove Index**.

Use the front panel to delete a measure configuration index:

1. Press the **MENU** key.
2. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **MyMeasList**. The configuration indexes are displayed.
4. Select the index.
5. Select **Remove Index**.

To use SCPI commands to delete index 2 from a source configuration list named MySourceList:

```
:SOURce:CONFiguration:LIST:DELete "MySourceList", 2
```

To use SCPI commands to delete a source configuration list named MySourceList:

```
:SOURce:CONFiguration:LIST:DELete "MySourceList"
```

To use SCPI commands to delete index 2 from a measure configuration list named MyMeasList:

```
:SENSe:CONFiguration:LIST:DELete "MyMeasList", 2
```

To use SCPI commands to delete a measure configuration list named MyMeasList:

```
:SENSe:CONFiguration:LIST:DELete "MyMeasList"
```

To use TSP commands to delete index 2 from a source configuration list named MySourceList:

```
smu.source.configlist.delete("MySourceList", 2)
```

To use TSP commands to delete a source configuration list named MySourceList:

```
smu.source.configlist.delete("MySourceList")
```

To use TSP commands to delete index 2 from a measure configuration list named MyMeasList:

```
smu.measure.configlist.delete("MyMeasList", 2)
```

To use TSP commands to delete a measure configuration list named MyMeasList:

```
smu.measure.configlist.delete("MyMeasList")
```

Overwrite an existing index

When you are using the front panel, you can overwrite an existing index by selecting the index before selecting **Add Settings**. You are prompted to either overwrite the existing index or append the index to the end of the list.

When you are using remote commands, you can overwrite an existing index by specifying the existing index number in the index parameter of the command.

List the available configuration lists

You can view the names of the configuration lists stored on the instrument.

From the front panel:

On the SOURCE CONFIGURATION LIST or MEASURE CONFIGURATION LIST screen, choose **Select** to display the list of configuration lists.

Using SCPI commands:

To return the name of one source configuration list stored on the instrument, use the following command.

```
:SOURce:CONFiguration:LIST:CATalog?
```

To receive the name of one measure configuration list stored on the instrument, use the command:

```
:SENSe:CONFiguration:LIST:CATalog?
```

Each time this command executes, the name of one defined configuration is returned. To get all defined configuration lists, send this command until it returns an empty string. After the command returns an empty string, it wraps around and starts returning names again. If only an empty string is returned, no configuration lists of the specified type exist.

Using TSP commands:

To return the name of one source configuration list stored on the instrument, use the following command.

```
print(smu.source.configlist.catalog())
```

To receive the name of one measure configuration list stored on the instrument, use the following command.

```
print(smu.measure.configlist.catalog())
```

Each time this command executes, the name of one defined configuration is returned. To get all defined configuration lists, send this command until it returns `nil`. After the command returns `nil`, it wraps around and starts returning names again. If only `nil` is returned, no configuration lists of the specified type exist.

Determine the number of indexes in a configuration list

You can view the number of configuration indexes that are in a configuration list.

Use the front panel to view the number of indexes in a source configuration list:

1. Press the **MENU** key.
2. Under **Source**, select **Config Lists**. The SOURCE CONFIGURATION LISTS screen is displayed.
3. Select a source configuration list.

The number of indexes is displayed below the index list.

Use the front panel to view the number of indexes in a measure configuration list:

1. Press the **MENU** key.
2. Under **Measure**, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select a measure configuration list.

The number of indexes is displayed below the index list. To go to a specific index, you can use the Jump to Index option.

Using SCPI commands:

To view the number of configuration indexes in a source configuration list named `MySourceList`, send the following command:

```
:SOURce:CONFiguration:LIST:SIZE? "MySourceList"
```

To view the number of configuration indexes in a measure configuration list named `MyMeasList`, send the following command:

```
:SENSe:CONFiguration:LIST:SIZE? "MyMeasList"
```

Using TSP commands:

To view the number of configuration indexes in a source configuration list named `MySourceList`, send the following command:

```
smu.source.configlist.size("MySourceList")
```

To view the number of configuration indexes in a measure configuration list named `MyMeasList`, send the following command:

```
smu.measure.configlist.size("MyMeasList")
```

Save a configuration list

Configuration lists are removed when you turn the instrument off and turn it on again or if you reset the instrument.

To save a configuration list, create a configuration script. A configuration script saves the settings of the instrument, including all defined source and measure configuration lists. You can do this using any of the following:

- Front panel option Menu > Save Setup.
- SCPI command `*SAV`
- TSP command `createconfigscript()`

See Saving setups (on page 3-45) for additional information.

Remote commands for configuration list operations

You can use the following remote commands to create and maintain configuration lists.

Action	SCPI command TSP command
Create a configuration list	[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62) smu.measure.configlist.create() (on page 14-128) :SOURce[1]:CONFiguration:LIST:CREate (on page 12-69) smu.source.configlist.create() (on page 14-172)
Restore the settings in one or more configuration lists to the instrument	[:SENSe[1]]:CONFiguration:LIST:RECall (on page 12-64) smu.measure.configlist.recall() (on page 14-131) :SOURce[1]:CONFiguration:LIST:RECall (on page 12-72) smu.source.configlist.recall() (on page 14-175)
View the contents of a configuration list index as TSP commands	[:SENSe[1]]:CONFiguration:LIST:QUERy? (on page 12-63) smu.measure.configlist.query() (on page 14-130) :SOURce[1]:CONFiguration:LIST:QUERy? (on page 12-71) smu.source.configlist.query() (on page 14-174)
Delete a configuration list or an index in a configuration list	[:SENSe[1]]:CONFiguration:LIST:DELete (on page 12-62) smu.measure.configlist.delete() (on page 14-129) :SOURce[1]:CONFiguration:LIST:DELete (on page 12-70) smu.source.configlist.delete() (on page 14-173)
View available configuration lists	[:SENSe[1]]:CONFiguration:LIST:CATalog? (on page 12-61) smu.measure.configlist.catalog() (on page 14-127) :SOURce[1]:CONFiguration:LIST:CATalog? (on page 12-68) smu.source.configlist.catalog() (on page 14-172)
Determine the number of configuration indexes in a configuration list	[:SENSe[1]]:CONFiguration:LIST:SIZE? (on page 12-65) smu.measure.configlist.size() (on page 14-132) :SOURce[1]:CONFiguration:LIST:SIZE? (on page 12-73) smu.source.configlist.size() (on page 14-176)
Save a configuration list	[:SENSe[1]]:CONFiguration:LIST:STORe (on page 12-66) smu.measure.configlist.store() (on page 14-133) :SOURce[1]:CONFiguration:LIST:STORe (on page 12-73) smu.source.configlist.store() (on page 14-177)
Store settings for a function into a configuration list regardless of the active state of the function	No SCPI version smu.source.configlist.storefunc() (on page 14-178) smu.measure.configlist.storefunc() (on page 14-134)
Set up functions regardless of the active state of the function so they can be saved using `smu.source.configlist.storefunc()` and `smu.measure.configlist.storefunc()`	No SCPI version smu.source.setattribute() (on page 14-190) smu.measure.setattribute() (on page 14-165)

Section 5

Source-measure considerations

In this section:

Circuit configurations 5-1
Operating boundaries 5-4
Output transient recovery 5-8
Load regulation 5-8
Using NPLCs to adjust speed and accuracy 5-9
Noise shield 5-11
Safety shield 5-11
Noise and chassis ground 5-12
Floating the 2470 5-13
Guarding 5-15
Sink operation 5-16
Battery charge and discharge 5-17
Measurement settling time considerations 5-18
Overtemperature protection 5-19
Current breakdown protection 5-19
Calculating accuracy 5-20
Offset-compensated ohm calculations 5-22
Power calculations 5-23
High-capacitance operation 5-23
Filtering measurement data 5-24
Order of operations 5-27
Reset default values 5-28

Circuit configurations

You can measure current or voltage with either type of source.

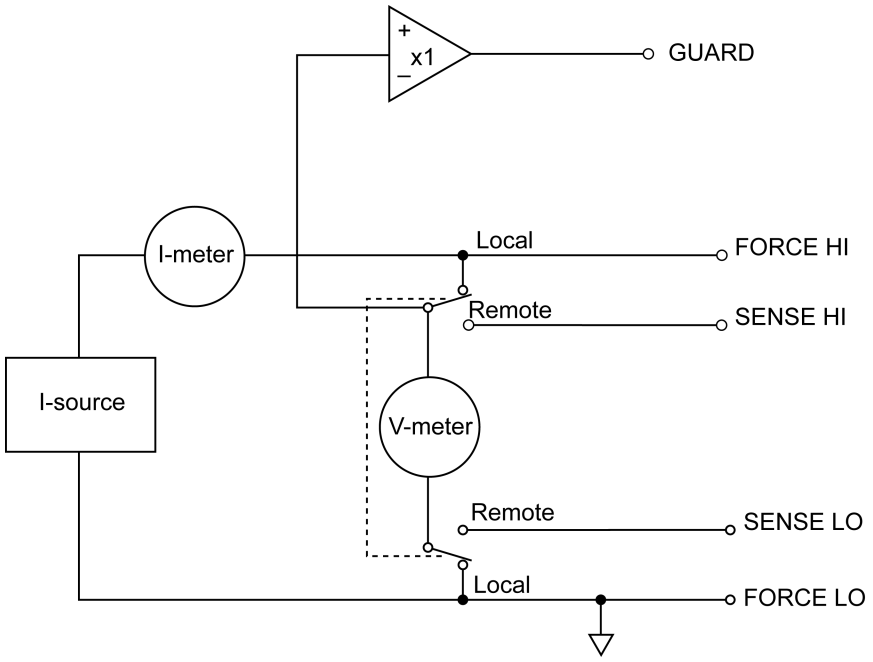
The fundamental source-measure configurations for the 2470 are described in the following section.

Source current

When you configure the instrument to source current, the instrument functions as a high-impedance current source that can limit voltage and can measure current or voltage.

As shown in the figure below, if you set the instrument to 2-wire sense, voltage is measured at the FORCE HI and FORCE LO terminals of the instrument. If you set the instrument to 4-wire sense, voltage is measured directly at the device under test using the sense terminals. Four-wire sense eliminates any voltage drops that may be in the test leads or in the connections between the instrument and the device under test.

Figure 82: Current source configuration



The current source does not use the sense leads to enhance current source accuracy. However, if the instrument is in 4-wire sense, the instrument may reach limit levels if you disconnect the sense leads. When 4-wire sense is selected, you must connect the sense leads. If the sense leads are not connected, incorrect operation will result.

If you are sourcing and measuring the same function (for example, sourcing current and measuring current), the measurement range is the same as the source range. This feature is valuable if you are operating when the source limit has been exceeded. When the source limit has been exceeded, the programmed source value is not reached. Thus, measuring the source lets you measure the actual output level. You can also use the source readback function to measure the source. See Source readback (on page 4-45) for information.

You can set overvoltage protection if there is potential for disconnection of the sense leads. For more information on overvoltage protection, see Overvoltage protection (on page 4-35).

Source voltage

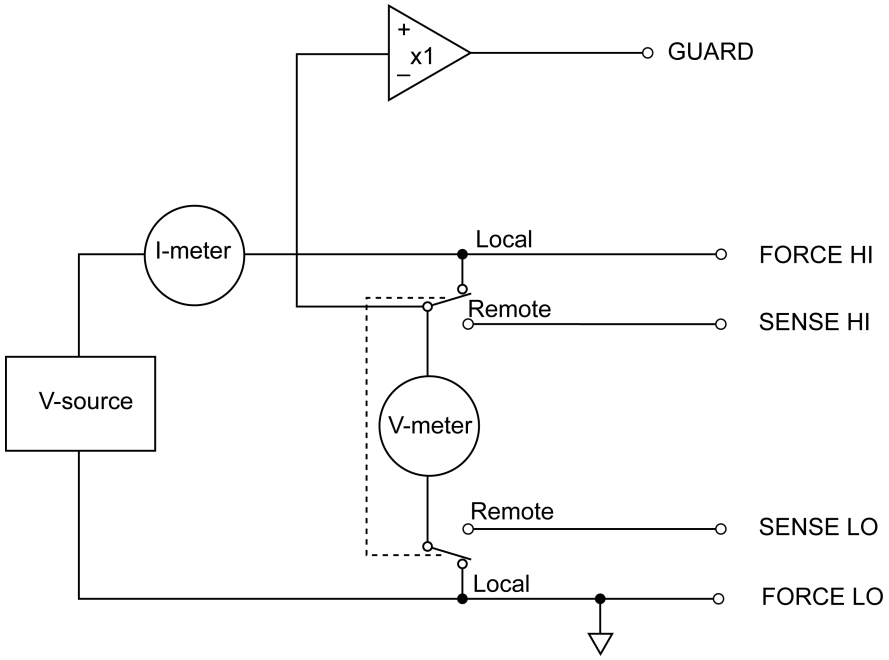
When you configure the instrument to source voltage, it functions like a low-impedance voltage source that can limit current. The instrument can measure current or voltage. This configuration is shown in the figure below.

Sense circuitry continuously monitors the output voltage and makes adjustments to the voltage source as needed. The voltmeter senses the voltage and compares it to the programmed voltage level. If the sensed level and the programmed value are not the same, the source voltage is adjusted accordingly.

If you set the instrument to 2-wire sense, the voltmeter senses the voltage at the FORCE HI and FORCE LO terminals. If it is set to 4-wire sense, the voltmeter sense the voltage at the device under test. Four-wire sense eliminates the effect of voltage drops in the test leads, ensuring that the exact programmed voltage is applied to the device under test.

If you are sourcing and measuring the same function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range. You can use this feature when operating when source limits have been exceeded. When the source limits have been exceeded, the programmed source value is not reached. Thus, measuring the source lets you measure the actual output level. You can also use the source readback function to measure the actual output level. See Source readback (on page 4-45) for information.

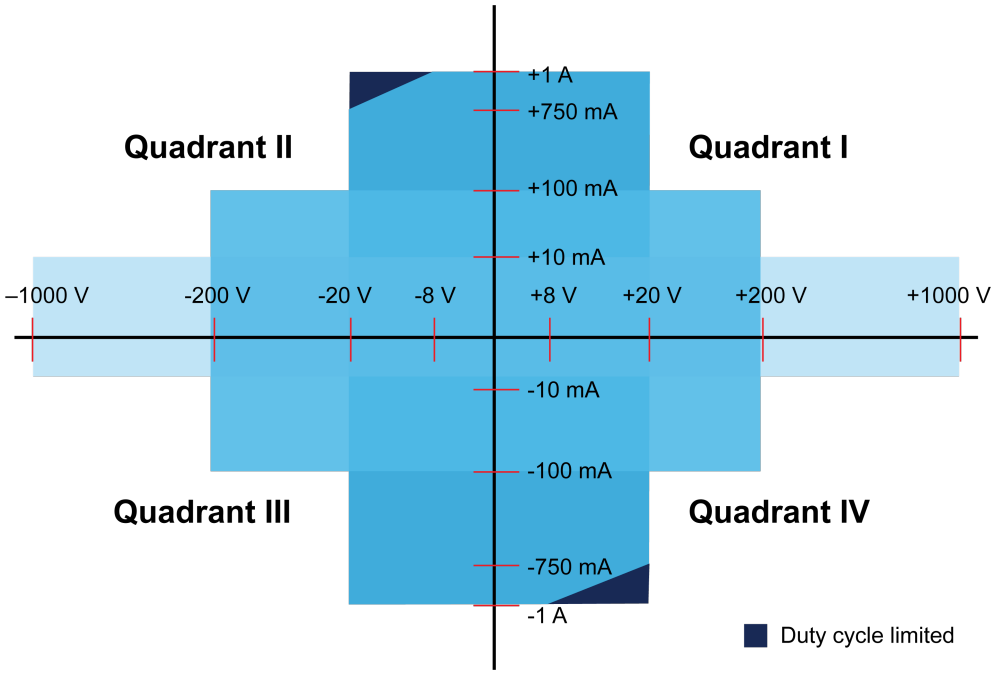
Figure 83: Voltage source configuration



Operating boundaries

Operating boundaries define the current and voltage limits of the instrument. The general operating boundaries of the 2470 are shown in the following figure.

Figure 84: Model 2470 operating boundaries



If the voltage or current exceeds the limits, the instrument limits the source voltage to keep operating currents and voltages in these boundaries. You can set operating limits to restrict current or voltage more tightly than the operating boundary limits. In this drawing, the magnitudes are nominal values. The specific maximum output magnitudes of the instrument are defined in the specifications. Also note that the boundaries are not drawn to scale.

These operating boundaries are valid only if the instrument is being operated in an environment where the ambient temperature is 23 °C or less. Above 23 °C, high power operation could overheat the instrument, causing the output to turn off.

The four quadrants of the operating boundaries are defined as I, II, III, and IV. The 2470 can operate in any of the four quadrants.

When the instrument is operating in quadrant I or III, the instrument is a source, which means that voltage and current have the same polarity. As a source, the instrument is delivering power to a load.

When the instrument is operating in quadrant II or IV, the instrument is operating as a sink, which means that voltage and current have opposite polarity. As a sink, the instrument dissipates the power internally. An external source or an energy storage device, such as a battery, solar cell, or power supply, can force operation in the sink region. The ability of the instrument to dissipate power is defined by the boundaries shown in the following figures.

Current source operating boundaries

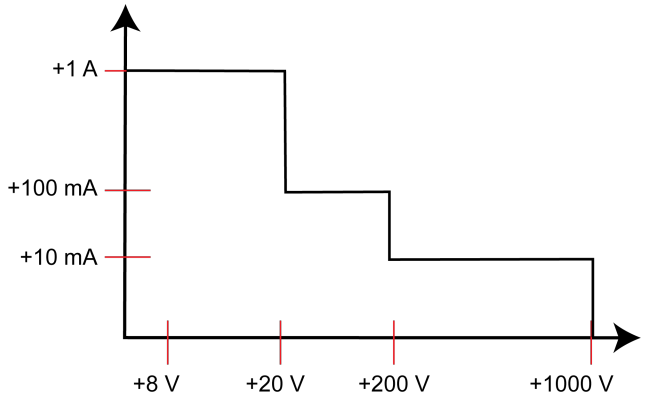
The operating boundaries for the current source are determined by the source range and limit settings. The operating boundary is the lower of the two settings. For example, if the 100 mA current source range is selected, the current source is limited to 105 mA, even if the source limit is set to 1 A.

The voltage limit line represents the actual limit that is in effect. These limit lines are boundaries that represent the operating limits of the instrument for this quadrant of operation.

The operating point can be anywhere in or on these limit lines. The figure below shows operating boundaries for the current source for quadrant I. Operation in the other three quadrants is similar.

The current source line is the maximum source value possible for the presently selected current source range.

Figure 85: 2470 current source output characteristics

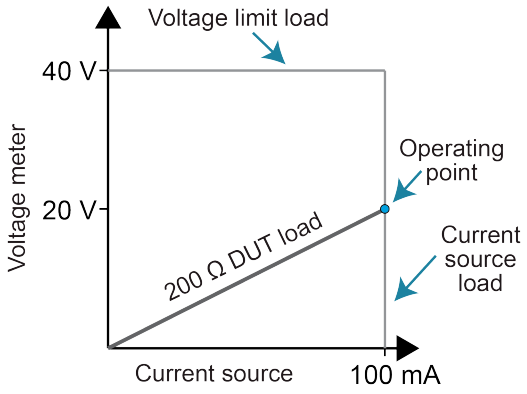


Voltage limit boundary examples

The actual boundaries where the instrument operates depends on the device under test (DUT) that is connected to the output of the instrument.

The following graphs show operation with the instrument set to source 100 mA with a limit of 40 V. In this graph, the resistive load is 200 Ω. The instrument is sourcing 100 mA to the 200 Ω load and subsequently measures 20 V. The load for 200 Ω intersects the 100 mA current source at 20 V.

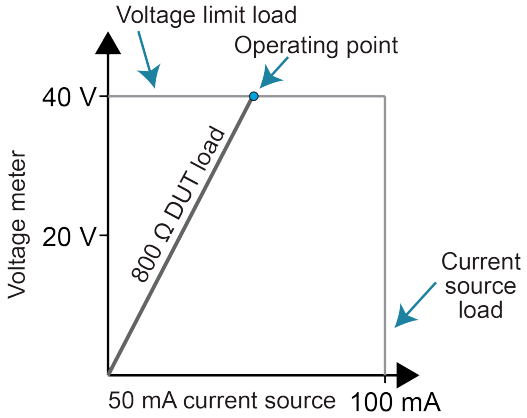
Figure 86: 2470 limit boundary example — normal



Voltage meter = Current source * DUT load
= (100 mA)(200 Ω)
= 20 V

In the following graph, the resistive load is increased to 800 Ω. The DUT load for 800 Ω intersects the voltage limit, which causes the instrument to limit the current that it is sourcing. For the 800 Ω DUT, the instrument will only output 50 mA at the 40 V limit.

Figure 87: 2470 limit boundary example when limited



$$\text{Current source} = \frac{\text{Voltage meter}}{\text{DUT load}}$$
$$= \frac{40\ \text{V}}{800\ \Omega}$$
$$= 50\ \text{mA}$$

Notice that as resistance increases, the slope of the DUT load increases. As resistance approaches infinity (open output), the instrument sources virtually 0 mA at 40 V. Conversely, as resistance decreases, the slope of the DUT load decreases. At zero resistance (shorted output), the instrument sources 100 mA at virtually 0 V. Regardless of the load, voltage will never exceed the limit of 40 V.

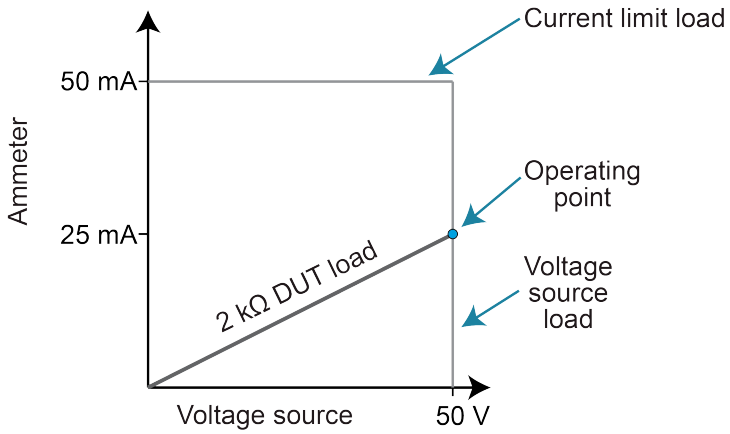
Current limit boundary examples

The actual boundaries where the instrument operates depends on the load (the device under test (DUT)) that is connected to the output of the instrument.

The following graphs show operation with the instrument set to source of 50 V with a limit of 50 mA.

In this graph, the resistive load is 2 kΩ. The instrument is sourcing 50 V to the 2 kΩ load and subsequently measures 25 mA. The load for 2 kΩ intersects the 50 V voltage source at 25 mA.

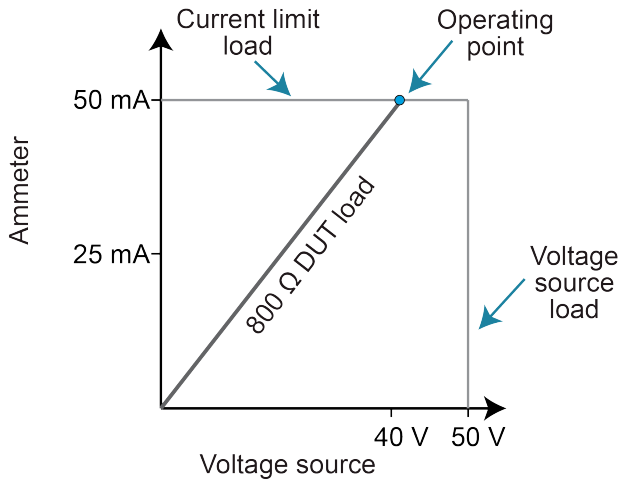
Figure 88: 2470 current limit boundary example normal



$$\text{Current meter} = \frac{\text{Voltage source}}{\text{DUT load}}$$
$$= \frac{50\ \text{V}}{2\ \text{k}\Omega}$$
$$= 25\ \text{mA}$$

In the following graph, the resistive load is decreased to 800 Ω. The DUT load for 800 Ω intersects the current limit, which causes the instrument to limit the voltage that it is sourcing. For the 800 Ω DUT, the instrument will only output 40 V at the 50 mA limit.

Figure 89: 2470 current limit boundary example limited



Voltage source = Current * DUT resistance
= (50 mA)(800 Ω)
= 40 V

Notice that as resistance decreases, the slope of the DUT load increases. Conversely, as resistance increases, the slope of the DUT load decreases. At zero resistance (shorted output), the instrument sources virtually 0 V at 50 mA. Regardless of the load, current never exceeds the limit of 50 mA.

Output transient recovery

The time required for the voltage source to recover to its original value (within 0.1% plus load regulation errors) after a step change in load current is < 250 µs. This does not include the response time of autoranging or the second order effects on loads that are not purely resistive.

Load regulation

The voltage specification for voltage source mode load changes is 0.01% +1 mV. This means that on the 200 mV range, the load current can be changed from zero to full scale with less than 1.02 mV of error. Calculation:

Error = (0.01% x 0.2 V) + 1 mV = 1.02 mV

Assuming a 0 to 1 A change in current, the output impedance equates to
1.02 mΩ (1.02 mV/1 A = 1.02 mΩ). This level can only be achieved using 4-wire remote sensing.

Using NPLCs to adjust speed and accuracy

You can adjust the amount of time that the input signal is measured. Adjustments to the amount of time affect the usable measurement resolution, the amount of reading noise, and the reading rate of the instrument.

The amount of time is specified in parameters that are based on the number of power line cycles (NPLCs). Each power line cycle for 60 Hz is 16.67 ms (1/60); for 50 Hz, it is 20 ms (1/50).

The shortest amount of time results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits. See Setting the number of displayed digits (on page 3-40).

NOTE

The speed setting affects the normal mode rejection ratio (NMRR) and common mode rejection ratio (CMRR). Normal mode noise is the noise signal between the HI and LO terminals; common-mode noise is the noise signal between LO and chassis ground. See the 2470 specification for NMRR and CMRR values at different PLC settings.

To set NPLC using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measure combination.
3. From the home screen, swipe to display the SETTINGS screen.
4. Next to NPLCs, select the number. The number pad dialog box is displayed.
5. Enter the value.
6. Select **OK**.

NOTE

You can also set the speed by pressing the **MENU** key. Under Measure, select **Settings**, and then select the value next to NPLCs.

Using SCPI commands:

To set the number of PLCs for current measurements, send the command:

```
:SENSe:CURRent:NPLCycles <n>
```

To set the number of PLCs for resistance measurements, send the command:

```
:SENSe:RESistance:NPLCycles <n>
```

To set NPLCs for voltage measurements, send the command:

```
:SENSe:VOLTage:NPLCycles <n>
```

Where <n> is a value from 0.01 to 10, with 0.01 resulting in the fastest reading rates and 10 resulting in the lowest reading noise.

For example, to set NPLC for resistance measurements to 0.5, send the command

```
RES:NPLC 0.5
```

Using TSP commands:

To set NPLC, send the command `smu.measure.nplc`. For example, to set the NPLC value to 0.5 for voltage measurements, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.nplc = 0.5
```

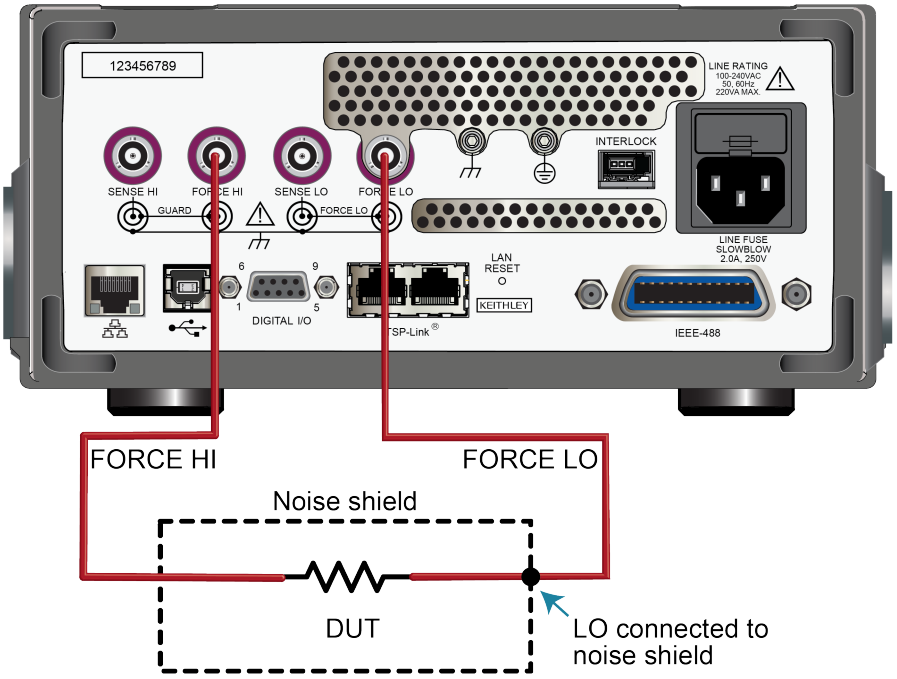
To assign a different measure function, replace `smu.FUNC_DC_VOLTAGE` with one of the following:

- For current measurements: `smu.FUNC_DC_CURRENT`
- For resistance measurements: `smu.FUNC_RESISTANCE`

Noise shield

Use a noise shield to prevent the introduction of unwanted signals into the test circuit. Low-level signals may benefit from effective shielding. The metal noise shield surrounds the test circuit and should be connected to LO, as shown.

Figure 90: 2470 rear-panel noise shield connections



Safety shield

⚠ WARNING

A safety shield must be used whenever hazardous voltages (>30 V_{RMS}, 42 V_{PEAK}) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the 2470 in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield.

The safety shield can be metallic or nonconductive and must completely surround the DUT test circuit. A metal safety shield must be connected to a known protective earth (safety ground). See Test fixtures (on page 4-15) for important safety information on the use of a metal or a nonconductive enclosure.

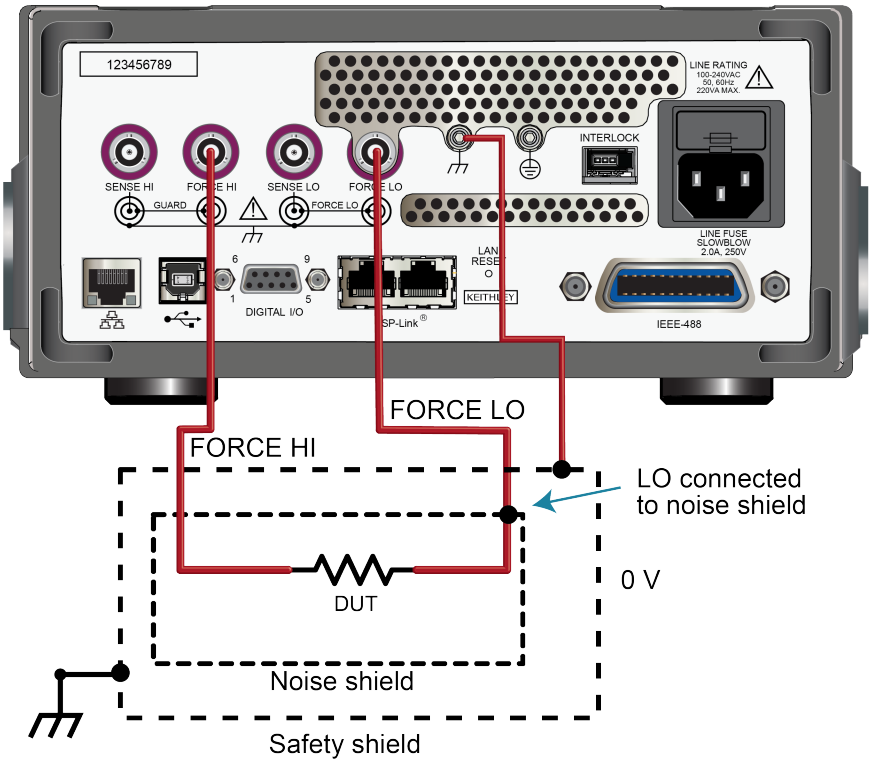
Safety shielding

Use #16 AWG wire or larger for connections to safety earth ground and chassis.

NOTE

For more detail on shielding, refer to the *Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*.

Figure 91: 2470 rear-panel noise and safety shields



Noise and chassis ground

Using the chassis as a ground point for signal connections to the 2470 chassis may result in different levels of noise, depending on your setup. If the 2470 common-mode current is channeled to the chassis instead of the device, the tie point to the chassis can help quiet measurements. However, if other equipment is connected to the chassis, you may have more noise because of other connected equipment.

If you choose to use the chassis as a ground point for signal connections, use the 2470 chassis screw as a connection point.

For more information on preventing noise problems, refer to the Keithley Instruments *Low Level Measurements Handbook*, available on tek.com/keithley.

Floating the 2470

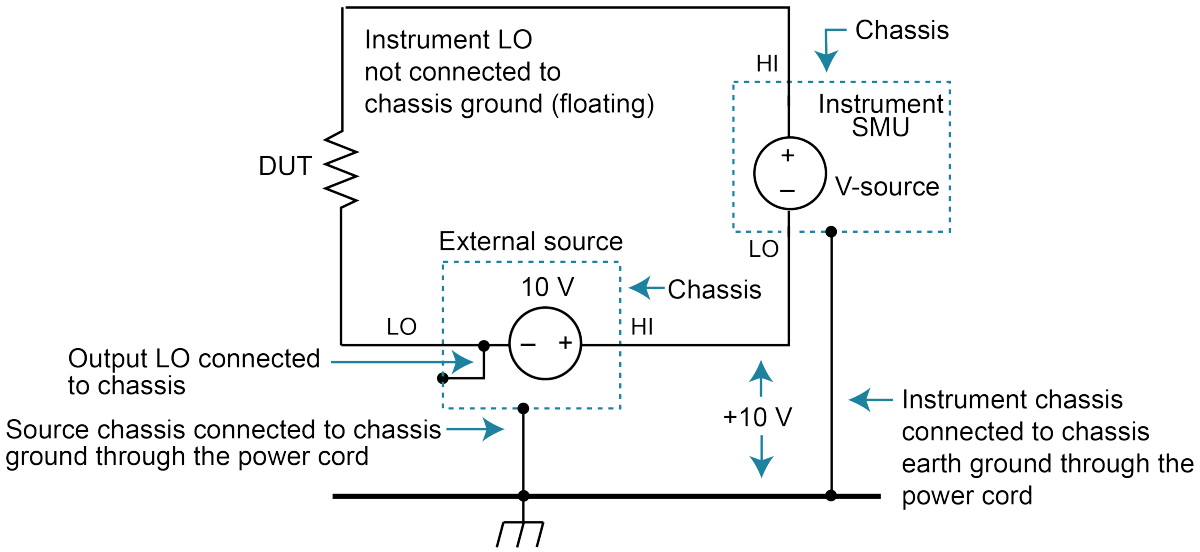
WARNING

SENSE LO and FORCE LO are not internally connected to the chassis. Do not allow them to float above chassis ground more than 250 V. Failure to adhere to these guidelines can result in personal injury or death due to electric shock.

If you use an external source in the test system, you may need the 2470 to float off chassis earth ground. An example is shown below, which includes an external voltage source. Notice that output LO of the external voltage source is connected to chassis ground.

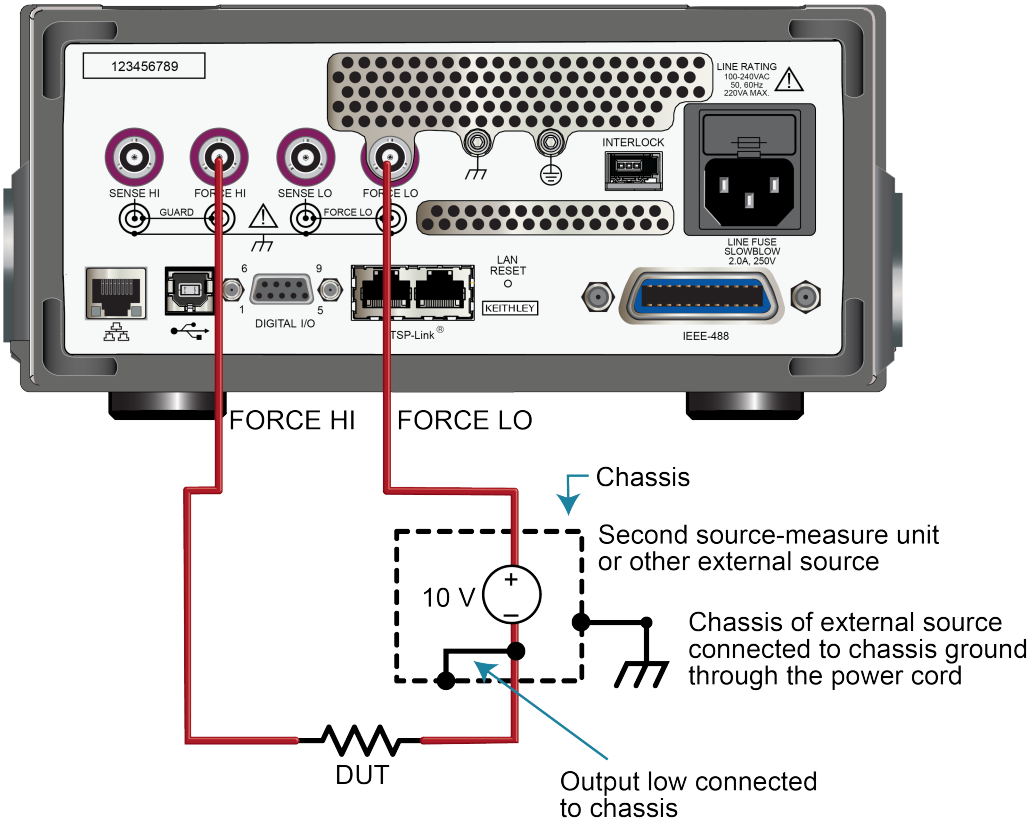
For the test circuit shown below, the 2470 must float off chassis ground. As shown, LO of the 2470 is floating +10 V above chassis earth ground. If LO of the 2470 was instead connected to chassis ground, the external voltage source would be shorted to the chassis ground.

Figure 92: Floating the instrument schematic



The connections for the floating configuration are shown below. To float the SMU, FORCE and SENSE LO must be isolated from chassis ground. To do this, do not connect FORCE and SENSE LO to chassis ground.

Figure 93: Connections for floating the instrument



The external voltage source can be a source-measure unit (SMU) of a second 2470 instrument or other instrument. Keep in mind that if the combined outputs of the sources exceed ±42 V, a safety shield is required for the device under test (DUT). Refer to the following warnings.

⚠ WARNING

All measurement connections should be considered to be hazardous.

The maximum floating (common mode) voltage for a source-measure unit (SMU) is ±250 V. Exceeding this level may cause damage to the instrument and create a shock hazard.

Using an external source to float a SMU could create a shock hazard in the test circuit. A shock hazard exists whenever >42 V_{PEAK} is present in the test circuit. Appropriately rated cables or insulators must be provided for all connections to prevent access to live parts.

When >42 V is present, the test circuit must be insulated for the voltage used or surrounded by a metal safety shield that is connected to a known protective earth (safety ground) and chassis ground; refer to Safety shield (on page 5-11).

Guarding

Use guarding to isolate impedance that you do not want to measure. Guarding is an effective way to reduce the leakage current and capacitance that can exist between HI and LO. A guard is a low impedance point in the circuit that is at nearly the same potential as the high impedance lead that is being guarded. Use guarding when you are sourcing or measuring low current (less than 1 µA) or when test circuit impedance is more than 1 GΩ. Also use guard in noisy environments.

The rear panel of the 2470 includes a current-limited (100 µA minimum) driven cable guard designed to drive cable capacitance at the sense HI and force HI connections. This guard is always enabled and provides a buffered voltage. For 2-wire measurements, guard is at the same level as the force HI voltage. For 4-wire measurements, it is at the same level as the sense HI voltage.

To use the built-in guards of the 2470, you must use the rear-panel triaxial connections. There are no guards available on the front panel.

WARNING

Guard is at the same potential as output HI. Therefore, if hazardous voltages are present at output HI, they are also present at the GUARD terminal. Failure to heed this warning may result in personal injury or death due to electric shock.

Using guard with a test fixture

A test fixture is typically used when testing high-impedance devices. The test fixture reduces noise and protects users from a potentially hazardous voltage on the guard shield.

Inside the test fixture, the guard can be connected to a guard plate or shield that surrounds the device under test (DUT).

Connect the test fixture chassis to LO to reduce noise.

WARNING

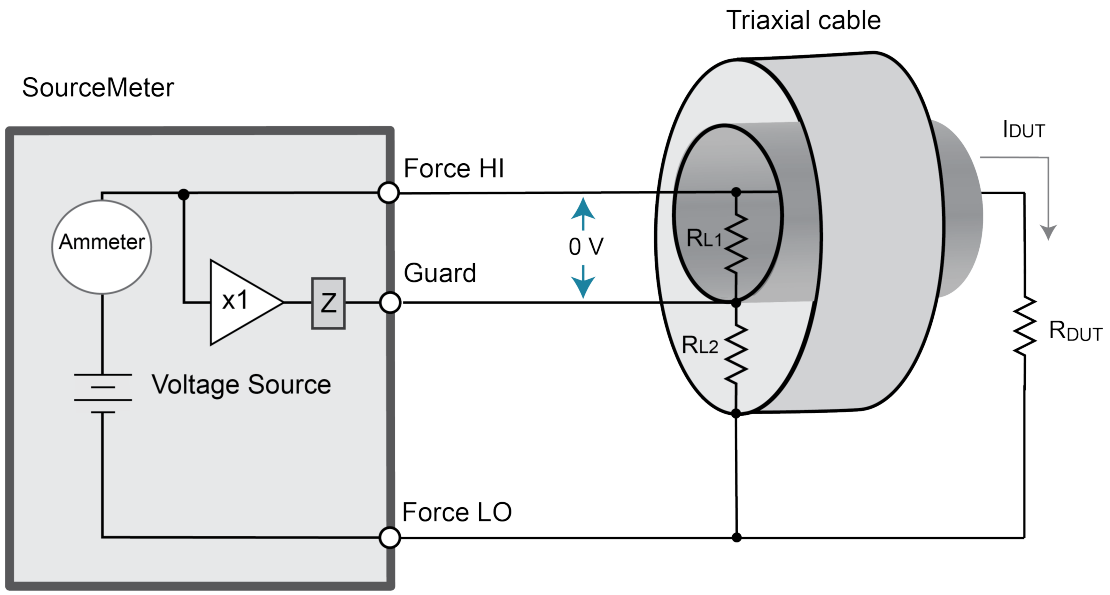
A safety shield must be used whenever hazardous voltages (>30 V_{RMS}, 42 V_{PEAK}) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the 2470 in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield.

Inside the test fixture, a triaxial cable can be used to extend guard to the DUT. See the connection diagrams on the rear panel of the instrument for the triaxial guard to conductor connections.

Guard circuit drawing

In the following schematic, note that guarding eliminates resistance leakage current (R_{L1}). The current flowing in resistance leakage 2 (R_{L2}) is supplied by the guard and does not affect the DUT current (I_{DUT}).

Figure 94: Guarded configuration



Sink operation

When the 2470 is operating as a sink, voltage and current have opposite polarities and the instrument is dissipating power rather than sourcing it. The instrument can be forced into sink operation by an external source, such as a battery, or an energy storage device, such as a capacitor. For detail on the sink region, see Operating boundaries (on page 5-4).

For example, if a 12 V battery is connected to the voltage source (HI to battery high) that is programmed for +10 V, sink operation occurs in the second quadrant (source +V and measure –I).

CAUTION

Carefully consider and configure the output-off state, source, and limits before connecting the 2470 to a device that can deliver energy. Devices that can deliver energy include voltage sources, batteries, capacitors, and solar cells. Configure instrument settings before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

CAUTION

When using the current source as a sink, always set the voltage limit and configure overvoltage protection (OVP) to levels that are higher than the external voltage level. Failure to do so could result in excessive current flow into the 2470 (> 105 mA) and incorrect measurements. When the instrument is operating in limit, be aware that the instrument always tries to maintain the limit.

When the instrument is operating as a sink and you set source or limit values that exceed the operating boundaries, the source limit is reached. When the sink limit is reached, the source value turns yellow and the limit annunciator is active.

Battery charge and discharge

WARNING

To prevent personal injury or damage to the 2470, do not attempt to charge nonrechargeable batteries. Some of the batteries that can be charged with a 2470 are nickel cadmium (Ni-Cd), nickel metal hydride (Ni-MH), lithium ion (Li-ion), rechargeable alkaline, and lead acid. If you are working with a battery type that is not listed here, please contact your local Keithley office, sales partner, or distributor, or call one of our Applications Engineers to get technical assistance.

Always follow the battery manufacturer's requirements for charging or discharging batteries using a 2470. Failure to properly charge or discharge batteries may cause them to leak or explode, resulting in personal injury and property damage. Overvoltage and current protection should be provided in the charge circuit, external to the instrument, when charging batteries without built-in protection.

Do not charge or discharge batteries that exceed 1100 V at 10.5 mA, 210 V at 105 mA, or 21 V at 1.05 A.

Charging

A battery is usually charged using a constant current. To do this, use a 2470 as a voltage source set to the voltage rating of the battery, with the target charging current set as the current limit. At the start of the test, the battery voltage is less than the voltage output setting of the 2470. As a result, this voltage difference drives a current that is immediately limited to the user-defined current limit. When in current limit, the 2470 acts as a constant current source until it reaches the programmed voltage level. As the battery becomes fully charged, the current decreases until it reaches zero or near zero. To prevent safety hazards or damage to the battery, be careful not to overcharge the battery.

Discharging

When discharging a battery, the 2470 operates as a sink because it is dissipating power instead of sourcing it. The voltage source of the 2470 is set to a lower level than the battery voltage. The current limit sets the discharge rate. When the output is enabled, the current from the battery flows into the HI terminal of the 2470. As a result, the current readings are negative. The discharge current should stay constant until the battery voltage decreases to the voltage source setting of the 2470.

CAUTION

Make sure the external voltage never exceeds the voltage limit setting of the current source. This causes excessive current to be drawn from the external battery or source.

If you are using the current source to charge or discharge batteries, the following precautions must be observed. Failure to observe these precautions could result in instrument damage that is not covered by the warranty.

Be sure to set the output-off state of the current source for high impedance. This setting opens the output relay when the output is turned off. With the normal output-off state selected, turning the output off sets the voltage limit to zero. This 0 V source limit condition will cause excessive current to be drawn from the external battery or source.

Carefully consider and configure the output-off state, source, and limits before connecting the 2470 to a device that can deliver energy. Devices that can deliver energy include voltage sources, batteries, capacitors, and solar cells. Configure instrument settings before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

When using the current source as a sink, always set the voltage limit and configure overvoltage protection (OVP) to levels that are higher than the external voltage level. Failure to do so could result in excessive current flow into the 2470 (> 105 mA) and incorrect measurements. When the instrument is operating in limit, be aware that the instrument always tries to maintain the limit.

Measurement settling time considerations

Several outside factors can influence measurement settling times. Effects such as dielectric absorption, cable leakages, and noise can all extend the times required to make stable measurements. Be sure to use appropriate shielding, guarding, and aperture selections when making low-current measurements.

Each current measurement range has a combination of a range resistor and a compensating capacitor that must settle out to allow a stable measurement.

Overtemperature protection

To prevent damaging heat build-up and ensure specified performance, make sure there is adequate ventilation and air flow around the instrument to ensure proper cooling. Do not cover the ventilation holes on the top, sides, or bottom of the instrument.

Even with proper ventilation, the instrument can overheat in the following situations:

- If the ambient temperature is too high.
- If you use the instrument as a power sink for long periods.

If the instrument overheats, the output is turned off and an event message is displayed.

CAUTION

If an overtemperature condition occurs, turn off the instrument and allow it to cool for 30 minutes. You cannot turn the output on until the instrument cools down. Verify that there is adequate ventilation. When you return power to the instrument, verify that the cooling fan is running. If not, contact Keithley Instruments. Leaving the instrument turned on with the failure message displayed or with an inoperative cooling fan may result in damage to the instrument.

Current breakdown protection

In some tests, the limited bandwidth of the SMU may cause current to exceed either the programmed current or the limit current value. To prevent this from occurring, you can turn on the breakdown protection function. This adds a 500 Ω resistor in series with the SMU force lead. This resistor limits, at a wide bandwidth, the breakdown current to a maximum value of $V_{OUTPUT}/500$.

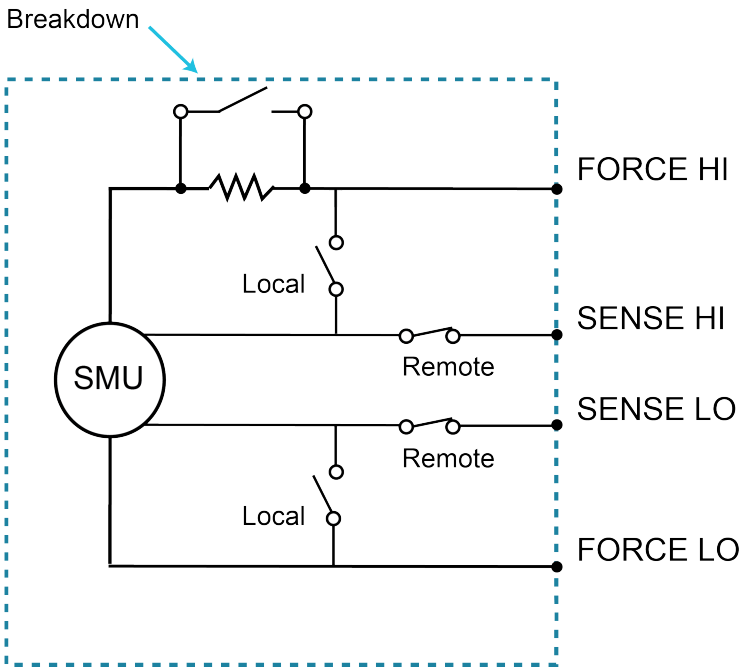
When the breakdown protection is set for AUTO operation, the resistor is in place for the 200 V and 1000 V ranges and current ranges less than or equal to the 10 mA range. Above the 10 mA range or on lower voltage ranges, the breakdown protection resistor is automatically taken out of series with the SMU force lead.

An example of when breakdown protection is appropriate is when you are testing components to specify the DUT breakdown voltage. One method is to test the component at the specified breakdown voltage and determine if the leakage current has exceeded a threshold level. For example, if you are testing a 1000 V rated MOSFET, you can short the gate to the source and apply 1050 V from the drain to the source while measuring the actual drain current. Testing at 1050 V instead of 1000 V provides some assurance that the component both meets and exceeds the breakdown requirement by some user-specified margin. In this test, if the breakdown protection is off, you may find that at the exact moment of component breakdown, the current may exceed the limit current value. With the breakdown function on, the peak current is limited to $V_{OUTPUT}/500$.

A more comprehensive method of testing components to specify the DUT breakdown voltage is to measure the actual component breakdown voltage. To do this on a 1000 V rated MOSFET, you need to switch the sourcing method to current and the limit voltage to 1100 V (higher than the highest expected breakdown voltage). Sourcing the value of the breakdown current permits the SMU to measure the actual breakdown voltage of the component, which occurs precisely when the SMU reaches the programmed source current. When this operating point is reached, a voltage measurement records the actual breakdown voltage of the component, after which the small source current used to find this breakdown voltage can be reduced to zero while waiting for the next component to test. The entire test should be done quickly with the lowest possible breakdown current to limit device self-heating. When performing this test, if the breakdown protection is off, you may find that at the exact moment of component breakdown, the current may exceed the limit current value. With the breakdown function on, the peak current is limited to V_{OUTPUT}/500.

A simplified schematic of the current breakdown protection is shown in the following figure.

Figure 95: Current breakdown protection schematic



The options for breakdown protection are only available using remote commands. To set it using a SCPI command, refer to :SYSTem:BREakdown:PROTection (on page 12-105). To set it using a TSP command, refer to smu.breakdownprotection (on page 14-118).

Calculating accuracy

Instrument accuracy specifications can be expressed in a variety of ways. To illustrate how to calculate measurement errors from instrument specifications, the following topics provide examples of calculations.

Calculating source or measure accuracy

The source and measure accuracy specifications are expressed as a percent of reading or source value and an offset. To calculate source accuracy, use the formula:

Accuracy = ± (% *of reading* + *offset)*

For example, assuming:

- Current output = 100 mA on 100 mA range
- Accuracy specification = ± (0.025% of output + 15 µA)

Calculate the current source accuracy as shown in the following equations.

Error = {(100 mA * 0.00025) + 15 µA}
= ± {25 µA + 15 µA}
= ± 40 µA

Thus, the current output in this example could fall anywhere within the range of 100 mA ± 40 µA, an uncertainty of ± 0.04%.

Calculate accuracy of a resistance measurement made by sourcing I and measuring V

This example shows how to use the summation method to calculate the accuracy of a resistance measurement made by sourcing current and measuring voltage. With this method, the accuracy of the source and measurements are found separately and then added together.

Device to be measured = 20 Ω resistor using 100 mA test current

Current source accuracy:

Current output = 100 mA on 100 mA range

Accuracy specification = ± (0.025% of output + 15 µA)

Error = ± {(100 mA x 0.00025) + 15 µA}
= ± {25 µA + 15 µA}
= ±40 µA

Error % = ±0.040%

Voltage measure accuracy:

Input signal = (20 Ω x 100 mA) = 2 V

Accuracy specification of 2 V range = ± (0.012% of output + 300 µV)

= ± {(2 V x 0.00012) + 300 µV}
= ± {240 µV + 300 µV}
= ± 540 µV

Error % = ±0.027%

Total measurement uncertainty = 0.04% + 0.027% = 0.067%

For higher accuracy measurements when using SMU Instruments, use the source readback function to actually measure the source output. Use the measured source value to calculate the resistance. To calculate the total accuracy from this example using source readback, add the current and voltage measurement accuracy specifications.

Current measure accuracy:

Input signal = 100 mA on 100 mA range

Accuracy specification of 100 mA measurement range = ± (0.025% of reading + 6 µA)

= ± {100 mA x 0.00025) +6 µA}
= ± {25 µA + 6 µA}
= ±31 µA

Error % = ±0.031%

Total measurement uncertainty using source readback = ± (0.031% + 0.027%) = ±0.058%

Notice the total uncertainty of 0.058% when measuring the output of the current source is much better than using the current source output specification to calculate the total error (±0.067%).

Offset-compensated ohm calculations

NOTE

Instrument operations, including offset-compensated ohms, are performed on the input signal in a sequential manner.

For a normal resistance measurement, the 2470 sources a current (I) and measures the voltage (V). The resistance (R) is then calculated as (R = V/I) and the reading is displayed.

For offset-compensated ohms, two measurements are performed: One normal resistance measurement and one measurement using the lowest current source setting.

The offset-compensated ohms reading is then calculated as follows:

$$\text{Offset-compensated } \Omega = \frac{\Delta V}{\Delta I}$$

where:

$\Delta V = V_2 - V_1$

$\Delta I = I_2 - I_1$

- V_1 is the voltage measurement with the current source at its normal level.
- V_2 is the voltage measurement using the lowest current source setting.
- I_1 is the current measurement with the source set to a specific level.
- I_2 is the current measurement with the source set to zero.

This 2-point measurement process and reading calculation eliminates the resistance contributed by the presence of V_{EMF}.

When the source is turned on, the output cycles between the programmed value and zero (0 A or 0 V) to derive the offset-compensated ohms measurement.

Power calculations

Power readings are calculated from the measured current and voltage as follows:

$$P = V \times I$$

Where:

P is the calculated power

V is the measured voltage

I is the measured current

High-capacitance operation

The 2470 high-capacitance mode can prevent problems when you are measuring low current and driving a capacitive load. In this situation, you may see overshoot, ringing, and instability. This occurs because the pole formed by the load capacitance and the current range resistor can cause a phase shift in the voltage-control loop of the instrument.

The actual operating conditions for a given capacitive load can vary. This is due to the large dynamic range of the current measurement capability and wide range of internal resistors in the instrument.

Some test applications require capacitors larger than 20 nF. In these applications, you can use the high-capacitance mode to minimize overshoot, ringing, and instability.

Enabling the high capacitance feature

Before enabling high-capacitance mode, note the following:

- Test the device under test (DUT) and the capacitor to determine the best current limit and range of output voltages.
- The settling times can vary based on the DUT. It is important to test the limits of the DUT before you use high-capacitance mode.
- Failure to test the DUT for the appropriate current limit and output voltages can result in damage to or destruction of the DUT.
- For optimal performance, do not continuously switch between normal mode and high-capacitance mode.
- Before you charge the capacitor, start with 0 (zero) voltage across the capacitor.

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Next to High Capacitance, select **On**.
4. Select **HOME** to return to the operating display.

Using SCPI commands:

Send the command:

```
:SOURce:CURRent:HIGH:CAPacitance ON
```

To turn on high capacitance for a voltage source, replace `CURRent` with `VOLTage`.

Using TSP commands:

Set the source function, then send the command:

```
smu.source.highc = smu.ON
```

Filtering measurement data

Filters allow you to produce one averaged sample from a number of measurements. In situations where you have noise levels that fluctuate above and below the measured signal, this can help you produce more accurate measurements.

If you create test algorithms and you are using the averaging filters, make sure the algorithms clear the filter memory stacks at appropriate times to avoid averaging an inappropriate set of measurements.

When the filter is turned on, the filter is applied before any relative offset, math, or limit operations. Once the relative offset is applied, the next filtered reading has the relative offset applied before it is reported to the instrument. This means that when you use relative offset, the next reading may not be zero.

For example, if the filter size is set to 10, ten internal measurements are stored. Once the tenth measurement is made, the display or remote interface updates and returns the average of the ten readings.

For additional information about the order in which math, filters, offsets, and limits are applied, see Order of operations (on page 5-27).

Repeating average filter

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed, and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced.

Moving average filter

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

Note that when the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements.

For example, if the filter size is four, the first measurement is copied to all four stack locations. Therefore, (Reading1 + Reading1 + Reading1 + Reading1)/4. The display and remote interface update after the first reading. With each additional measurement, the average updates:

(Reading2 + Reading1 + Reading1 + Reading1)/4

(Reading3 + Reading2 + Reading1 + Reading1)/4

(Reading4 + Reading3 + Reading2 + Reading1)/4

Do not use the moving average filter when performing a sweep in which source levels are being changed. You should always use a repeating average filter in this case.

Setting up the averaging filter

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Calculations**.
3. For Filter, select **On** to enable filtering.
4. Select **Settings**.
5. For the Filter Type, select **Moving** or **Repeat**.
6. For the Filter Count, enter the number of measurements to be made for each averaged measurement sample.
7. Select **OK**.
8. Select **HOME** to return to the home screen to view the measurements with the filter applied.

NOTE

Once the filter is set up, you can enable and disable the filter from the SETTINGS swipe screen. When filtering is enabled, the FILT indicator on the home screen is lit.

Using SCPI commands:

To set the averaging filters using SCPI commands, refer to the following command descriptions:

[:SENSe[1]]:<function>:AVERage:COUNt (on page 12-43)
[:SENSe[1]]:<function>:AVERage[:STATe] (on page 12-44)
[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 12-45)

Using TSP commands:

To set the averaging filters using TSP commands, refer to the following command descriptions:

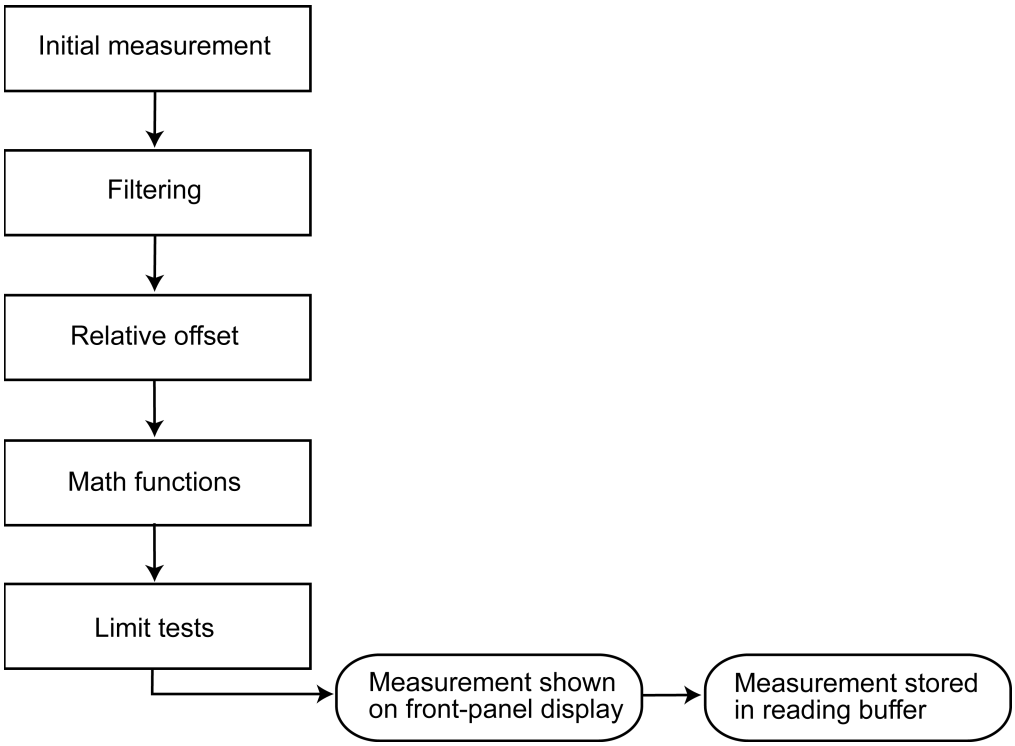
smu.measure.filter.count (on page 14-139)
smu.measure.filter.enable (on page 14-139)
smu.measure.filter.type (on page 14-140)

Order of operations

The measurements have filtering, relative offset values, math operations, and limit testing applied to them in a predetermined order. The measurements that are displayed on the front panel and stored in the reading buffers represent the measurements with any selected operations applied to them.

These operations are applied to the measurement as shown in the following figure.

Figure 96: 2470 order of operations



For more information on these operations, see the following topics:

- Filtering measurement data (on page 5-24)
- Relative offset (on page 4-47)
- Calculations that you can apply to measurements (on page 4-50)
- Limit testing and binning (on page 4-66)

Reset default values

When you turn instrument power on and off or send a reset command, many of the settings in the instrument are reset to their default values.

The settings that are affected are listed in the following tables. The tables show SCPI, TSP, and front-panel settings for each setting. They are sorted alphabetically by the name of the SCPI command.

Default values

Math and limit reset values

Setting	Default value on reset
MENU > Measure > Calculations > Filter > Config > Filter Count [:SENSe[1]]:<function>:AVERage:COUNt (on page 12-43) smu.measure.filter.count (on page 14-139)	10
MENU > Measure > Calculations > Filter [:SENSe[1]]:<function>:AVERage[:STATe] (on page 12-44) smu.measure.filter.enable (on page 14-139)	Off
MENU > Measure > Calculations > Filter > Config > Filter Type [:SENSe[1]]:<function>:AVERage:TCONtrol (on page 12-45) smu.measure.filter.type (on page 14-140)	Repeat
MENU > Measure > Calculations > Math > Config :CALCulate[1]:<function>:MATH:FORMat (on page 12-10) smu.measure.math.format (on page 14-152)	Percent
MENU > Measure > Calculations > Math > Config > Math Format > mx+b > b (Offset) :CALCulate[1]:<function>:MATH:MBFactor (on page 12-12) smu.measure.math.mxb.bfactor (on page 14-153)	0
MENU > Measure > Calculations > Math > Config > Math Format > mx+b >m (Scalar) :CALCulate[1]:<function>:MATH:MMFactor (on page 12-13) smu.measure.math.mxb.mfactor (on page 14-154)	1
MENU > Measure > Calculations > Math > Config > Math Format > Zero Reference :CALCulate[1]:<function>:MATH:PERCent (on page 12-15) smu.measure.math.percent (on page 14-155)	1.0
MENU > Measure > Calculations > Math :CALCulate[1]:<function>:MATH:STATe (on page 12-16) smu.measure.math.enable (on page 14-151)	Off
MENU > Measure > Calculations > Limit1 > Auto Clear :CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 12-18) smu.measure.limit[Y].autoclear (on page 14-144)	On
MENU > Measure > Calculations > Limit1 > Low Value :CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 12-21) smu.measure.limit[Y].low.value (on page 14-150)	–1
MENU > Measure > Calculations > Limit1 > High Value :CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 12-23) smu.measure.limit[Y].high.value (on page 14-149)	1

MENU > Measure > Calculations > Limit1 > Audible :CALCulate2:<function>:LIMit<Y>:AUDible (on page 12-17) smu.measure.limit[Y].audible (on page 14-143)	None
MENU > Measure > Calculations > Limit2 > Auto Clear :CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 12-18) smu.measure.limit[Y].autoclear (on page 14-144)	On
MENU > Measure > Calculations > Limit2 > Low Value :CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 12-21) smu.measure.limit[Y].low.value (on page 14-150)	–1
MENU > Measure > Calculations > Limit2 > High Value :CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 12-23) smu.measure.limit[Y].high.value (on page 14-149)	1
MENU > Measure > Calculations > Limit2 > Audible :CALCulate2:<function>:LIMit<Y>:AUDible (on page 12-17) smu.measure.limit[Y].audible (on page 14-143)	None

Digital I/O reset values

Setting	**Default value on reset**
Not available from front panel :DIGital:LINE<n>:MODE (on page 12-24) digio.line[N].mode (on page 14-58)	Digital line, input

Display reset values

Setting	**Default value on reset**
SETTINGS swipe screen > Display Digits :DISPlay:<function>:DIGits (on page 12-29) smu.measure.displaydigits (on page 14-138)	5½

Format reset values

Setting	**Default value on reset**
Not available from front panel :FORMat:BORDer (on page 12-35) format.byteorder (on page 14-89)	Swapped Little endian
Not available from front panel :FORMat[:DATA] (on page 12-36) format.data (on page 14-90)	ASCII
Not available from front panel :FORMat:ASCii:PRECision (on page 12-34) format.asciiprecision (on page 14-88)	Automatic

Localnode reset values

Setting	Default value on reset
Not available from front panel Not applicable for SCPI localnode.prompts (on page 14-102)	Disabled
Not available from front panel Not applicable for SCPI localnode.prompts4882 (on page 14-103)	Enabled
Not available from front panel Not applicable for SCPI localnode.showevents (on page 14-105)	0

Output reset values

Setting	Default value on reset
MENU > Source > Settings > Output Off :OUTPut[1]:<function>:SMODe (on page 12-37) smu.source.offmode (on page 14-183)	Normal
OUTPUT ON/OFF switch :OUTPut[1][:STATe] (on page 12-41) smu.source.output (on page 14-185)	Off

Terminal reset values

Setting	Default value on reset
TERMINALS switch :ROUTe:TERMinals (on page 12-41) smu.terminals (on page 14-203)	Front

Measurement reset values

Setting	Default value on reset
Menu > Measure > Config List > New List [:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62) smu.measure.configlist.create() (on page 14-128)	None (existing configuration lists deleted)
Menu > Measure > Config List > System to Index [:SENSe[1]]:CONFiguration:LIST:STORe (on page 12-66) smu.measure.configlist.store() (on page 14-133)	None (existing configuration lists deleted)
Menu > Measure > Settings > Count [:SENSe[1]]:COUNt (on page 12-67) smu.measure.count (on page 14-135)	1
SETTINGS swipe > Auto Zero [:SENSe[1]]:<function>:AZERo[:STATe] (on page 12-47) smu.measure.autozero.enable (on page 14-126)	On
Not available from front panel [:SENSe[1]]:<function>:DELay:USER<n> (on page 12-48) smu.measure.userdelay[N] (on page 14-168)	0

Setting	Default value on reset
MENU > Measure > Settings > Offset Compensation (when instrument is set to source current and measure resistance) [:SENSe[1]]:<function>:OCOMpensated (on page 12-50) smu.measure.offsetcompensation (on page 14-157)	Off
SETTINGS swipe > NPLCs [:SENSe[1]]:<function>:NPLCycles (on page 12-49) smu.measure.nplc (on page 14-156)	1
FUNCTION key [:SENSe[1]]:FUNCtion[:ON] (on page 12-68) smu.measure.func (on page 14-142)	Current
HOME > Range [:SENSe[1]]:<function>:RANGe:AUTO (on page 12-51) smu.measure.autorange (on page 14-122)	On
MENU > Measure > Settings > Auto Range Low Limit [:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 12-52) smu.measure.autorangelow (on page 14-124)	Current: 10 nA Voltage: 200 mV Resistance: 2 Ω
Not available from front panel [:SENSe[1]]:<function>:RANGe:AUTO:ULIMit (on page 12-54) smu.measure.autorangehigh (on page 14-123)	Resistance: 200 MΩ
HOME > Measure Range [:SENSe[1]]:<function>:RANGe[:UPPer] (on page 12-55) smu.measure.range (on page 14-158)	AUTO
Not available from front panel [:SENSe[1]]:<function>:RELative (on page 12-56) smu.measure.rel.level (on page 14-163)	0
MENU > Measure > Calculations > Rel [:SENSe[1]]:<function>:RELative:STATe (on page 12-58) smu.measure.rel.enable (on page 14-162)	Off
MENU > Measure > Settings > Sense [:SENSe[1]]:<function>:RSENse (on page 12-59) smu.measure.sense (on page 14-164)	2-wire
Not available from front panel [:SENSe[1]]:<function>:UNIT (on page 12-60) smu.measure.unit (on page 14-167)	Current: Amp Voltage: Volt

Source reset values

Setting	Default value on reset
Menu > Source > Config List > New List :SOURce[1]:CONFiguration:LIST:CREate (on page 12-69) smu.source.configlist.create() (on page 14-172)	None (existing configuration lists deleted)
Menu > Source > Config List > System to Index :SOURce[1]:CONFiguration:LIST:STORe (on page 12-73) smu.source.configlist.store() (on page 14-177)	None (existing configuration lists deleted)
Not available from front panel :SOURce[1]:<function>:DELay:AUTO (on page 12-75) smu.source.autodelay (on page 14-171)	On
MENU > Source > Settings > High Capacitance :SOURce[1]:<function>:HIGH:CAPacitance (on page 12-77) smu.source.highc (on page 14-181)	Off
HOME > Source :SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] (on page 12-78) smu.source.level (on page 14-182)	0
HOME > Limit :SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 12-79) smu.source.xlimit.level (on page 14-201)	Current: 105 μA Voltage: 21 V
FUNCTION key :SOURce[1]:FUNCtion[:MODE] (on page 12-80) Not applicable for TSP	Current
MENU > Source > Settings > Overvoltage Protection :SOURce[1]:<function>:PROTection[:LEVel] (on page 12-81) smu.source.protect.level (on page 14-186)	None
MENU > Source > Settings > Source Range :SOURce[1]:<function>:RANGe (on page 12-82) smu.source.range (on page 14-187)	Current: 10 nA Voltage: 200 mV
HOME > Source Range :SOURce[1]:<function>:RANGe:AUTO (on page 12-84) smu.source.autorange (on page 14-170)	On
MENU > Source > Settings > Source Readback :SOURce[1]:<function>:READ:BACK (on page 12-85) smu.source.readback (on page 14-189)	On
FUNCTION key :SOURce[1]:FUNCtion[:MODE] (on page 12-80) smu.source.func (on page 14-180)	Voltage

Buffer reset values

Setting	Default value on reset
Not available from front panel :TRACe:FILL:MODE (on page 12-124) bufferVar.fillmode (on page 14-30)	Default buffers: Continuous User-defined buffers: Once
Not available from front panel :TRACe:LOG:STATe (on page 12-125) bufferVar.logstate (on page 14-33)	Default buffers: ON User-created buffers: OFF

Trigger reset values

Setting	Default value on reset
Not available from front panel :TRIGger:BLENder<n>:MODE (on page 12-146) trigger.blender[N].orenable (on page 14-217)	AND
Not available from front panel :TRIGger:DIGital<n>:IN:EDGE (on page 12-175) trigger.digin[N].edge (on page 14-224)	Falling edge
Not available from front panel :TRIGger:DIGital<n>:OUT:LOGic (on page 12-177) trigger.digout[N].logic (on page 14-227)	Negative
Not available from front panel :TRIGger:DIGital<n>:OUT:PULSewidth (on page 12-177) trigger.digout[N].pulsewidth (on page 14-228)	10e-6 s
Not available from front panel :TRIGger:DIGital<n>:OUT:STIMulus (on page 12-178) trigger.digout[N].stimulus (on page 14-229)	None
Not available from front panel :TRIGger:LAN<n>:IN:EDGE (on page 12-179) trigger.lanin[N].edge (on page 14-231)	Either edge
Not available from front panel :TRIGger:LAN<n>:OUT:IP:ADDRess (on page 12-181) trigger.lanout[N].ipaddress (on page 14-236)	0.0.0.0
Not available from front panel :TRIGger:LAN<n>:OUT:PROTocol (on page 12-183) trigger.lanout[N].protocol (on page 14-237)	TCP
Not available from front panel :TRIGger:LAN<n>:OUT:STIMulus (on page 12-184) trigger.lanout[N].stimulus (on page 14-238)	None
Not available from front panel :TRIGger:TIMer<n>:COUNt (on page 12-197) trigger.timer[N].count (on page 14-279)	1
Not available from front panel :TRIGger:TIMer<n>:DELay (on page 12-199) trigger.timer[N].delay (on page 14-281)	10e-6 s

Not available from front panel :TRIGger:TIMer<n>:STARt:FRACtional (on page 12-199) trigger.timer[N].start.fractionalseconds (on page 14-284)	0
Not available from front panel :TRIGger:TIMer<n>:STARt:GENerate (on page 12-200) trigger.timer[N].start.generate (on page 14-285)	Off
Not available from front panel :TRIGger:TIMer<n>:STARt:SECond (on page 12-201) trigger.timer[N].start.seconds (on page 14-286)	0
Not available from front panel :TRIGger:TIMer<n>:STARt:STIMulus (on page 12-202) trigger.timer[N].start.stimulus (on page 14-287)	No event
Not available from front panel :TRIGger:TIMer<n>:STATe (on page 12-203) trigger.timer[N].enable (on page 14-282)	Off
Not available from front panel Not applicable for SCPI trigger.tsplinkin[N].edge (on page 14-289)	Falling
Not available from front panel Not applicable for SCPI trigger.tsplinkout[N].logic (on page 14-292)	Negative
Not available from front panel Not applicable for SCPI trigger.tsplinkout[N].pulsewidth (on page 14-293)	10e-6 s

TSP-Link and TSP-Net reset values

Setting	**Default value on reset**
Not available from front panel Not applicable for SCPI tsplink.line[N].mode (on page 14-298)	Digital open drain
Not available from front panel Not applicable for SCPI tspnet.timeout (on page 14-311)	20
Not available from front panel Not applicable for SCPI tspnet.tsp.abortonconnect (on page 14-312)	1

Section 6

Reading buffers

In this section:

Introduction to reading buffers 6-1
Getting started with buffers 6-2
Creating buffers 6-5
Setting reading buffer options 6-9
Selecting a buffer 6-14
Viewing and saving buffer content 6-17
Clearing buffers 6-23
Deleting buffers 6-25
Remote buffer operation 6-25
Apply mathematical expressions to reading buffer data 6-33
Using buffers across TSP-Link nodes 6-34

Introduction to reading buffers

Reading buffers capture measurements, ranges, the output state of the instrument, and instrument status. The 2470 has two default reading buffers. You can also create user-defined reading buffers.

Reading buffers provide statistics, including average, minimum, maximum, and standard deviation. If you use SCPI commands over the remote interface, peak-to-peak statistics are also available.

When you create a reading buffer, that buffer becomes the active buffer until you choose a different buffer.

You can perform the following operations on reading buffers from the front panel or a remote interface:

- Configure, store, and recall reading buffers.
- View reading buffer content.
- Choose to store readings in a default reading buffer or the user-defined reading buffers.
- Save reading buffer content to a USB flash drive.
- Set reading buffers to fill once or fill continuously.
- Change the capacity of reading buffers.
- Delete user-defined reading buffers. You cannot delete `defbuffer1` and `defbuffer2`.
- Clear reading buffers.
- Clear the default reading buffers and delete the user-defined reading buffers by turning the instrument off or sending an instrument reset command.

Getting started with buffers

The following sections provide you with information to help you start using reading buffers. The Remote buffer operation (on page 6-25) section provides additional information about accessing the reading buffers with remote commands.

Types of reading buffers

There are two default buffers, `defbuffer1` and `defbuffer2`.

If you do not select a specific buffer, all readings are stored in `defbuffer1`. If you want to store readings in `defbuffer2`, you need to select it.

If you want to store readings in a user-defined buffer, you need to create the buffer. The user-defined buffer is automatically set to be the active buffer. New readings are stored in the active buffer.

For information about default values, see Reset default values (on page 5-28).

For information about writable reading buffers, see Writable reading buffers (on page 6-31).

Effects of reset and power cycle on buffers

The instrument clears the default buffers when a reset command is sent or when the power is turned off and then turned on again.

The instrument deletes all user-defined buffers when a reset command is sent or when the power is turned off and then turned on again.

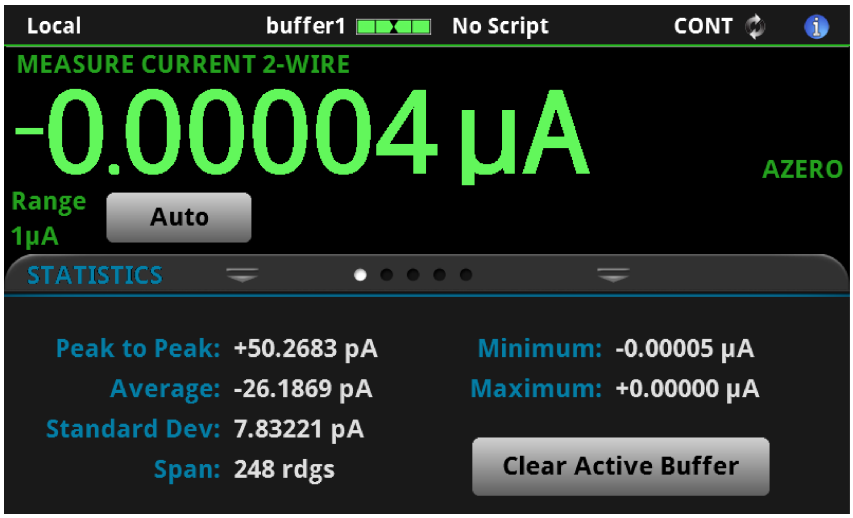
The active buffer is cleared when the function is changed using the front panel.

Buffer fill status

There are several ways to view buffer fill status from the front panel.

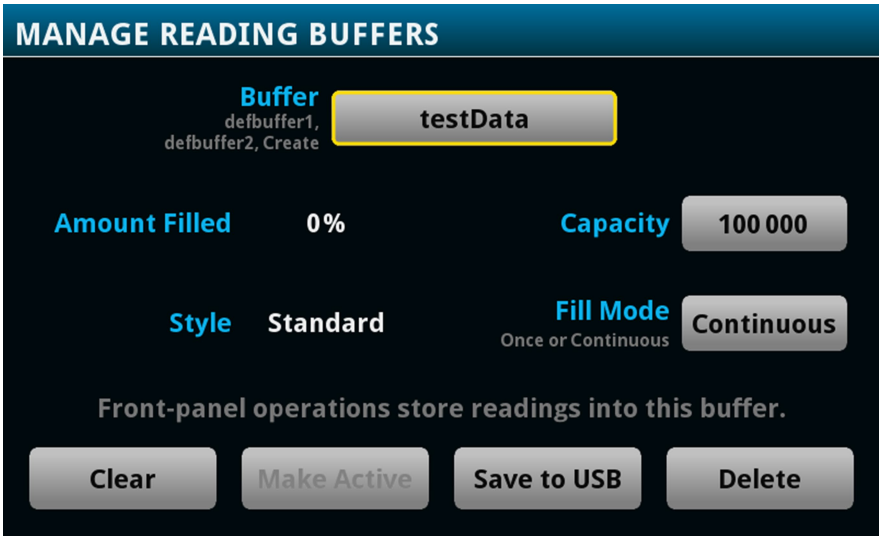
As shown in the following figure, the active buffer indicator on the home screen displays buffer fill status and the STATISTICS swipe screen (on page 3-17) displays buffer statistics. Refer to Active buffer indicator (on page 3-11) for more information on the indicator. To view the STATISTICS swipe screen, swipe the bottom of the screen.

Figure 97: Statistics swipe screen and active buffer indicator



The MANAGE READING BUFFERS window displays buffer fill status as the Amount Filled, as shown in the following figure.

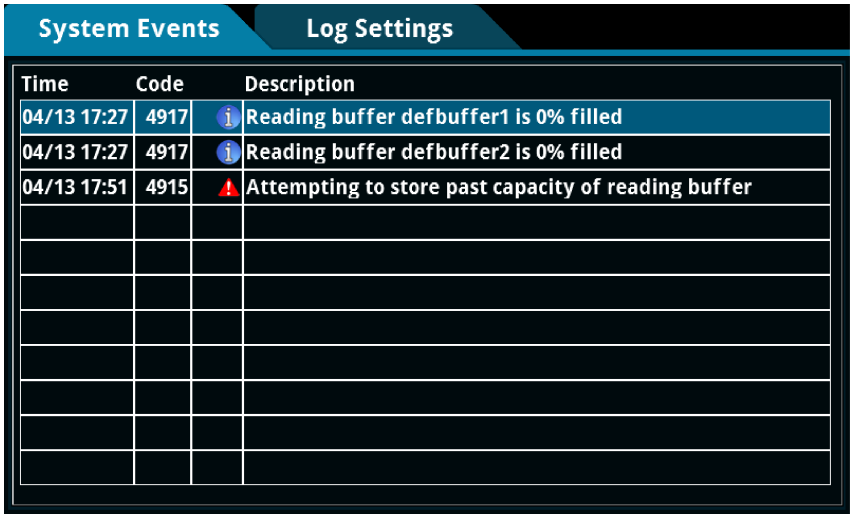
Figure 98: MANAGE READING BUFFERS window



The System Events tab on the System Event Log menu (on page 3-36) displays the following buffer events:

- Event code 4915, "Attempting to store past the capacity of reading buffer," which occurs when a buffer that is set to fill once is full.
- Event code 4917, "The fill status of `defbuffer1` is 0% filled" or "The fill status of `defbuffer2` is 0% filled" if a default buffer is at 0% capacity.
- Event code 4918, "Reading buffer `defbuffer1` is 100% filled" or "Reading buffer `defbuffer2` is 100% filled" if a default buffer is at 100% capacity.

Figure 99: System Events tab



NOTE

Reading buffer status is only reported if the log state is set to on. Refer to the SCPI command description :TRACe:LOG:STATe (on page 12-125) or TSP command description bufferVar.logstate (on page 14-33) for detail on setting the log state for a buffer.

Timestamps

The measurements in the reading buffers contain timestamps. Readings start at the first entry in the empty reading buffer. Readings are then taken sequentially until the end of the buffer is reached. If the buffer fill mode is continuous, readings wrap to the first entry and overwrite the older data. The relative time is taken from the first reading made after a buffer is cleared.

For a buffer that fills once, the first entry starts at index 1 with the timestamp in absolute time. For continuous buffers, the lowest timestamp is after the last entry. For example, if you take 150 readings into a buffer with a capacity of 100, the last reading is at entry 50 and the earliest reading is at 51.

The buffer style you select when creating a buffer affects the resolution of the timestamp. For the Compact buffer style, the timestamp is a 1 µs accuracy relative timestamp with a one-hour time span before the timestamp starts over. For Standard and Full buffer styles, the timestamp is absolute; full date and time is recorded.

Creating buffers

To create a new user-defined reading buffer, you need to provide a name, capacity, and style for the new buffer.

User-defined buffer names must start with an alphabetic character. The names cannot contain any periods or the underscore (_) character. The name can be up to 31 characters long. If you create a reading buffer that has the same name as an existing user-defined buffer, the existing buffer is overwritten by the new buffer. Any data in the existing buffer is lost.

You cannot assign reading buffers the name `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

When you create a buffer, you set a buffer style. The buffer style controls the amount of information that is saved with each reading in the reading buffer. Buffer styles are:

- **Compact:** Store readings with reduced accuracy (6.5 digits) with no formatting information, 1 µs accurate timestamp. Once you store the first reading in a compact buffer, you cannot change certain measurement settings, including range, display digits, and units; you must clear the buffer first.
- **Standard:** Store readings with full accuracy with formatting.
- **Full:** Store the same information as standard, plus additional information.
- **Writable:** Manually write external data to a reading buffer. For more information, see Writable reading buffers (on page 6-31). You cannot select this buffer style from the front panel; you must use remote commands.
- **Full Writable:** Manually write external data to a reading buffer with two values per buffer index. You cannot select this buffer style from the front panel; you must use remote commands.

NOTE

You can only select the style of the reading buffer when you first create the buffer. Not all remote commands are compatible with the compact, writable, and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

There is no fixed limit on the number of user-defined reading buffers you can create. However, you are limited by available memory in the instrument.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you created becomes the active buffer.

The following topics provide information about using the front panel to create buffers and introduce how to use remote commands to create buffers.

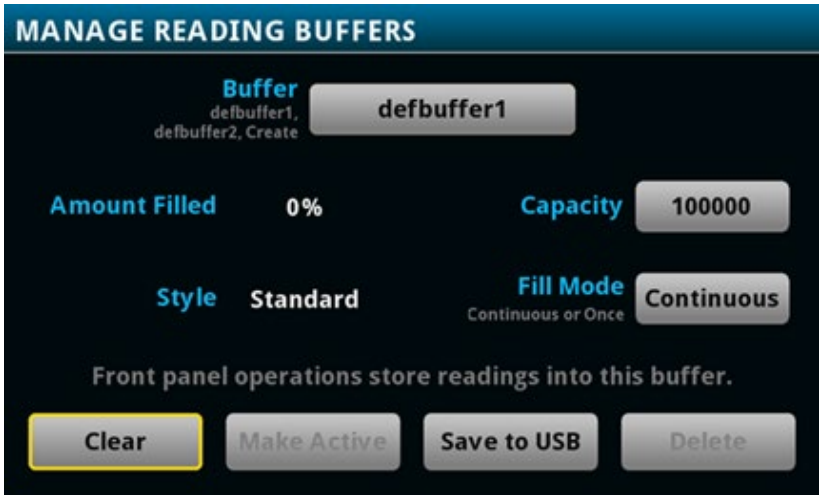
For additional information about using remote commands for buffer operations, see the following sections of this manual:

- Remote buffer operation (on page 6-25)
- SCPI commands, see TRACe subsystem (on page 12-117)
- TSP commands, see TSP commands (on page 14-8)

To use the front panel to create a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 100: MANAGE READING BUFFERS window



3. Select **Buffer**.
4. Select **Create New**. A keyboard is displayed.
5. Enter a name for the buffer you are creating. The following figure uses the name `testData`.

Figure 101: Buffer Name keyboard



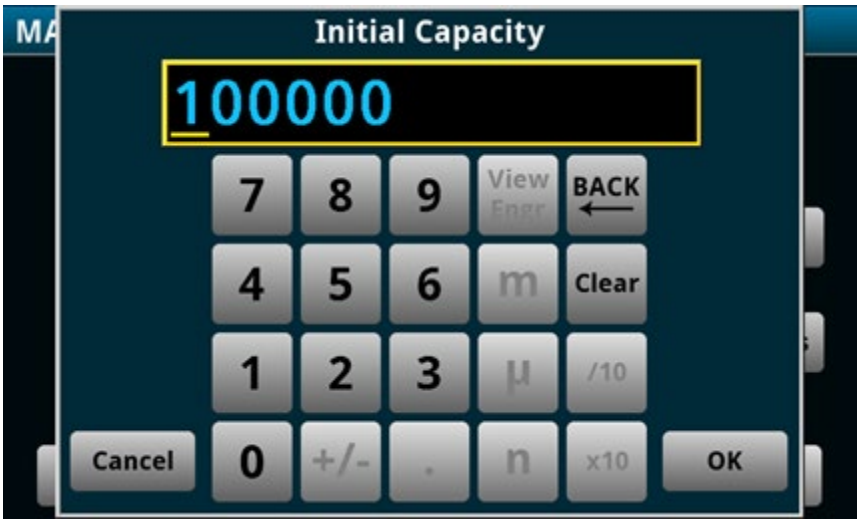
6. Select the **OK** button on the displayed keyboard.
7. The Style dialog box is displayed. You can select:
   - **Standard**: Store readings with full accuracy with formatting.
   - **Compact**: Store readings with reduced accuracy (6.5 digits) with no formatting information, 1 µs accurate timestamp. Once you store the first reading in a compact buffer, you cannot change certain measurement settings, including range, display digits, and units; you must clear the buffer first.
   - **Full**: Store the same information as standard, plus additional information.

Figure 102: Select the buffer style



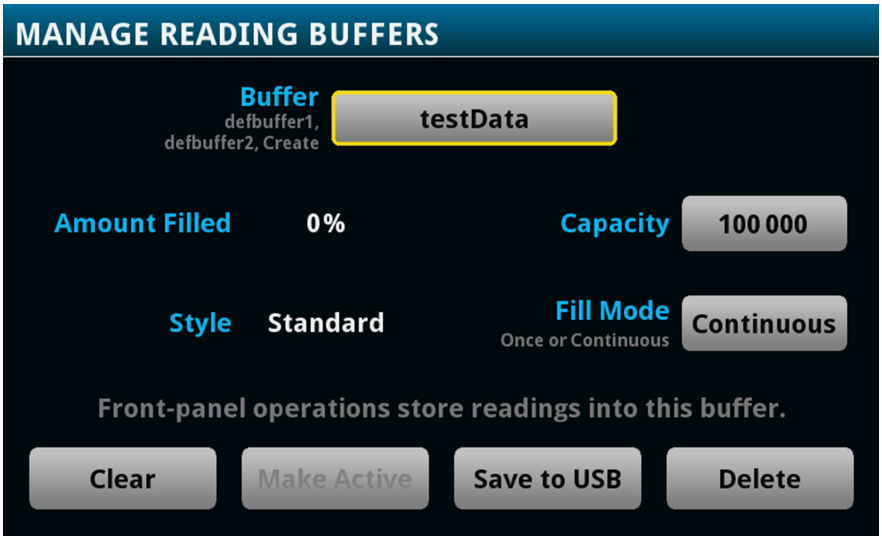
8. The Initial Capacity window is displayed. Enter the number of readings that the buffer will hold. To set the capacity to the maximum buffer size, based on the available memory of the instrument, enter **0**.

Figure 103: Initial Capacity window



9. Select **OK**. The MANAGE READING BUFFERS window is displayed, showing the buffer you just created.

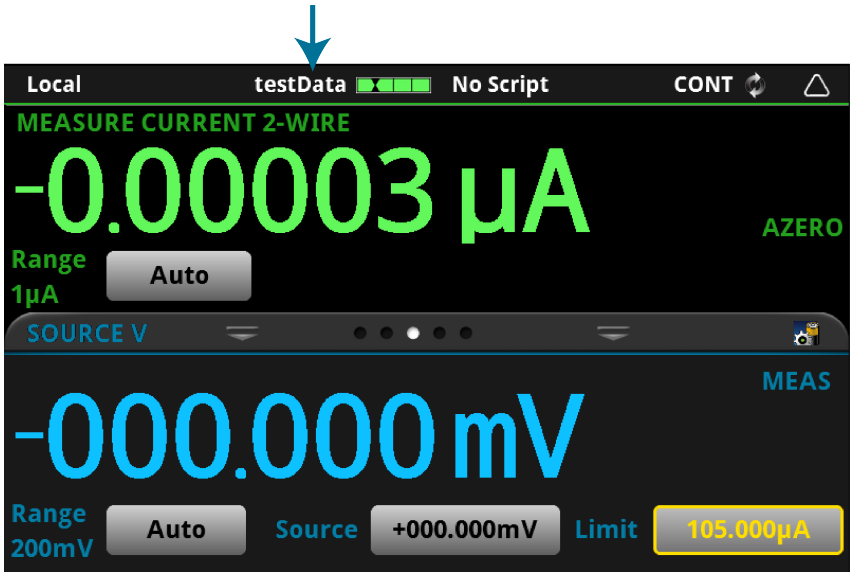
Figure 104: MANAGE READING BUFFERS window



10. Press the **HOME** key to return to the home screen.

After you create a new reading buffer, the new reading buffer becomes the active buffer.

Figure 105: Active buffer indicator



NOTE

You can select the reading buffer indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar. The green bar next to the buffer name indicates how full the buffer is. To create a new buffer, select Create New from the reading buffer indicator.

To use SCPI commands to create a reading buffer:

To create a full reading buffer named `testData` with a capacity of 200 readings, send the following command:

```
TRACe:MAKE "testData", 200, FULL
```

To use TSP commands to create a reading buffer:

To create a full reading buffer named `testData` with a capacity of 200 readings, send the following command:

```
testData = buffer.make(200, buffer.STYLE_FULL)
```

Setting reading buffer options

You can specify the settings for the reading buffers. The settings you can select include:

- **Buffer capacity:** The amount of data the buffer holds
- **Fill mode:** How the incoming data is managed as the buffer fills

Setting reading buffer capacity

The capacity of a reading buffer determines how many readings that buffer holds. You can change the capacity of reading buffers.

NOTE

Readings and statistics that are stored in the reading buffer are deleted when you change the capacity of a buffer.

For user-defined buffers, you assign a capacity when you create the reading buffer. For default buffers (`defbuffer1` and `defbuffer2`), the initial buffer size is 100,000 readings. If you specify 0 for the size of the buffer, the instrument assigns the largest buffer size possible based on the available memory.

The buffer style you choose when you create the reading buffer affects the capacity of the reading buffer. For example, the compact buffer style stores more readings with lower resolution and less reading information than a standard or full buffer. For more information about buffer styles and their capacities, see Creating buffers (on page 6-5).

The available capacity for reading buffers is affected by other operations of the instrument. Factors that can affect buffer capacity include:

- Other reading buffers. The total capacity of the reading buffers in the instrument affects the space that can be allocated to a new buffer.
- Scripts that are loaded onto the instrument.
- Applications that are loaded onto the instrument.
- Configuration lists.
- Scripts that are actively running.
- Variables that reside in the run-time environment.
- The TriggerFlow® trigger model.
- USB drives that contain files that use a lot of memory, such as image files.
- TSP-Link nodes.

CAUTION

The 2470 notifies you when the system runs out of memory. If the instrument encounters memory allocation errors (errors that specifically state "Out of Memory"), the state of the instrument cannot be guaranteed. After attempting to save any important data, turn off power to the instrument and turn it back on to reset the runtime environment and return the instrument to a known state. Unsaved scripts and reading buffers will be lost.

To maximize the memory available for reading buffers:

- Delete user-defined buffers that you are not using. If you power cycle the instrument, all user-defined buffers in the instrument are deleted.
- If you are not using the default buffers, set them to the minimum capacity of 10.
- Remove the USB flash drive from the USB port.
- Reduce the number of scripts. Refer to Working with scripts (on page 13-5) for information on managing scripts.
- Reduce the number of configuration lists or the number of indices stored in configuration lists. Refer to Configuration lists (on page 4-82) for information on managing configuration lists.
- Reduce the number of TSP-Link nodes.
- Delete unneeded global variables from the run-time environment by setting them to `nil`.
- Adjust the `collectgarbage()` settings in Lua. See Lua memory management (on page 13-28) for more information.
- Review scripts to improve their memory usage. In particular, you can see memory gains by changing string concatenation lines into a Lua table of string entries. You can then use the `table.concat()` function to create the final string concatenation.

For additional information on memory use by the instrument, refer to Memory considerations for the run-time environment (on page 13-41).

The overall capacity of all buffers stored in the instrument can be up to 4,500,000 readings for standard reading buffers and 20,000,000 for compact reading buffers.

The maximum capacity for a reading buffer that is set to the Full style is less than one set to the Standard style. Full style buffers store more data for each reading.

The following topics describe how to set the reading buffer capacity.

Using the front panel to set buffer capacity:

NOTE

When you resize a reading buffer, data in the buffer is cleared.

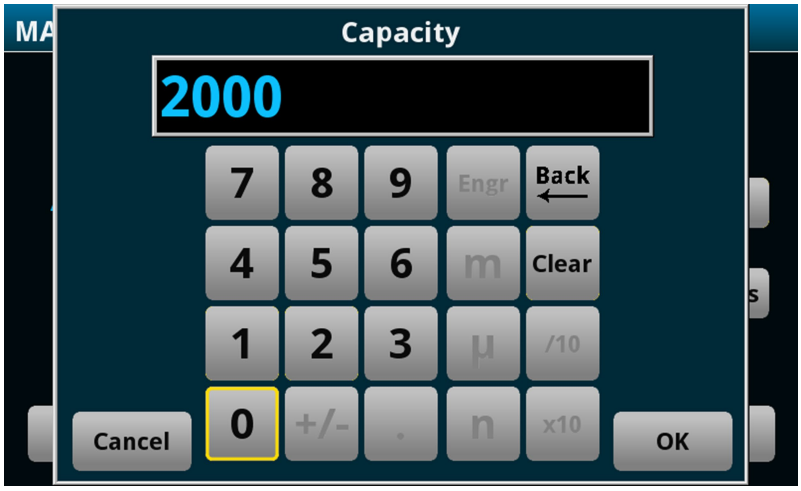
1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 106: MANAGE READING BUFFERS window



3. Select a reading buffer from the list. For example, select `testData`. The settings for `testData` are displayed.
4. Select **Capacity** and enter the new size for the buffer.

Figure 107: Buffer Capacity number pad



5. Select **OK**. The MANAGE READING BUFFERS window is displayed.
6. Press the **HOME** key to return to the home screen.

Using SCPI commands to set buffer capacity:

To set the `testData` reading buffer to hold 300 readings, send the following command:

```
TRACe:POINts 300, "testData"
```

Using TSP commands to set buffer capacity:

To set the `testData` reading buffer to hold 300 readings, send the following command:

```
testData.capacity = 300
```

Setting the fill mode

The fill mode setting for the reading buffer controls how the incoming data is managed as the buffer fills. You can set the read buffer to:

- **Fill once:** The buffer stops accepting data once it fills to capacity. When the buffer reaches capacity, no more readings are made and event code 4915, "Attempting to store past capacity of reading buffer," is displayed.
- **Fill continuously:** Data fills the buffer normally until the end of the buffer is reached. When the end is reached, the data returns to the beginning of the buffer and overwrites the oldest reading. This is a traditional circular buffer. In this case, the buffer never technically fills.

NOTE

When readings are made using a high sample rate and stored into a continuous reading buffer with a capacity of less than 10,000 readings, the instrument may not be able to fully process the incoming data before it is overwritten with new data. This can result in gaps in graph traces and the loss of statistics and histogram information. To prevent these problems, increase the buffer capacity or reduce the sample rate. If you are measuring with a measure count of more than 1 into a continuous buffer, size your buffer two or more times greater than the measure count to allow graphing and other system operations to work more efficiently.

The following topics describe how to set the reading buffer fill mode.

Using the front panel to set fill mode:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 108: MANAGE READING BUFFERS window



3. Select a reading buffer from the list. For example, select `testData`. The settings for `testData` are displayed.
4. Select the **Fill Mode** option.
5. Press the **HOME** key to return to the home screen.

Using SCPI commands to set the buffer fill mode:

To set the `testData` reading buffer fill mode to continuous, send the following command:

```
TRACe:FILL:MODE CONT, "testData"
```

To set the `defbuffer1` reading buffer fill mode to fill once, send the following command:

```
TRACe:FILL:MODE ONCE, "defbuffer1"
```

To get the fill mode that is set, send the following command:

```
TRACe:FILL:MODE? "defbuffer1"
```

Where a return of `ONCE` indicates the buffer is set to fill once and a return of `CONT` indicates the buffer is set to fill continuously.

Using TSP commands to set a buffer fill mode:

To set the `testData` reading buffer fill mode to continuous, send the following command:

```
testData.fillmode = buffer.FILL_CONTINUOUS
```

To set the `defbuffer1` reading buffer fill mode to fill once, send the following command:

```
defbuffer1.fillmode = buffer.FILL_ONCE
```

To print the `defbuffer1` fill mode setting, send the following command:

```
print(defbuffer1.fillmode)
```

Where a return of `0` indicates the buffer is set to fill once and a return of `1` indicates the buffer is set to fill continuously.

Selecting a buffer

The default reading buffer is `defbuffer1`. You can also use a different buffer (`defbuffer2` or a user-defined reading buffer).

When you use remote commands to create buffers, the buffers are available to the system and can be used with any command that takes a buffer parameter. A newly created buffer automatically becomes the active buffer. If the active buffer is deleted, `defbuffer1` becomes the active buffer.

Using the front panel:

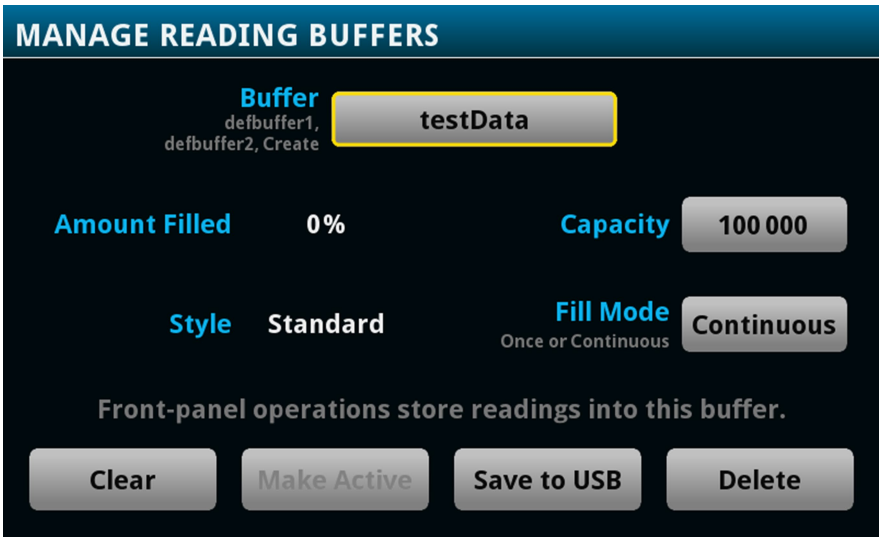
1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 109: MANAGE READING BUFFERS window



3. Select a reading buffer from the list. In the example below, `testData` is selected.

Figure 110: MANAGE READING BUFFERS window



4. Select the **Make Active** button. The "Are you sure" dialog box is displayed.
5. Select **Yes**.

You can also select reading buffers from the active buffer indicator on the home screen. Refer to Active buffer indicator (on page 3-11) for information about using the indicator to select buffers.

Using SCPI commands to select a reading buffer:

To make a measurement and store the readings in a specific reading buffer, send the command:

```
:READ? "<bufferName>"
```

If you do not specify a buffer name, readings are stored in `defbuffer1`.

An alternative to sending the `:READ? "<bufferName>"` command is to send the command:

```
:TRACe:TRIGger "<bufferName>"
```

The `:TRACe:TRIGger` command stores readings in the specified reading buffer. If no buffer is specified for the parameter, `defbuffer1` is used. To see the readings stored in the buffer after using this command, use the `:FETCh?` command to see the last reading stored in the buffer or the `:TRACe:DATA?` command to see multiple readings from the buffer.

NOTE

To specify a user-defined reading buffer, you must create the buffer first.

To select current as the measurement function, measure current, and return the readings in the `testData` reading buffer, send the following commands:

```
:SENSe:FUNCtion "CURRent"
:READ? "testData"
```

To measure current and store the readings in the `defbuffer2` reading buffer, send the following command:

```
:MEASure:CURRent? "defbuffer2"
```

To measure voltage and store the readings in the `defbuffer2` reading buffer, send the following command:

```
:MEASure:VOLTage? "defbuffer2"
```

To measure current and return the relative time and a reading, send the following command:

```
:MEASure:CURRent? "testData", REL, READ
```

Buffer storage is consistent whenever readings are taken. Parameters such as `REL` and `READ` only affect what is included in the response. If you do not include parameters, the command only returns the reading.

Using TSP commands to select a reading buffer:

To make a measurement and store the readings in a specific reading buffer, use the `smu.measure.read(`*`bufferName`*`)` function. If you do not specify a buffer when you use the `smu.measure.read()` function, readings are stored in `defbuffer1`.

To measure voltage and store the readings in the `voltMeasBuffer`, send the following commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.read(voltMeasBuffer)
```

To measure voltage, store the readings in the `voltMeasBuffer`, and print the last reading in the buffer, send the following command:

```
print(smu.measure.read(voltMeasBuffer))
```

To measure current, store the readings in `defbuffer1`, and print the last reading in the buffer, send the following commands:

```
smu.measure.func = smu.FUNC_DC_CURRENT
print(smu.measure.read())
```

Viewing and saving buffer content

You can view the content of buffers from the front panel.

You can also save the contents of the reading buffer to a USB flash drive. The stored file can be loaded directly into Microsoft® Excel® or another tool. The file contains all the information the instrument records about each data point in the reading buffer. When you save the buffer data, you may indicate a starting or ending point to save only a portion of the data. If you do not specify a starting and ending point, the entire buffer data is saved. You may also specify how you want the time saved with the time format parameter.

Using remote commands, you can append the contents of a reading buffer to a file that is already on the USB flash drive. When you append data, you can specify the starting and ending point in the buffer to save only a portion of the data.

All readings are saved in the comma-separated value (`.csv`) file format. This format stores tabular data (numbers and text) in plain-text form. You can import the CSV file into a spreadsheet.

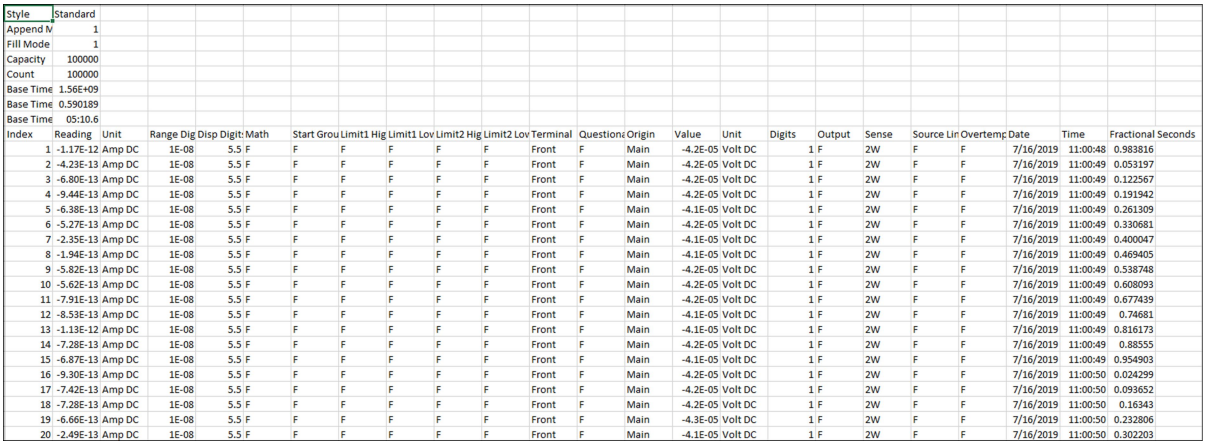
The 2470 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing CSV files on the flash drive.

NOTE

The header rows in the CSV file are not fixed. They may change as buffers are enhanced or when additional information is needed.

See the following figure for an example of a buffer CSV file imported into a spreadsheet.

Figure 111: Example of spreadsheet with reading buffer content



Style	Standard																								
Append M	1																								
Fill Mode	1																								
Capacity	100000																								
Count	100000																								
Base Time	1.56E+09																								
Base Time	0.590189																								
Base Time	05:10.6																								
Index	Reading	Unit	Range Dig	Disp Digit	Math	Start Grou	Limit1 Hig	Limit1 Lov	Limit2 Hig	Limit2 Lov	Terminal	Questiona	Origin	Value	Unit	Digits	Output	Sense	Source Lir	Overtemp	Date	Time	Fractional Seconds		
1	-1.17E-12	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:48	0.983816		
2	-4.23E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.053197		
3	-6.80E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.122567		
4	-9.44E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.191942		
5	-6.38E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.261309		
6	-5.27E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.330681		
7	-2.35E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.400047		
8	-1.94E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.469405		
9	-5.82E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.538748		
10	-5.62E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.608093		
11	-7.91E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.677439		
12	-8.53E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.74681		
13	-1.13E-12	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.816173		
14	-7.28E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.88555		
15	-6.87E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:49	0.954903		
16	-9.30E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:50	0.024299		
17	-7.42E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:50	0.093652		
18	-7.28E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.2E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:50	0.16343		
19	-6.66E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.3E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:50	0.232806		
20	-2.49E-13	Amp DC	1E-08	5.5	F	F	F	F	F	F	Front	F	Main	-4.1E-05	Volt DC	1	F	2W	F	F	7/16/2019	11:00:50	0.302203		

The following table describes the information that is stored in each column of the spreadsheet.

An F in a column indicates the corresponding heading item is false for that reading. For example, if an F is listed in the Math column, it indicates that the item was not used or did not occur on that reading.

A T in a column indicates that the corresponding heading item is true for that reading. For example, if a T is listed in the Math column, it indicates that the item was applied to that reading.

Heading	**Description**
Index	Provides an identifier for each reading
Reading	Measured value for each reading
Unit	Indicates the unit of measure for the reading; values may be any of the following: `Volt`, `Amp`, `Ohm`, or `Watt`
Range Digits	Positive full-scale value of the measurement range that the instrument is presently using; values may be any of the following: `".000000001", ".00000001", ".0000001", ".000001", ".00001", ".0001", ".001", ".01", ".1", "1", "10", "100", "1000", "10000", "100000", "1000000", "10000000", "100000000", "1000000000", "10000000000", "100000000000"`
Disp Digits	The number of digits that are displayed for measurements on the front panel. Values may be `3.5`, `4.5`, `5.5`, or `6.5`
Math	T when Math is ON; F when Math is off
Limit1 High	Specifies that the upper limit for limit 1 has been exceeded
Limit1 Low	Specifies that the lower limit for limit 1 has been exceeded
Limit2 High	Specifies that the upper limit for limit 2 has been exceeded
Limit2 Low	Specifies that the lower limit for limit 2 has been exceeded
Terminal	Specifies which set of input and output terminals the instrument was using when the measurements were made; values may be any of the following: `Front` or `Rear`
Questionable	T or F
Origin	The A/D converter from which the reading originated; for the 2470, this will always be Main.
Value	Value is the source value when the reading was taken; if readback is ON, it is the measured source value; otherwise, it is the programmed value

Heading	Description
Unit	Units of measure of the source; values may be any of the following: `Volt`, `Amp`, `Ohm`, or `Watt`
Digits	Source digits
Output	`On` or `Off`
Sense	`2W` or `4W`
Source Limit	`T` indicates that the source limit was exceeded; `F` indicates that the source limit was not exceeded
Overtemp	`T` indicates that the instrument is in an overtemperature state; `F` indicates that the instrument is not in an overtemperature state
Date	Date the readings were made
Time	Time the readings were made
Fractional Seconds	Fractional portion of the timestamp (in seconds) when each reading was made; fractional seconds are the fractional part of an absolute timestamp, for example, the fractional part of the 11:11:23.45678 timestamp is .45678

You can view data from the reading buffers through the front panel using the Reading Table. The Reading Table displays the following information:

- **Index:** The sequential number of the reading.
- **Time:** The data and time of the reading.
- **Reading:** The data that was measured.
- **Source:** Source value output by the instrument.

If you select a data point, additional detail about that data point is displayed, including the function, math, and limits.

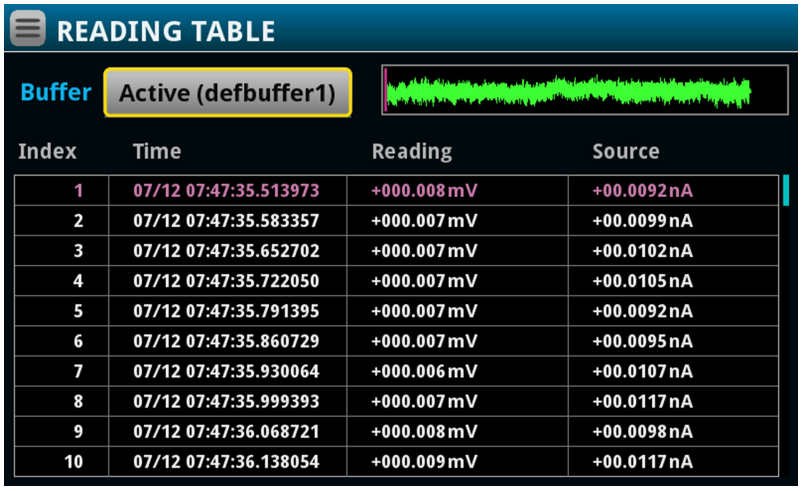
To jump to a specific spot in the data, select the menu in the upper left and select **Jump to Index**. The selected data point is displayed at the top of the reading table.

To save the data, select the menu in the upper left and select **Save to USB**. For information on the options, refer to Options when saving buffer data to a USB flash drive (on page 6-21).

Using the front panel to view the contents of a reading buffer:

1. Press the **MENU** key.
2. Under **Views**, select **Reading Table**. Data for the active reading buffer is displayed.

Figure 112: Reading table



3. To display data for a different reading buffer, select the new buffer.
4. To view details for a specific data point, swipe the table up or down and select the data point to view the Reading Details. If there are many data points, select an area on the reading preview graph in the upper right corner of the screen to get closer to the data you want, and then scroll to the data point. You can also select the menu and select **Jump to Index** to go to a specific point.
5. Press the **HOME** key to return to the home screen.

Using the front panel to store readings in the selected buffer

Before you store readings, make sure the correct reading buffer is selected. See Selecting a buffer (on page 6-14) for more information.

NOTE

Each time a reading buffer is created, the instrument automatically selects the newly created buffer as the active buffer.

To store a reading from the front panel, make a measurement. The buffer-fill indicators light up to indicate that the buffer is filling. Depending on the size of the buffer, the lit indicator may be difficult to observe. When all four indicators are lit, the buffer is completely filled. All the indicators will not be lit if the number of readings stored is less than the selected buffer capacity.

To stop storing readings in a buffer when you are making continuous readings, select the measurement method indicator and select **Manual Trigger Mode**. You can press and hold the **TRIGGER** key for about 3 seconds to display the measurement method options.

NOTE

Stored readings are lost when the instrument is turned off or reset. Stored readings are also lost when you resize a reading buffer.

Options when saving buffer data to a USB flash drive

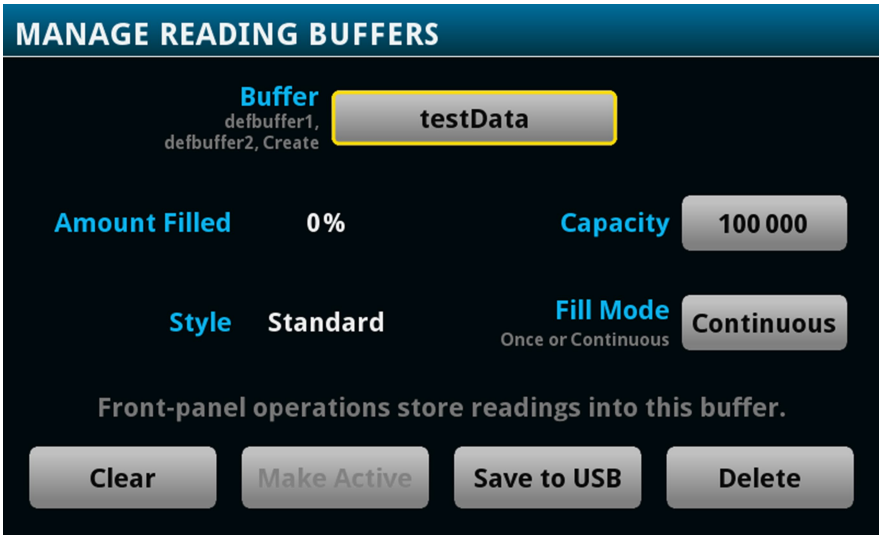
The options available when you save buffer data to a flash drive are described in the following table.

Setting	Description
Extra Value	Available for buffers set to the Full or Writable style. Include extra values in the saved file.
File Layout	Set to Rdg Row: Readings are displayed in rows.
Filename	By default, the file name is the same as the buffer name, followed by the date and time when the file was saved. The date and time is in the format `mmdd_hhmmss`. To save the file with a different name, select **Change**. The date and time is not included if you change the file name.
Reading Format	Include the unit, range digits, and display digit settings in the saved file.
Source Value	Include the source value in the saved reading buffer.
Status	Include math, limits, and terminal settings in the saved file.
Time Format	Sets the time format: ■ **Absolute:** Each timestamp provides the time and date that the reading was made or the number of seconds from the first buffer reading that the reading was made. ■ **None:** No timestamp. ■ **Parts:** Timestamps contain dates, hours, minutes, seconds, and fractions of seconds according to Coordinated Universal Time (UTC). ■ **Raw:** Timestamps display the absolute time in seconds. ■ **Relative:** Timestamps are oriented to a timer with the first buffer reading timestamped at 0.000000 seconds. Each following timestamp is then based on the presently selected format.

Using the front panel to save buffer content to files:

1. Insert a USB flash drive into the USB port.
2. Press the **MENU** key.
3. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.
4. Select the reading buffer that you want to save. For example, select `testData`.

Figure 113: MANAGE READING BUFFERS window



5. Select **Save To USB**. The File Content dialog box is displayed. For information on the options, see Options when saving buffer data to a USB flash drive (on page 6-21).
6. To change the file name, select **Change**. A keyboard is displayed.
7. Enter the name of the file in which to save the readings.

NOTE

You only need to enter the name of the file you want to save. It is not necessary to enter the file extension. All files are saved as CSV files.

8. Select **OK** on the keyboard.
9. Select **OK** to save the file. When the MANAGE READING BUFFERS window is displayed again, the file is saved.
10. Press the **HOME** key to return to the home screen.

NOTE

You can also save buffer data from the READING TABLE window. Select the menu in the upper left of the READING TABLE window and select **Save to USB**.

Using SCPI commands to save or append buffer content to files:

Before using any of these commands, insert a USB flash drive into the USB port.

To save readings and formatted timestamps from the default buffer to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE "/usb1/myData.csv", "defbuffer1"
```

To save readings and formatted timestamps from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE "/usb1/myData.csv", "testData"
```

To append readings and formatted timestamps from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE:APPend "/usb1/myData.csv", "testData"
```

To append readings and formatted timestamps from a reading buffer named `testData` from index 6 to index 10 in file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE:APPend "/usb1/myData.csv", "testData", FORM, 6, 10
```

Using TSP commands to save or append buffer content to files:

Before using any of these commands, insert a USB flash drive into the USB port.

To save readings from the default buffer to a file named `myData.csv` on a USB flash drive, send the following command:

```
buffer.save(defbuffer1, "/usb1/myData.csv")
```

To save readings from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
buffer.save(testData, "/usb1/myData.csv")
```

To append readings from a reading buffer named *`testData`* with default time information to a file named `myData.csv` on the USB flash drive, send the following command:

```
buffer.saveappend(testData, "/usb1/myData.csv")
```

Clearing buffers

You can clear all readings and statistics from buffers.

If you reset or power down the instrument, data is cleared from all buffers and user-defined reading buffers are removed.

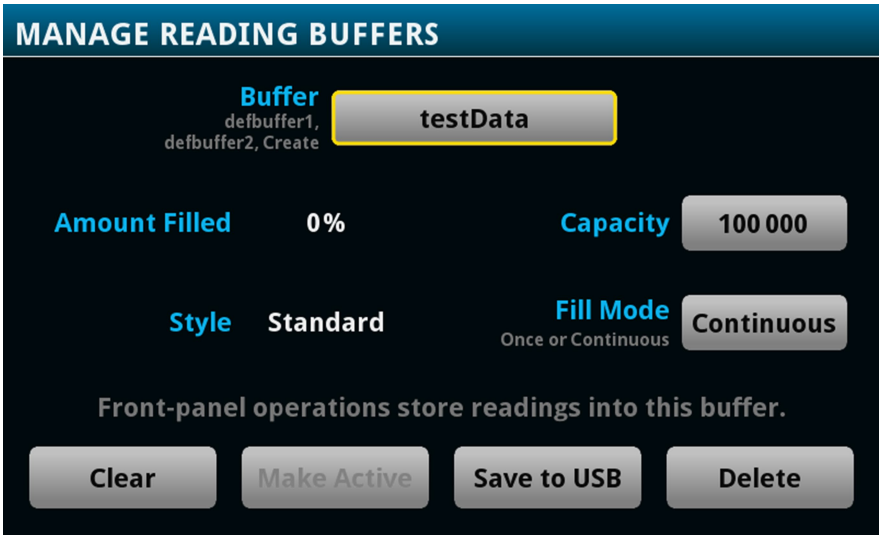
The following topics provide additional information about using the front panel and remote commands to clear buffers.

In addition to the following steps, to clear the active buffer, you can press the **MENU** key and **EXIT** key simultaneously. You can also go to the Statistics swipe screen and select **Clear Active Buffer**.

Using the front-panel Reading Buffers menu to clear a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.
3. Select a reading buffer from the list. For example, select `testData`.

Figure 114: MANAGE READING BUFFERS window



4. Select **Clear** to clear the buffer.
5. A confirmation message is displayed. Select **Yes**.
6. Press the **HOME** key to return to the home screen.

Using SCPI commands to clear a buffer:

To clear a user-defined buffer named `testData`, send the following command:

```
TRACe:CLEar "testData"
```

Using TSP commands to clear a buffer:

To clear a user-defined buffer named `testData`, send the following command:

```
testData.clear()
```

Deleting buffers

If you want to save the readings in a buffer before deleting the buffer, save the buffer to a USB flash drive. See Viewing and saving buffer content (on page 6-17) for details.

You cannot delete the default buffers `defbuffer1` or `defbuffer2`. However, the data in the default buffers is lost when the instrument is reset or the power is turned off.

Using the front panel to delete a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.
3. Select **Buffer**.
4. Select the buffer to be deleted.
5. Select **Delete** to delete the buffer.
6. When the "Are you sure you want to delete testData" prompt is displayed, select **Yes**.

Using SCPI commands:

To delete a user-defined buffer named `testData`, send the following command:

```
:TRACe:DELete "testData"
```

Using TSP commands:

To delete a user-defined buffer named `testData`, send the following command:

```
buffer.delete(testData)
```

NOTE

Do not set the buffer name to `nil` to delete it. To cleanly delete the buffer from the instrument, use the `buffer.delete()` command.

Remote buffer operation

You can control the 2470 buffers through a remote interface using SCPI or TSP remote commands.

This section provides a summary of some of the remote commands available to control and access data stored in buffers; however, this section does not describe all of the available commands. See the following sections for command descriptions:

- For information about SCPI commands, see the SCPI command reference (on page 12-1)
- For information about TSP commands, see the TSP command reference (on page 14-1)

Storing data in buffers

Using SCPI commands:

The table below lists the SCPI commands that you use for data storage.

Command	Description
`:TRACe:ACTual?`	This command contains the number of readings in the specified buffer. See :TRACe:ACTual? (on page 12-117) for more information.
`:TRACe:ACTual:END?`	This command returns the last index in a reading buffer. See :TRACe:ACTual:END? (on page 12-118) for more information.
`:TRACe:ACTual:STARt?`	This command returns the starting index in a reading buffer. See :TRACe:ACTual:STARt? (on page 12-119) for more information.
`:TRACe:CLEar`	This command clears all readings and statistics from the specified buffer. See Clearing buffers (on page 6-23) for more information. Also see :TRACe:CLEar (on page 12-120).
`:TRACe:DELete`	This command deletes a buffer. See :TRACe:DELete (on page 12-124) for more information.
`:TRACe:FILL:MODE`	This command determines if a reading buffer is filled continuously or is filled once and stops. See :TRACe:FILL:MODE (on page 12-124) for more information.
`:TRACe:LOG:STATe`	This command indicates whether the reading buffer should log informational events. See :TRACe:LOG:STATe (on page 12-125) for more information.
`:TRACe:MAKE`	This command creates a user-defined reading buffer. You cannot use this command on the default buffers. See Creating buffers (on page 6-5) for more information. Also see :TRACe:MAKE (on page 12-126).
`:TRACe:POINts`	This command reads the number of readings a buffer can store. This allows you to change the number of readings the buffer can store. See :TRACe:POINts (on page 12-130) for more information.
`:TRACe:SAVE`	This command saves data from the specified reading buffer to a USB flash drive. See :TRACe:SAVE (on page 12-131) for information.
`:TRACE:SAVE:APPend`	This command appends data from the reading buffer to a file on the USB flash drive. See :TRACE:SAVE:APPend (on page 12-133) for more information.
`:TRACe:STATistics:CLEar`	This command clears the statistical information associated with the specified buffer. This command does not clear the readings.
`:TRACe:WRITe:FORMat`	For use with writable buffers only; this function sets the units and number of digits that are written into the reading buffer. See :TRACe:WRITe:FORMat (on page 12-141) for more information.
`:TRACe:WRITe:READing`	For use with writable buffers only; this function writes the data you specify into a reading buffer. See :TRACe:WRITe:READing (on page 12-143) for more information.

Using TSP commands:

CAUTION

Once you create a reading buffer using TSP commands, if you use that buffer name for another buffer or variable, you can no longer access the original buffer.

The table below lists the TSP commands that you use for data storage.

Command	Description
`buffer.clearstats()`	This function clears all statistics from the specified buffer. This function does not clear the readings. See buffer.clearstats() (on page 14-9) for more information.
`buffer.delete()`	This function deletes a user-defined reading buffer. See buffer.delete() (on page 14-10) for more information.
`buffer.make()`	This function creates a user-defined reading buffer. You cannot use this command on the default buffers. See Creating buffers (on page 6-5) for more information. Also see buffer.make() (on page 14-13).
`buffer.save()`	This function saves data from the specified reading buffer to a USB flash drive. See buffer.save() (on page 14-18) for more information.
`buffer.saveappend()`	This function appends data from the reading buffer to a file on the USB flash drive. See buffer.saveappend() (on page 14-19) for information.
`buffer.write.format()`	For use with writable buffers only; this function sets the units and number of digits that are written into the reading buffer. See buffer.write.format() (on page 14-48) for more information.
`buffer.write.reading()`	For use with writable buffers only; this function writes the data you specify into a reading buffer. See buffer.write.reading() (on page 14-50) for more information.
`bufferVar``.capacity`	This attribute reads the number of readings a buffer can store. This allows you to change the number of readings the buffer can store. See bufferVar.capacity (on page 14-22) for more information.
`bufferVar``.clear()`	This function clears all readings and statistics from the specified buffer. See Clearing buffers (on page 6-23) and bufferVar.clear() (on page 14-23) for more information.
`bufferVar``.fillmode`	This attribute determines if a reading buffer is filled continuously or is filled once and stops. See bufferVar.fillmode (on page 14-30) for more information.
`bufferVar``.logstate`	This attribute indicates whether the reading buffer should log informational events. See bufferVar.logstate (on page 14-33) for more information.
`bufferVar``.n`	This attribute contains the number of readings in the specified reading buffer. See bufferVar.n (on page 14-33) for more information.
`bufferVar``.endindex`	This attribute returns the last index in a reading buffer. See bufferVar.endindex (on page 14-25) for more information.
`bufferVar``.startindex`	This attribute returns the starting index in a reading buffer. See bufferVar.startindex (on page 14-42) for more information.

Accessing the data in buffers

Using SCPI commands:

To access a buffer, include the buffer name in the respective command. For example, the following commands:

- Create a buffer named `testData` to store 100 readings
- Set the instrument to make five readings for all measurement requests
- Make the readings and store them in the buffer
- Return five readings (including the measurement and relative time) from the user-defined buffer named `testData`

```
TRAC:MAKE "testData", 100
SENS:COUN 5
TRAC:TRIG "testData"
TRAC:DATA? 1, 5, "testData", READ, REL
```

Using TSP commands:

A reading buffer is based on a Lua table. When you use TSP commands, the measurements themselves are accessed by ordinary array notation. If `rb` is a reading buffer, the first measurement is accessed as `rb[1]`, the ninth measurement as `rb[9]`, and so on. The additional information in the table is accessed as additional members of the table.

To access a buffer, include the buffer name in the respective command. For example, the following commands:

- Create a buffer named `testData` to store 100 readings
- Set the instrument to make five readings for all measurement requests
- Make the readings and store them in the buffer
- Return five readings (including the measurement and relative time) from the user-defined buffer named `testData`

```
-- Create a buffer named testData to store 100 readings.
testData = buffer.make(100)
-- Set the instrument to make 5 readings and store them in the buffer.
trigger.model.load("SimpleLoop", 5, 0, testData)
-- Make the readings
trigger.model.initiate()
waitcomplete()
-- Read the 5 readings and print them including the measurement
-- and relative time for each reading.
printbuffer(1, 5, testData.readings, testData.relativetimestamps)
```

Buffer read-only attributes

Use buffer read-only attributes to access the information contained in an existing buffer.

Using SCPI commands:

The following commands are available for each reading buffer.

Attribute	Description
`:TRACe:ACTual?`	This command returns the number of readings in the specified buffer. See :TRACe:ACTual? (on page 12-117) for information.
`:TRACe:ACTual:END?`	This command returns the last index in a reading buffer. See :TRACe:ACTual:END? (on page 12-118) for more information.
`:TRACe:ACTual:STARt?`	This command returns the starting index in a reading buffer. See :TRACe:ACTual:STARt? (on page 12-119) for more information.
`:TRACe:DATA?`	This command returns the readings stored in a specified reading buffer. See :TRACe:DATA? (on page 12-121) for more information.
`:TRACe:STATistics:AVERage?`	This command returns average of all readings added to the buffer. See :TRACe:STATistics:AVERage? (on page 12-134) for more information.
`:TRACe:STATistics:MAXimum?`	This command returns the maximum reading value added to the buffer. See :TRACe:STATistics:MAXimum? (on page 12-136) for more information.
`:TRACe:STATistics:MINimum?`	This command returns the minimum reading value added to the buffer. See :TRACe:STATistics:MINimum? (on page 12-136) for more information.
`:TRACe:STATistics:PK2Pk?`	This command returns the peak-to-peak value of all readings added to the buffer. See :TRACe:STATistics:PK2Pk? (on page 12-137) for more information.
`:TRACe:STATistics:STDDev?`	This command returns the standard deviation of all readings added to the buffer. See :TRACe:STATistics:STDDev? (on page 12-138) for more information.

Using TSP commands:

See printbuffer() (on page 14-110) for a list of available attributes.

Reading buffer time and date values

Time and date values are represented as a number of UTC seconds since 12:00 a.m. Jan. 1, 1970. Use the following TSP commands to return values in the following formats:

- Hours and minutes: bufferVar.times (on page 14-44)
- UTC seconds: bufferVar.seconds (on page 14-36)
- Month-day-year format, or to access the timestamp table: bufferVar.dates (on page 14-24)

For example, to return the hours and minutes of the readings in `defbuffer1`, send the command:

```
printbuffer(1, 5, defbuffer1.times)
```

The return is similar to:

```
20:30:16, 20:30:16, 20:30:16, 20:30:16, 20:30:16
```

Reading buffer for . . . do loops

The following TSP examples illustrate the use of `for . . . do` loops when recalling data from a reading buffer called `mybuffer`. The following code may be sent as one command line or as part of a script. Example outputs follow the line of code. Also see the printbuffer() (on page 14-110) command.

This example loop uses the `printbuffer()` command to show the reading, units, and relative timestamps for all readings stored in the reading buffer. The information for each reading (reading, units, and relative timestamps) is shown on a single line with the elements comma-delimited.

```
for x = 1, mybuffer.n do
   printbuffer(x, x, mybuffer, mybuffer.units, mybuffer.relativetimestamps)
end
```

Example comma-delimited output of above code:

```
-1.5794739960384e-09, Amp DC, 0
-1.5190692453926e-11, Amp DC, 0.411046134
-2.9570144943758e-11, Amp DC, 0.819675745
-2.9361919146043e-11, Amp DC, 1.228263492
-3.0666566508408e-11, Amp DC, 1.636753752
-4.0868204653766e-11, Amp DC, 2.034403917
```

The following loop uses the `print` command instead of the `printbuffer` command. This loop shows the same information described in the previous example (reading, units, and relative timestamps for all readings stored in the buffer). However, because the `print()` command is used instead of `printbuffer()`, each line is tab-delimited (rather than comma-delimited) to produce a columnar output, as shown below:

```
for x = 1, mybuffer.n do
   print(mybuffer.readings[x], mybuffer.units[x], mybuffer.relativetimestamps[x])
end
```

Example columnar-delimited output of above code:

```
-1.5794739960384e-09 Amp DC      0
-1.5190692453926e-11 Amp DC      0.411046134
-2.9570144943758e-11 Amp DC      0.819675745
-2.9361919146043e-11 Amp DC      1.228263492
-3.0666566508408e-11 Amp DC      1.636753752
-4.0868204653766e-11 Amp DC      2.034403917
```

Writable reading buffers

Writable reading buffers allow you to add external data manually to a user-defined buffer on the 2470.

You can create a writable buffer by specifying the writable or full writable style when you create the buffer over a remote interface using SCPI or TSP commands. You cannot create a writable buffer from the 2470 front panel.

NOTE

Be aware that when you create a writable buffer, it immediately becomes the active buffer. If you try to save readings from the instrument to the writable buffer, errors occur.

If you switch to front-panel control to make readings after selecting or creating a writable buffer, be sure that you select a buffer that is not of the writable style to be the active buffer before you try to store readings. Writable buffers are for manual entry of user-supplied data only and do not store readings measured by the instrument.

To populate a writable reading buffer, you set the format of the units and the unit values for each buffer index using the following commands:

- SCPI: :TRACe:WRITe:FORMat (on page 12-141) and :TRACe:WRITe:READing (on page 12-143)
- TSP: buffer.write.format() (on page 14-48) and buffer.write.reading() (on page 14-50)

After you have populated a writable buffer, you can view the data on your computer from the 2470 virtual front panel or on the front-panel graph screen.

The following example resets the instrument and creates a writable buffer named `writBuffer`. The units for the data are set to Watts and number of display digits to 3½. The example then loads ten lines of data, two with timestamp data and a status marker (256) that shows the data is the first reading in a group, into the buffer.

Using SCPI commands:

```
*RST
TRAC:MAKE "writBuffer", 10, WRIT
TRACe:WRIT:FORM "writBuffer", WATT, 3
TRACe:WRITe:READing "writBuffer", 1, 0, 0, 256
TRACe:WRITe:READing "writBuffer", 2
TRACe:WRITe:READing "writBuffer", 3
TRACe:WRITe:READing "writBuffer", 4
TRACe:WRITe:READing "writBuffer", 5
TRACe:WRITe:READing "writBuffer", 1, 10, 0, 256
TRACe:WRITe:READing "writBuffer", 2
TRACe:WRITe:READing "writBuffer", 3
TRACe:WRITe:READing "writBuffer", 4
TRACe:WRITe:READing "writBuffer", 5
```

The following example resets the instrument and creates a writable buffer named `writBuffer`. The units for the data are set to Watts and number of display digits to 3½. The example then loads ten lines of data, two with timestamp data and a status marker (`buffer.STAT_START_GROUP`) that shows the data is the first reading in a group, into the buffer.

Using TSP commands:

```
reset()
writBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(writBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(writBuffer, 1, 0, 0, buffer.STAT_START_GROUP)
buffer.write.reading(writBuffer, 2)
buffer.write.reading(writBuffer, 3)
buffer.write.reading(writBuffer, 4)
buffer.write.reading(writBuffer, 5)
buffer.write.reading(writBuffer, 1, 10, 0, buffer.STAT_START_GROUP)
buffer.write.reading(writBuffer, 2)
buffer.write.reading(writBuffer, 3)
buffer.write.reading(writBuffer, 4)
buffer.write.reading(writBuffer, 5)
```

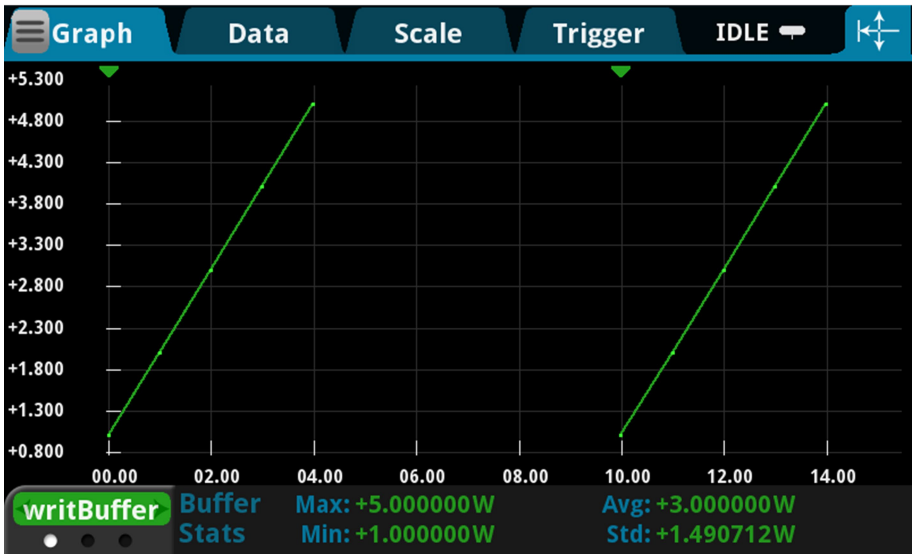
To view the data in the writable buffer on the front-panel graph screen:

1. Press the **MENU** key.
2. Under Views, select **Graph**. By default, time is plotted on the x-axis.
3. Select the **Scale** tab.
4. Set the x-axis method to **Show All Readings**.
5. Select the **Graph** tab.

You can compare the external data to data in another buffer by adding an additional trace to the graph. For more information about graphing data, see Graphing (on page 7-1).

The graph from the example looks similar to the following figure.

Figure 115: Graph with data from a writable buffer with start group markers



Apply mathematical expressions to reading buffer data

You can apply a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

Mathematical expressions for buffer math

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading
- a = previous reading
- t = timestamp of the reading
- c = constant

Expression	**SCPI parameter** **TSP parameter**	**Formula**
No math applied	`NONE` `buffer.EXPR_NONE`	Not applicable
Add	`ADD` `buffer.EXPR_ADD`	$r + a$
Average	`AVERage` `buffer.EXPR_AVERAGE`	$\frac{(r+a)}{2}$
Divide	`DIVide` `buffer.EXPR_DIVIDE`	$\frac{r}{a}$
Exponent	`EXPonent` `buffer.EXPR_EXPONENT`	10^r
Log10	`LOG10` `buffer.EXPR_LOG10`	$\log_{10} r$
Multiply	`MULTiply` `buffer.EXPR_MULTIPLY`	$r * a$
Polynomial	`POLY` `buffer.EXPR_POLY`	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	`POWer` `buffer.EXPR_POWER`	r^c
Rate of change	`RATE` `buffer.EXPR_RATE`	$\frac{(r-r_{-1})}{(t-t_{-1})}$
Reciprocal	`RECiprocal` `buffer.EXPR_RECIPROCAL`	$\frac{1}{r}$
Square root	`SQRoot` `buffer.EXPR_SQROOT`	$\sqrt{r}$
Subtract	`SUBtract` `buffer.EXPR_SUBTRACT`	$r - a$

Set up buffer math using SCPI commands

The SCPI command for setting buffer math is:

:TRACe:MATH (on page 12-128)

Set up buffer math using TSP commands

The TSP command for setting buffer math is:

buffer.math() (on page 14-15)

Using buffers across TSP-Link nodes

After connecting two TSP-Link® enabled instruments, you can access buffers over the TSP-Link network.

For local node access to default and custom buffers, you do not need a TSP-Link node number.

For custom buffers on a remote node, you specify the node number when you create the buffer. After the buffer is created, the buffer name is handled as a local variable, so you do not need the node number to refer to the buffer. You can only use that buffer on the remote node on which it was created.

To use the default buffers on a remote node, you need to use the node number in the command to store readings in the default buffer. You also need the node number to access the default buffer data on a remote node.

The following script illustrates how and when you need to include a node reference to access default and custom buffers when the instrument is part of a TSP-Link network.

```
tsplink.initialize()
reset()
-- Access defbuffer1 on the local node.
smu.measure.read(defbuffer1)
-- Access defbuffer1 on a remote node.
node[9].smu.measure.read(node[9].defbuffer1)

-- Access a custom buffer on the local node.
myBuffer = buffer.make(100)
smu.measure.read(myBuffer)
-- Access a custom buffer on a remote node.
myRemoteBufferOnNode9 = node[9].buffer.make(100)
node[9].smu.measure.read(myRemoteBufferOnNode9)

-- It is illegal to reference the custom buffer on a different node.
-- For example, node[8].smu.measure.read(myRemoteBufferOnNode9) generates an error.
```

Section 7

Graphing

In this section:

Introduction 7-1
About the graph screens 7-1
How to work with the graph 7-3
Use the Graph swipe bar 7-4
Change the data that is graphed 7-6
Add, remove, and clear traces 7-6
Active buffer 7-7
Change the display of data 7-7
Change the scale of the graph 7-8
Set up triggers 7-9
About the Histogram screen 7-12

Introduction

The graphing features of the 2470 allow you to view your measurement data on the front panel of the instrument. The instrument graphs up to twenty traces in an X-Y graph or in a histogram. Each trace represents the data from a reading buffer. You can access each graph individually.

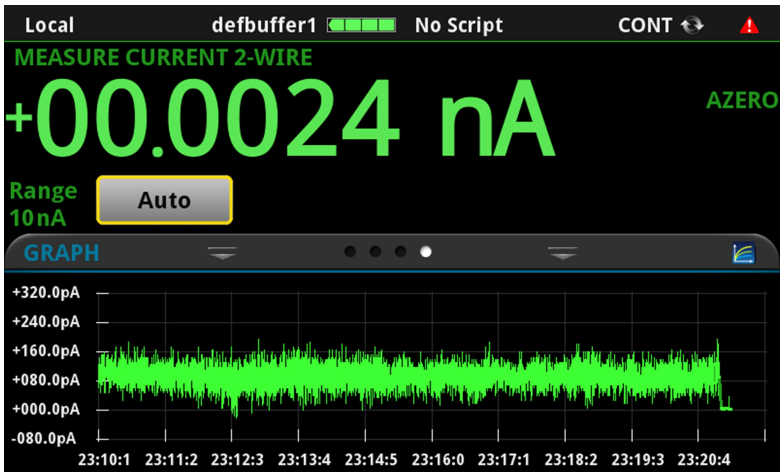
You can manipulate the graph to view minimums and maximums, view averages, determine deltas, and view the values of specific data points. You can also set up triggers and initiate data collection from the graph and histogram screens.

About the graph screens

When you start the instrument, the instrument graphs data from the default reading buffer, defbuffer1. You can change the reading buffer as needed. You can view the graph from either the Graph swipe screen or the Graph screen.

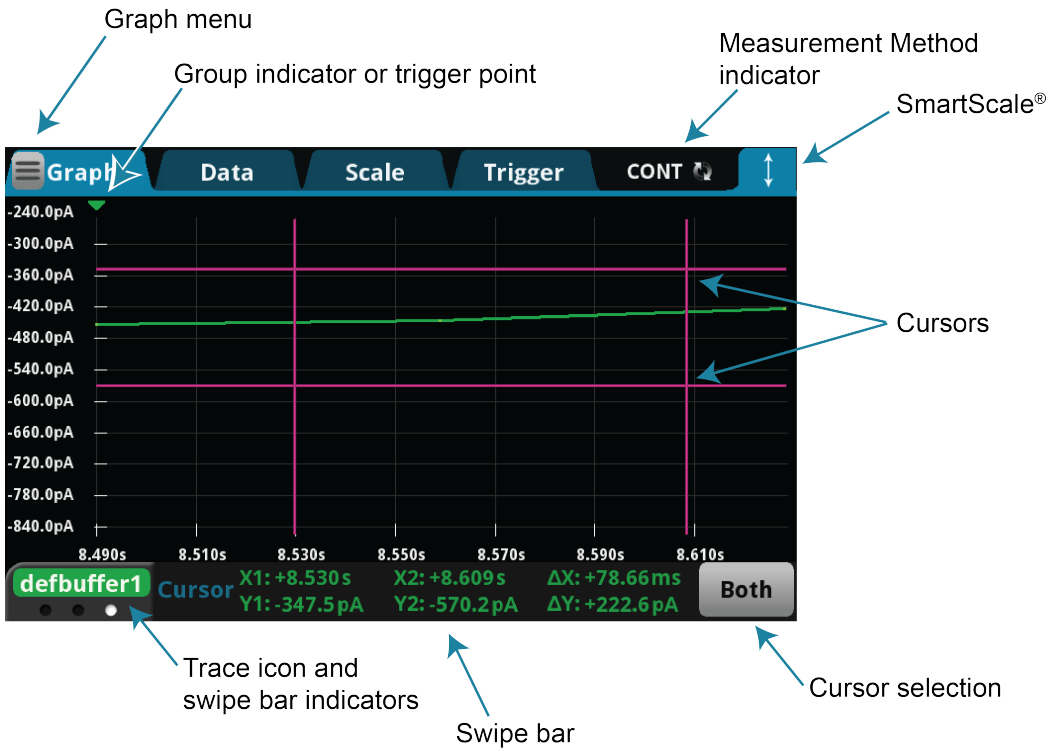
The Graph swipe screen is available from the home screen and displays a smaller version of the full graph. It allows you to see both the data on the home screen and a graph of the information. It does not allow you to zoom in on data or view data at a specific point. To view the graph on the Graph swipe screen, swipe the bottom of the home screen until the graph is displayed. An example of the Graph swipe screen is shown in the following figure.

Figure 116: Home screen with the GRAPH swipe screen displayed



You can open the full graph by selecting the graph icon on the right side of the Graph swipe screen header. You can also open it from the main menu. To open it from the main menu, under Views, select Graph. An example of the full graph is shown in the following figure.

Figure 117: Graph tab



Use the measurement method indicator in the upper right corner of the screen to select the measurement method. Refer to Measurement method indicator (on page 3-12) and Setting up triggers (on page 7-9) for details.

The SmartScale® feature in the upper right allows you to quickly return to automatic scaling. Automatic scaling is turned off if you change the graph by dragging or using pinch to zoom. When the SmartScale feature is on, the instrument keeps the latest data displayed and determines the best way to scale that data based on the data and the instrument configuration (such as the measure count).

The swipe bar at the bottom of the graph displays different types of information about the content of the graph. You can display swipe bars for the scale, buffer statistics, and cursors. The dots below the trace icon show how many swipe bars are available. Refer to Use the Graph swipe bar (on page 7-4) for detail on the swipe bar options.

If the count is set to more than one, the group indicator shows the point at which one count ends and another begins. The group indicator does not necessarily mark the location of a trigger event. System latency and programmed delays may cause the first measurement of a group to be displaced in time from its associated trigger event.

The Graph menu in the upper left corner of the screen allows you to manipulate how the data is displayed and tracked on the Graph tab.

How to work with the graph

The Graph tab displays data in an x-y graph. In most cases, the graph shows the timestamp on the x-axis and the measurements on the y-axis. The Graph tab displays the data as it is added to the selected reading buffer.

The timestamps are displayed on the x-axis. When data is coming in quickly, the first timestamp displays the first few digits in orange. Other timestamps show two orange dots (..) in place of those values.

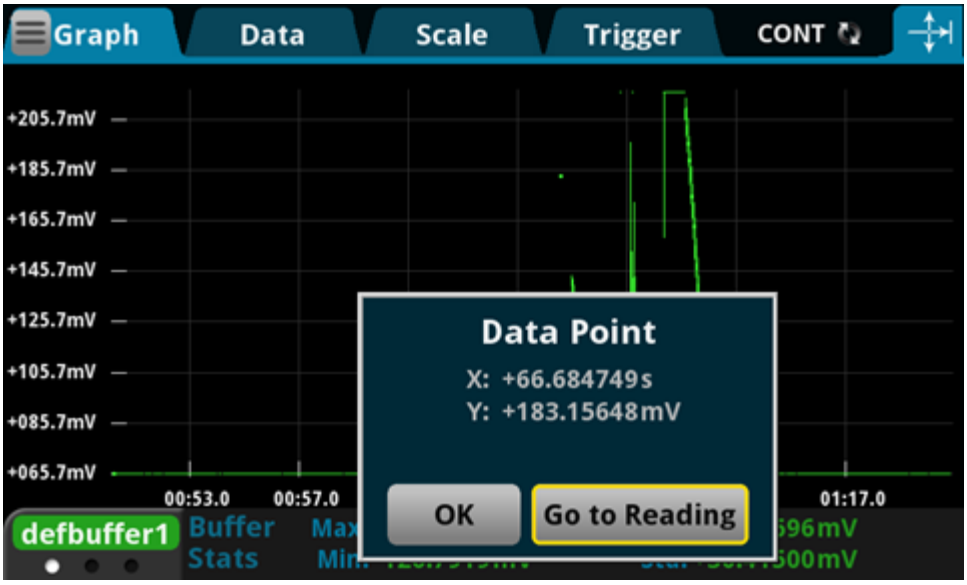
If the y-axis displays a question mark (?), there are multiple units of measure in the reading buffer. Clear the buffer to clear the inconsistent units.

The name of the trace at the bottom left indicates the source of the graph data. If more than one reading is selected as a trace, you can switch between the traces using the trace icon on the lower left. Select the left side of the icon to display the previous trace. Select the right side to display the next trace. Only traces that contain data are displayed.

You can pinch to zoom in the graph. You can also drag the graph to the left or right. When you adjust the view, the SmartScale® feature is turned off. To turn SmartScale on again, select the icon in the upper right of the Graph tab. When SmartScale is on, the instrument determines the best way to scale data based on the data and the instrument configuration (such as the measure count).

You can zoom to display a specific data point. When you select the data point, the Data Point dialog box is displayed with the x and y values of that point. You can also select Go to Reading, which opens the Reading Table screen with that data point selected.

Figure 118: Data point selected on the graph

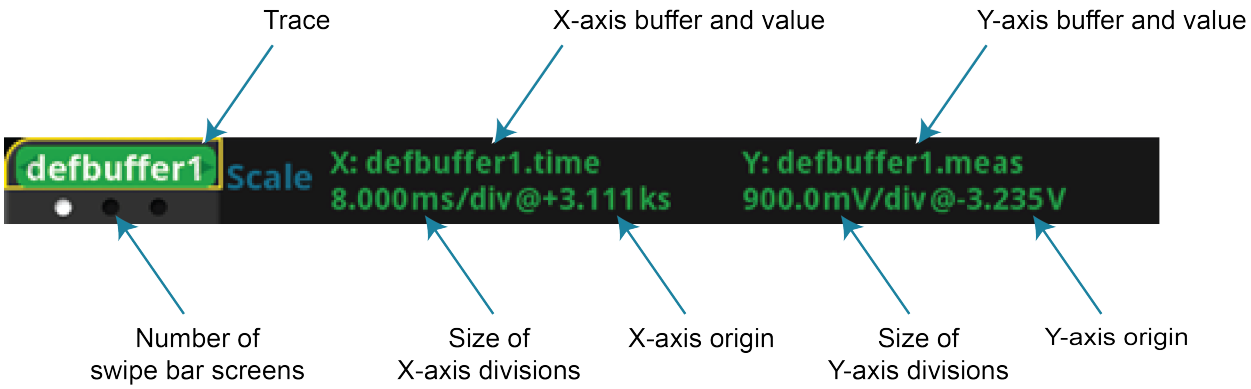


Use the Graph swipe bar

You can swipe the bottom of the Graph tab to display different types of information about the content of the graph. You can display swipe bars for the scale, buffer statistics, and cursors. The dots below the trace icon show how many swipe bars are available.

The Scale swipe bar displays the buffer data that is used for the x and y axes. It also displays the origin value of each axis and the size of the divisions for each axis in the present view.

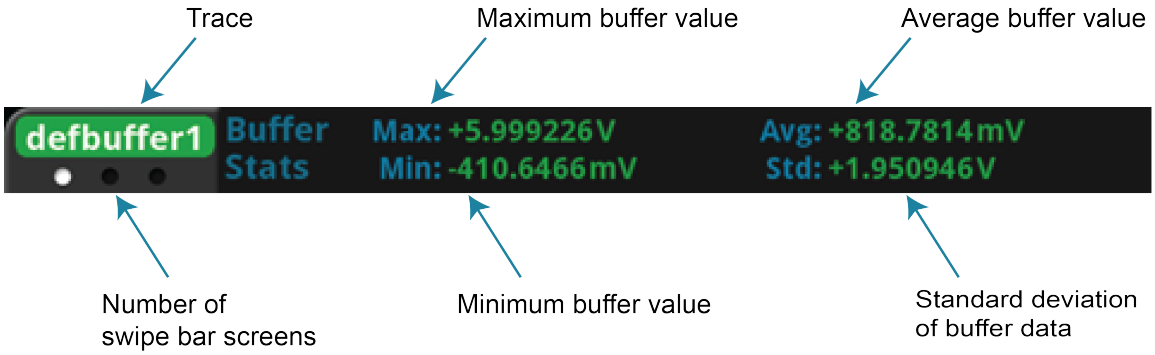
Figure 119: Scale swipe bar



The Buffer Stats swipe bar displays statistics for the readings in the selected trace. If vertical cursors are displayed, the statistics reflect the value within the cursors. You can move the cursors to change the reported statistics. The statistics are:

- **Max:** Maximum value.
- **Min:** Minimum value.
- **Avg:** Average of the values.
- **Std:** Standard deviation for the buffer.
- **Pk2Pk:** Shown instead of standard deviation if vertical cursors are selected. The deviation between the peak-to-peak values.

Figure 120: Buffer Stats swipe bar

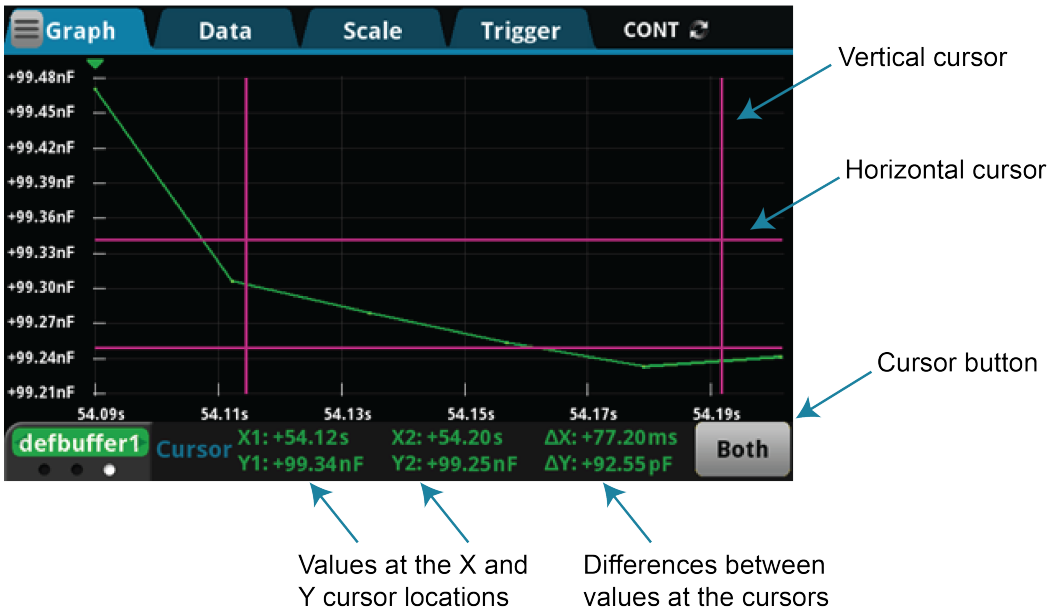


If the buffer type is set to full and contains extra values, the statistics are shown as Not Applicable.

The Cursor swipe bar allows you to display cursors on the graph. If the cursors are displayed, the swipe bar also displays the value at each cursor and the differences between the values at the cursors. You can display vertical, horizontal, or both cursors.

When cursors are displayed, you can drag them to change their positions. You can also move the graph behind the cursors. To move the graph, select a portion of the graph that is not near a cursor and drag. Note that you cannot use the navigation control to change cursor position.

Figure 121: Cursor swipe bar



When cursors are enabled, they are displayed for all traces.

Change the data that is graphed

You can change the data that is displayed on the Data tab of the Graph view. On the Data tab, you can:

- Add, remove, and clear traces
- Change how data is displayed

When you change the traces assigned to the histogram, it also changes the traces assigned to the graph. Conversely, changing the traces for the graph changes the traces assigned to the histogram.

Add, remove, and clear traces

The graph plots data from reading buffers, which are set up on the Data tab as traces. The data from each buffer is shown as a separate trace on the Graph tab. You can select up to 20 traces. The selected traces are shown in the Traces list on the Data tab.

To add a trace, select Add. You are prompted to select source or measure data. Traces that display measurement data are labeled with `.meas` after the buffer name, such as `defbuffer1.meas`. Traces that display source data are labeled with `.src` after the buffer name.

If the reading buffer contains extra data, you are prompted to select which buffer element to graph. Select Measure to plot the measurement values or Extra to plot the extra values. Extra values are available when the buffer type is set to Full or Full Writable.

If you have 20 traces in the list, Add is not available. You must remove a trace before you can add another one.

Colors are automatically assigned to the traces and cannot be changed.

To remove a trace, select the trace and select Remove. The trace is removed from the graph. The data in the buffer is not affected.

To remove data from a reading buffer, select the reading buffer from the Traces list and select Clear Buffer. To clear data from the active buffer, you can press the MENU and EXIT keys simultaneously.

Active buffer

The active buffer contains the data that is displayed on the home screen and where readings are stored when Continuous Measurement is selected or a manual trigger is generated.

When an active buffer is selected on the Data tab, that trace tracks the active buffer instead of a specific buffer. If the active buffer changes, the data that is displayed changes to match the new active buffer.

To remove the active buffer from the list of traces, select it and select Remove Trace. This does not affect the active buffer that is selected on the home screen. To restore the active buffer to the list of traces, select Add and select the trace labeled Active.

Change the display of data

You can set the drawing style and the graph type of the graph on the Data tab of the Graph screens.

The drawing style determines how data is represented when there are many data points. You can select Line, Marker, or Both.

When Line is selected, the data points are connected with solid lines. When Marker is selected, the individual data points are shown with no connecting lines. When Both is selected, the individual data points are shown, and the points are connected with solid lines.

The Graph Type sets the data to be plotted on the x-axis for all traces. You can select Scatter/IV or Time. Scatter/IV is only available if the buffer style is set to Full. All traces must be based on buffers that are set to full in order to select Scatter/IV.

Change the scale of the graph

The Scale tab contains settings that allow you to fine-tune the display of data on the Graph tab. You can select automatic scaling or specific values for the x and y axes.

For both axes, you can select the SmartScale® feature. When the SmartScale feature is selected, the instrument scales the graph automatically, determining the best scaling and tracking method based on the data, reading groups, number of traces, and instrument configuration. The scale is set to show the most relevant portion of the data that is in the selected reading buffer.

For the x-axis, you can also set the Method to one of the following options:

- **Show New Readings:** The graph always displays the latest data on a fixed scale.
- **Show Group of Readings:** The graph displays a group of readings. A group is automatically created when the measure count is set to more than 1.
- **Show All Readings:** All data in the buffer is displayed on the graph.

If you are graphing one trace, you can set the y-axis to one of the following methods:

- **Autoscale Always**: Continuously scales the y-axis of the trace so it fits the height of the screen.
- **Autoscale Once**: Scales the y-axis of the trace once.

If you are graphing more than one trace, you can set the Y-Axis Method to one of the following options:

- **Independent Autoscale**: Scales the y-axis of the trace so it fits the height of the screen.
- **Swim Lanes**: Scales the y-axis of the traces in equal, non-overlapping portions of the height of the screen.
- **Shared Autoscale**: Scales the y-axis so that the minimum and the maximum are shared across all traces. Accommodates the minimum and maximum of all traces.

You can also set the automatic scaling method to off for either axis. When automatic scaling is off, you can manually set the scale and the minimum position. For the y-axis, you can set the scale for each trace. Use the Trace button above the Scale and Minimum Position options to select the trace. Information specific to the selected trace is shown in the same color as the trace.

Scale sets the reading value scale for each division. To select a scale, chose Up or Down until the scale you want to set is displayed, then select the value. The scale changes to show the new value.

Minimum Position sets the first value that is visible on the graph for the selected trace.

The Y-Axis Scale Format determines if the graph is linear or logarithmic. Select Linear to increase the step size in even increments. Select Log to increase the step size exponentially.

Set up triggers

The Source Event option on the Trigger tab lets you define the trigger events and attributes that initiate system measurements.

When you set up triggers through the Trigger tab, the instrument defines the LoopUntilEvent trigger-model template with the trigger settings. The active reading buffer is used as the buffer for the trigger model. If a trigger model exists, it is replaced by the new settings.

To use the configured source event for triggered measurements, you need to change the measurement method from Continuous to Initiate Trigger Model. You can do this while on the Trigger tab. Select the measurement method indicator (to the right of the Trigger tab) and select Initiate Trigger Model.

When you set up the source event, settings are applied as you select the options for the selected source event. The settings are retained and displayed when you return to the Trigger tab.

Types of triggers

You can set triggers to be generated from the:

- **Digital Input Line:** The trigger occurs when a pulse is detected from the digital input line.
- **TSP-Link Input:** The trigger occurs on a falling, rising, or either edge of the TSP-Link input.
- **Display TRIGGER Key:** The trigger occurs when you press the TRIGGER key.
- **Source Limit:** The trigger occurs when the source attempts to exceed the source limit.

Select None to clear the setup for the source event trigger.

NOTE

This section describes in general how to set up triggering. It does not describe details on the trigger sources.

- For detail on the digital I/O, refer to Digital I/O (on page 8-12).
- For detail on TSP-Link, refer to TSP-Link System Expansion Interface (on page 9-1).

Trigger settings

For all triggers, you can set the delay, position, and trigger clear options. Some of the triggers have additional settings. All the settings are described in the following text.

Source Event

The Source Event selects the type of trigger.

When you select the source event, you may be prompted for additional information:

- When you select the Digital Input Line, you are prompted for the input line that generates the trigger (1 to 6).
- When you select TSP-Link Input, you are prompted for the TSP-Link input line that generates the trigger (1 to 3).

To change the line or waveform, select the source event again.

Settings

The Settings icon is available when the source is set to Digital Input Line or TSP-Link Input. Select the icon to set the type of edge that generates a trigger. You can set rising, falling, or either.

Delay

The Delay is the length of time that occurs before the measurement occurs after detecting the selected source event trigger. You can select from 167 ns to 10 ks. Select 0 to set no delay.

Position

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled.

Trigger Clear

Trigger Clear determines if triggers are cleared before the wait period for the trigger begins. The wait period is set in the trigger model as the wait block.

If you set Trigger Clear to Never, the trigger model clears triggers for the wait block when the trigger model is initiated. The instrument begins making measurements as soon as the trigger model reaches the wait block if it detected the event after being initiated and before reaching the wait block. If the trigger was not detected, the trigger model waits to detect the event before making measurements.

If you set Trigger Clear to Enter, any triggers that occurred after the trigger model was initiated and before reaching wait block are cleared. The source event trigger must occur after reaching the wait block before measurements will begin.

All triggers are cleared when the trigger model begins. Trigger Clear only affects triggers that occur after the trigger model is initiated. Triggers are also cleared when the trigger model exits the Wait block.

Graph measurement using triggers

To set up triggers to occur when a trigger occurs:

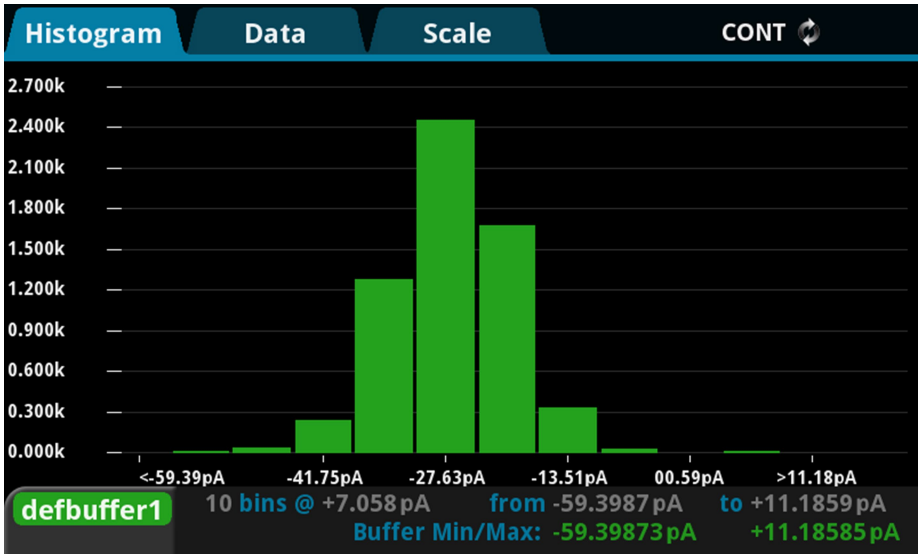
1. Press the **MENU** key.
2. In the View menu, select **Graph**.
3. Select the **Trigger** tab.
4. Set the **Source Event**.
5. Make other settings as needed.
6. Select the **measurement method** indicator at the upper right of the screen and select **Initiate Trigger Model**.
7. To start the measurements, generate the trigger event.
8. Select the **Graph** tab to view the readings.

About the Histogram screen

The Histogram tab graphs readings as a bar graph of the data distribution into bins. To view the histogram, select the **MENU** key, then select **Histogram**.

An example of the histogram screen is shown in the following figure.

Figure 122: Histogram screen



When you start the instrument, the instrument bins data from the default reading buffer defbuffer1. You can change the reading buffer as needed.

The measurement method indicator in the upper right of the screen selects the measurement method. Refer to Measurement method indicator (on page 3-12) and Set up triggers (on page 7-9) for details.

How to work with the Histogram

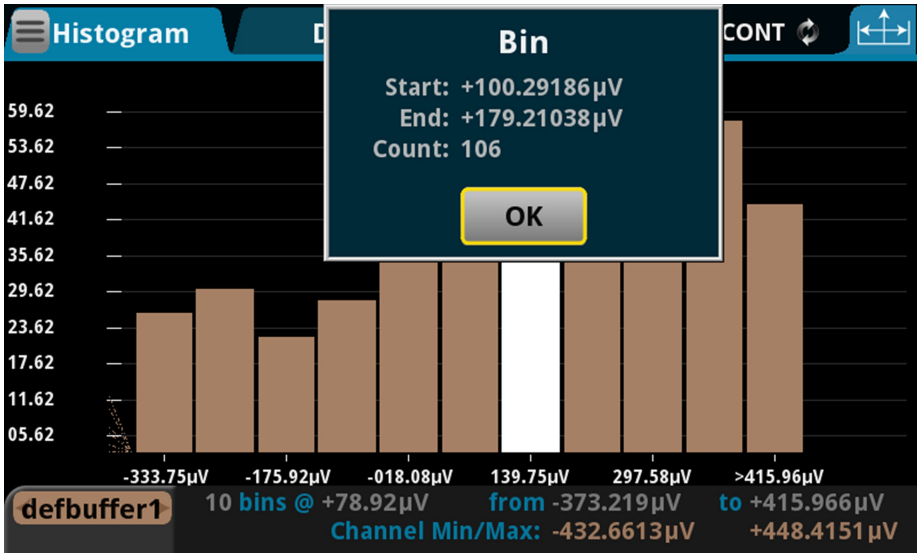
The Histogram tab groups data into bins as it is added to the reading buffer for the selected trace. The range for each bin is shown on the x-axis. The number of readings is shown on the y-axis.

The trace icon at the bottom left indicates the source of the graph data. If more than one reading buffer is selected as a trace, you can switch between the traces using the trace icon. Select the left side of the icon to display the previous trace. Select the right side to display the next trace. Only traces that contain data are displayed.

You can pinch to zoom into or out of data on the histogram. You can also swipe the histogram to the left or right. When you adjust the view, the SmartScale® feature is turned off. To turn SmartScale on again, select the icon in the upper right of the Graph tab. When SmartScale is on, the instrument determines the best way to scale data based on the data and the instrument configuration.

You can select a bin to display the detail for that bin. When you select a bin, the Bin dialog box is displayed with the Start and End values of the bin and the number of measurements within that range.

Figure 123: Histogram Bin dialog box



If the top of a bin has a brighter color rectangle, there is additional data in the bin that is off the screen.

The histogram legend displays information about that data, including the number of bins and the total data range. It also displays the minimum and maximum values of the data in the buffer.

Change the data that is binned

You can change the data that is displayed on the Data tab of the Graph view. On the Data tab, you can add, remove, and clear traces.

NOTE

When you change the traces assigned to the graph, it also changes the traces assigned to the histogram. Conversely, changing the traces for the histogram changes the traces assigned to the graph.

Add, remove, and clear traces

The histogram plots data from reading buffers, which are set up on the Data tab. The data from each buffer is shown as a separate trace on the Histogram tab. You can select up to 20 traces. The selected traces are shown in the Traces list in the Data tab.

To add a trace, select Add and select the buffer. You are prompted to select source or measure data. Traces that display measurement data are labeled with `.meas` after the buffer name, such as `defbuffer1.meas`. Traces that display source data are labeled with `.src` after the buffer name.

If you have 20 traces in the list, Add is not available. You must remove a trace before you can add another one.

Colors are automatically assigned to the traces and cannot be changed.

To remove a trace, select the trace and select Remove. The trace is removed from the graph. The data in the buffer is not affected.

To remove data from a reading buffer, select the reading buffer from the Traces list and select Clear Buffer. To clear the active buffer, you can press the MENU and EXIT keys simultaneously.

Active buffer

The active buffer contains the data that is displayed on the home screen and where readings are stored when Continuous Measurement is selected or a manual trigger is generated.

When an active buffer is selected on the Data tab, that trace tracks the active buffer instead of a specific buffer. If the active buffer changes, the data that is displayed changes to match the new active buffer.

To remove the active buffer from the list of traces, select it and select Remove Trace. This does not affect the active buffer that is selected on the home screen. To restore the active buffer to the list of traces, select Add and select the trace labeled Active.

Change the scale of the histogram

The Scale tab contains settings that allow you to fine-tune the display of data on the Histogram tab. You can select automatic scaling or specific values for the bins and boundaries.

The SmartScale® feature scales the histogram automatically. The instrument determines the best way to bin the data.

The Auto Bin option redistributes the data evenly in the bins based on the present minimum and maximum boundaries.

The Fit option adjusts the y-axis scale so that that tops of all bins are visible.

You can also set the automatic scaling method off. When automatic scaling is off, you manually set the minimum and maximum boundaries. The Maximum Boundary is the highest value of the data that is binned in the histogram. Data that is above this level is binned in the high outlier bin. The Minimum Boundary is the lowest value of the data that is binned in the histogram. Data that is below this level is binned in the low outlier bin.

The Number of Bins determines how many bins the data are sorted into. The histogram creates two outlier bins in addition to the bins you define. These bins are used to collect data that is below or above the defined minimum and maximum boundaries.

Section 8

Triggering

In this section:

Measurement methods 8-1
Triggering 8-3
Digital I/O 8-12
Trigger model 8-25

Measurement methods

Triggers are signals that instruct the instrument to make a measurement. You can set the 2470 to use the following triggering measurement methods:

- **Continuous measurement:** The instrument is making measurements continuously.
- **Manual trigger mode:** Press the front-panel TRIGGER key to initiate a single measurement.
- **Trigger model:** The instrument makes measurements according to the settings of the trigger model. To select this method, a trigger model must be set up. Select Initiate Trigger Model to start the trigger model or Abort Trigger Model to stop a trigger model that is presently running.

Continuous measurement triggering

When you select the continuous measurement method, the instrument makes measurements continuously.

The continuous measurement method is only available when you are controlling the instrument locally (through the front panel).

The instrument stores the readings in the active reading buffer. See Reading buffers (on page 6-1) for detail on the buffer options that are available.

If you press the front-panel TRIGGER key when the instrument is set to the continuous measurement method, measurements are not made. Instead, a dialog box is displayed that asks if you want to change the measurement method.

Trigger key triggering

When you select the Manual Trigger Mode from the 2470 front-panel, the instrument only makes a measurement when you press the front-panel TRIGGER key.

The instrument stores the readings in the active reading buffer. See Reading buffers (on page 6-1) for detail on the buffer options that are available.

Trigger model triggering

When you select the trigger model measurement method, the instrument uses a trigger model to control the sequence in which measurements occur. The 2470 trigger model is flexible, allowing you to control as much or as little as needed for your measurement application.

When you are remotely controlling the instrument, the trigger model measure method is automatically selected. In addition, you can view different buffers from the front panel, but the actual buffer that is used is defined by the remote commands.

For detail on the trigger model, see Trigger model (on page 8-25).

Switching between measurement methods

The measurement methods that are available to you depend on how you are controlling the instrument.

If you are using the front panel to control the instrument, you can choose any of the measurement methods.

If you are using a remote interface to control the instrument, you can only use the trigger model measurement method. When you switch to a remote interface, the trigger model measurement method is automatically selected. If you switch from remote control to front-panel control, the trigger model measurement method remains selected.

If you are running a script, the instrument automatically switches to the trigger model measurement method.

Using the front panel:

1. Press the front-panel **TRIGGER** key for 2 seconds. A dialog box displays the available trigger methods. The presently selected method is highlighted.
2. Select the method you want to use.
3. If the instrument is in remote control, the instrument displays a confirmation dialog box. Select **Yes** to change to local control.

Triggering

Triggering allows you to start and synchronize source and measure actions on one or more instruments with a trigger event or a combination of trigger events that you set. This section describes some of the options available for triggering, including command interface triggering, timers, and event blenders.

Command interface triggering

A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A USBTMC trigger message is received (USB only)
- A `*TRG` message is received

To use a command interface trigger event as an input stimulus for another trigger object, set the stimulus as TSP event `trigger.EVENT_COMMAND` or the SCPI event `COMMAND`. To ensure that trigger commands that are issued over the command interface are processed in the correct order, the instrument does not generate a trigger event until:

- The trigger command is executed
- TSP only: `trigger.wait()` retrieves the trigger command from the command queue before it would normally be executed

Command interface triggering does not generate action overruns. The triggers are processed in the order that they are received in the 2470 command queue. The 2470 only processes incoming commands when no commands are running. Unprocessed input triggers can cause an overflow in the command queue. It is important to make sure a script processes triggers while it is running.

NOTE

If you are using a test script using TSP, the command queue can fill up with trigger entries if over 50 `*TRG` messages are received while a test script is running, even if the script is processing triggers. You can avoid this by using the localnode.prompts4882 (on page 14-103) attribute, and by using `trigger.wait()` calls that remove the `*TRG` messages from the command queue. If the command queue fills with too many trigger entries, messages such as `abort` are not processed.

Triggering using hardware lines

You can use the digital I/O lines and TSP-Link® synchronization lines to synchronize the operations of the 2470 with those of external instruments. You can use these lines to synchronize the 2470 with other TSP-enabled instruments, including other 2470 instruments. You must use the digital I/O lines to synchronize the 2470 with other Keithley products or other non-Keithley products.

The lines are configured and controlled similarly. See Digital I/O (on page 8-12), TSP-Link System Expansion Interface (on page 9-1), and Connecting the 2470 to a Trigger Link system (on page 8-21) for more information about connections and configuration and control of the lines.

LAN triggering overview

You can send and receive triggers over the LAN interface. The 2470 supports LAN extensions for instrumentation (LXI). It has eight LAN triggers that generate and respond to LXI trigger packets.

Understanding hardware value and pseudo line state

LAN triggering and hardware synchronization are similar, except that LAN triggering uses LXI trigger packets instead of hardware signals. A bit in the LXI trigger packet called the hardware value simulates the state of a hardware trigger line. The 2470 stores the hardware value as the pseudo-line state. Only the state of the last LXI trigger packet that was sent or received is stored.

The stateless event flag is a bit in the LXI trigger packet that indicates if the hardware value should be ignored. If it is set, the 2470 ignores the hardware value of the packet and generates a trigger event. The 2470 always sets the stateless flag for outgoing LXI trigger packets. If the stateless event flag is not set, the hardware value indicates the state of the signal.

The instrument interprets changes in the hardware value of consecutive LXI trigger packets as edge transitions. Edge transitions generate trigger events. If the hardware value does not change between successive LXI trigger packets, the 2470 assumes an edge transition was missed and generates a trigger event. The following table shows edge detection in LAN triggering.

LXI trigger edge detection

Stateless event flag	Hardware value	Pseudo-line state	Falling edge	Rising edge
0	0	0	Detected	Detected
0	1	0	-	Detected
0	0	1	Detected	-
0	1	1	Detected	Detected
1	-	-	Detected	Detected

You can set the LAN trigger edge detection method in incoming LXI trigger packets. The edge that is selected also determines the hardware value in outgoing LXI trigger packets. The following table lists the LAN trigger edges.

Trigger mode	Input detected	Output generated
Either edge	Either	Negative
Falling edge	Falling	Negative
Rising edge	Rising	Positive

LAN trigger objects generate LXI trigger events, which are LAN0 to LAN7 (zero based). To specify the LAN trigger event in a command, use `LAN`*`N`*, where *`N`* is 1 to 8. `LAN1` corresponds to LXI trigger event LAN0 and `LAN8` corresponds to LXI trigger event LAN7. To specify the LAN trigger event in a command, see Trigger events (on page 8-53).

Generate LXI trigger packets

You can configure the 2470 to output an LXI trigger packet to other LXI instruments.

To generate LXI trigger packets:

1. Call the SCPI `:TRIGger:LAN<n>:OUT:CONNect:STATe` command or TSP `trigger.lanout[`*`N`*`].connect()` function.
2. Select the event that triggers the outgoing LXI trigger packet by assigning the specific event to the LAN stimulus input using the SCPI `:TRIGger:LAN<n>:OUT:STIMulus` command or TSP `trigger.lanout[N].stimulus` attribute.

Make sure to use the same LXI domain on both the 2470 instrument and the other instrument. If the 2470 has a different LXI domain from the instrument at the other end of the trigger connection, the LXI trigger packets are ignored by both instruments.

Trigger timers

You can assign trigger timers in trigger model wait, branch on event, and notify blocks. If you are using remote commands, you can set the trigger timers to be the stimulus for the LAN, blender, or TSP-Link output.

The 2470 has four independent timers.

You can set the count, delay, and time when the trigger occurs for the trigger timers. You need to set these parameters before you enable the trigger timers.

Count

The count sets the number of events to generate each time the timer generates a trigger event. Each event is separated by the trigger timer delay.

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

If you are using remote commands, the delay is set by the SCPI `:TRIGger:TIMer<n>:DELay` or TSP `trigger.timer[N].delay` command. The count is set using the SCPI command `:TRIGger:TIMer<n>:COUNt` or the TSP command `trigger.timer[N].count`.

Timer delays

You can set up the timers to perform delays. A delay is the period after the timer is triggered and before the timer generates a trigger event. All delay values are specified in seconds.

Delay lists, which are only available using remote TSP commands, allow the timer to sequence through an array of delay values. Delay lists allow the timer to use a different interval each time it performs a delay. Each time the timer generates a trigger event, it uses the next delay in the list. The timer repeats the delay list after all the elements in the delay list have been used.

If you use the trigger timer with a trigger model, make sure the trigger timer delay is set so that the readings are paced correctly.

Using SCPI commands:

To set up a 50 µs trigger timer delay for timer 2, send the command:

```
TRIGger:TIMer2:DELay 50E-6
```

Using TSP commands:

To set up a 50 µs trigger timer delay for timer 2, send the command:

```
trigger.timer[2].delay = 50e-6
```

To set up a delay list for timer 3 for delays of 2, 10, 15, and 7 s, send the command:

```
trigger.timer[3].delaylist = {2, 10, 15, 7}
```

Define when to generate a timer event

You can specify if the trigger timer starts when an event occurs or at a specific time.

On the front panel, the timer Start Condition option sets when the timer will start. If you select Trigger Event, the trigger timer starts when the event defined in Stimulus occurs. If you select Specific Date and Time, you can select the date and time when the trigger occurs.

If you are using remote comments, you can specify when timer events are generated using the SCPI `:TRIGger:TIMer<n>:STARt:GENerate` or TSP `trigger.timer[N].start.generate` command. When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses.

You can watch for a stimulus before starting the timer by using the SCPI `:TRIGger:TIMer<n>:STARt:STIMulus` command or `trigger.timer[N].start.stimulus` command.

The following example resets trigger timer 4 (TSP only), then sets it to a 0.5 s delay, a stimulus of notify 5, to occur when the timer delay elapses, and a count of 20, and enables the timer.

SCPI example

```
TRIG:TIM4:DEL 0.5
TRIG:TIM4:STAR:STIM NOTify5
TRIG:TIM4:STAR:GEN ON OFF
TRIG:TIM4:COUN 20
TRIG:TIM4:STAT ON
```

TSP example

```
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY5
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON
```

You can also set a time when the timer will start using the seconds and fractional seconds commands. (SCPI commands `:TRIGger:TIMer<n>:STARt:SECond`s and `:TRIGger:TIMer<n>:STARt:FRACtional`; TSP commands `trigger.timer[N].start.seconds` and `trigger.timer[N].start.fractionalseconds`.) When you specify a time, the timer starts immediately if the time has passed.

Timer action overruns

The timer receives an action overrun when it generates a trigger event while a timer delay is still in progress. Use the status model to monitor for the occurrence of action overruns. For details, see the Status model (on page 16-1).

Using trigger timers with timing blocks

For precise timing or if you need to synchronize timing with other execution blocks or events, you can use the trigger timer commands with trigger model wait blocks and notify blocks. You can use the trigger timer commands to add small precise delays or to start measurements or to overcome variable measurement delays. The 2470 has 1 to 4 independent timers.

For example, you can use a trigger timer to control the delay between non-sequential blocks. After creating a trigger timer, you can insert a notify block to start the timer at a specific point in the trigger model. You could then add a wait block to wait for the timer to expire.

Another example is a measure/digitize block that takes a variable amount of time. To ensure a precise time between measurements, you can create a trigger timer and define it to be a fixed interval that is longer than the longest possible measurement. Then you can set up the trigger model to include:

- A notify block that starts the trigger timer
- A measure/digitize block that makes a measurement
- A wait block that waits for the timer to expire
- A branch counter block that iterates some number of times

NOTE

Some attributes of trigger timers should not be used with the trigger model. Attributes you should not set are:

- Count value of 0 (resulting in generation of trigger events indefinitely)
- Delay lists

Remote trigger timer commands

SCPI trigger timer commands:

- :TRIGger:TIMer<n>:CLEar (on page 12-197)
- :TRIGger:TIMer<n>:COUNt (on page 12-197)
- :TRIGger:TIMer<n>:DELay (on page 12-199)
- :TRIGger:TIMer<n>:STARt:FRACtional (on page 12-199)
- :TRIGger:TIMer<n>:STARt:GENerate (on page 12-200)
- :TRIGger:TIMer<n>:STARt:OVERrun? (on page 12-200)
- :TRIGger:TIMer<n>:STARt:SEConds (on page 12-201)
- :TRIGger:TIMer<n>:STARt:STIMulus (on page 12-202)
- :TRIGger:TIMer<n>:STATe (on page 12-203)

TSP trigger timer commands:

- trigger.timer[N].clear() (on page 14-279)
- trigger.timer[N].count (on page 14-279)
- trigger.timer[N].delay (on page 14-281)
- trigger.timer[N].delaylist (on page 14-281)
- trigger.timer[N].enable (on page 14-282)
- trigger.timer[N].reset() (on page 14-283)
- trigger.timer[N].start.fractionalseconds (on page 14-284)
- trigger.timer[N].start.generate (on page 14-285)
- trigger.timer[N].start.overrun (on page 14-286)
- trigger.timer[N].start.seconds (on page 14-286)
- trigger.timer[N].start.stimulus (on page 14-287)
- trigger.timer[N].wait() (on page 14-288)

Event blenders

The ability to combine trigger events is called event blending. You can use an event blender to wait for up to four input trigger events to occur before responding with an output event.

You set the event blender operation using remote commands. You cannot set them up through the front panel.

You can program up to two event blenders for the 2470.

Event blender operations

You can use event blenders to perform logical AND or logical OR operations on trigger events. For example, trigger events can be triggered when either a manual trigger or external input trigger is detected.

When AND operation is selected, the event blender generates an event when an event is detected on all the assigned stimulus inputs.

When OR operation is selected, the event blender generates an event when an event is detected on any one of the four stimulus inputs.

Using SCPI commands:

Send the command `:TRIGger:BLENder<n>:MODE`.

Set the command to `OR` or `AND`.

Using TSP commands:

Send the command `trigger.blender[N].orenable`.

Setting the command to `true` enables OR operation; setting it to `false` enables AND operation.

Assigning blender trigger events

Each event blender has four stimulus inputs. You can assign a different trigger event to each stimulus input.

You set the blender stimulus events using remote commands. See the command descriptions for the list of events that you can assign.

Using SCPI commands:

Send the command `:TRIGger:BLENder<n>:STIMulus<m>`.

Using TSP commands:

Send the command `trigger.blender[N].stimulus[M]`.

Trigger blender action overruns

The event blenders can generate action overruns.

When the event blender operation is set to AND, overruns occur when a second event on any of its inputs is detected before an output event is generated.

When the operation is set to OR, overruns occur when two events are detected simultaneously.

Use the status model to monitor for the occurrence of action overruns. For details, see the Status model (on page 16-1).

Interactive triggering

Interactive triggering is only available if you are using the TSP command set.

If you need more control of triggering than you can get using a trigger model, you can use interactive triggering to enable your system to generate and detect trigger events anywhere in the test flow. Interactive triggering is typically used in the context of TSP script operation. For example, interactive triggering can be used when you need to make multiple source function changes or implement conditional branching to other test setups based on recent measurements.

All the 2470 trigger objects have built-in event detectors that monitor for trigger events. The event detector only monitors events generated by that object. They cannot be configured to monitor events generated by any other trigger object.

You can use the `wait()` function of the trigger object to cause the instrument to suspend command execution until a trigger event occurs or until the specified timeout period elapses. For example, use `trigger.blender[`*`N`*`].wait(`*`timeout`*`)` to suspend command execution until an event blender generates an event, where *N* is the specific event blender and *timeout* is the timeout period. After executing the `wait()` function, the event detector of the trigger object is cleared.

The following programming example illustrates how to suspend command execution while waiting for various events to occur:

```
-- Wait up to 60 seconds for timer 1 to complete its delay.
trigger.timer[1].wait(60)
-- Wait up to 30 seconds for input trigger to digital I/O line 5.
trigger.digin[5].wait(30)
```

You can use some trigger objects to generate output triggers on demand. These trigger objects are the digital I/O lines, the TSP-Link synchronization lines, and the LAN.

The programming example below generates output triggers using the assert function of the trigger object.

```
-- Generate a 20 us pulse on digital I/O line 3.
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.digout[3].pulsewidth = 20e-6
trigger.digout[3].assert()
-- Generate a rising edge trigger on TSP-Link sync line 1.
tsplink.line[1].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[1].edge = trigger.EDGE_RISING
trigger.tsplinkout[1].logic = trigger.LOGIC_POSITIVE
trigger.tsplinkout[1].assert()
-- Generate a LAN trigger on LAN pseudo line 6.
-- Note that connection parameters and commands that
-- establish a connection are not shown.
trigger.lanout[6].assert()
```

Use the release function to allow the hardware line to output another external trigger when the pulse width is set to 0.

Setting the pulse width to 0 results in an indefinite length pulse when the assert function is used to output an external trigger. When an indefinite length pulse is used, the release function must be used to release the line before another external trigger can be output.

The release function can also be used to release latched input triggers when the hardware line mode is set to synchronous. In synchronous mode, the receipt of a falling edge trigger latches the line low. The release function releases this line high in preparation for another input trigger.

The programming example below illustrates how to output an indefinite external trigger.

```
-- Set digio line 1 to output an indefinite external trigger.
digio.line[1].mode = digio.MODE_TRIGGER_OUT
trigger.digout[1].logic = trigger.LOGIC_NEGATIVE
trigger.digout[1].pulsewidth = 0
trigger.digout[1].assert()
-- Release digio line 1.
trigger.digout[1].release()
-- Output another external trigger.
trigger.digout[1].assert()
```

For information about hardware lines, see Digital I/O lines (on page 8-16) and Triggering using TSP-Link trigger lines (on page 9-6).

The programming example below checks and responds to detector overruns.

```
testOver = trigger.digin[4].overrun
if testOver == true then
   print("Digital I/O overrun occurred.")
end
```

The programming example below illustrates how to clear triggers, turn on the SMU output, and then enable a 30-second timeout to wait for a command interface trigger. When the trigger is received, the instrument performs a voltage reading.

```
-- Clear any previously detected command interface triggers.
trigger.clear()
-- Turn on output.
smu.source.output = smu.ON
-- Wait 30 seconds for a command interface trigger.
triggered = trigger.wait(30)
-- Get voltage reading.
reading = smu.measure.read()
-- Send command interface trigger to trigger the measurement.
*TRG
```

NOTE

`*TRG` cannot be used in a script. It must be sent from a remotely connected host computer.

Digital I/O

The 2470 digital I/O port provides six independently configurable digital input/output lines.

You can use these lines for digital control by writing a bit pattern to the digital I/O lines. Digital control is used for applications such as providing binning codes to a component handler. Digital control uses the state of the line to determine the action to take.

You can also use these lines for triggering by using the transition of the line state to initiate an action. The instrument can generate output trigger pulses and detect input trigger pulses. Triggering is used for applications such as synchronizing the operations of a source-measurement instrument with the operations of other instruments.

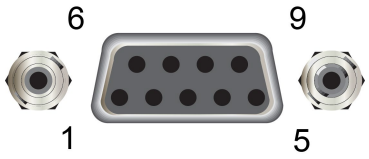
You cannot configure or directly control the digital I/O lines from the front panel. To configure and control any of the six digital input/output lines, you need to send commands to the 2470 over a remote interface. You can use either the SCPI or TSP command set.

See Remote communications interfaces (on page 2-7) for information about setting up a remote interface and choosing a command set.

Digital I/O connector and pinouts

The digital I/O port uses a standard female DB-9 connector, located on the rear panel of the 2470. You can connect to the 2470 digital I/O using a standard male DB-9 connector. The port provides a connection point to each of the six digital I/O lines and other connections, as shown in the following table.

Figure 124: 2470 digital I/O port



2470 digital I/O port pinouts

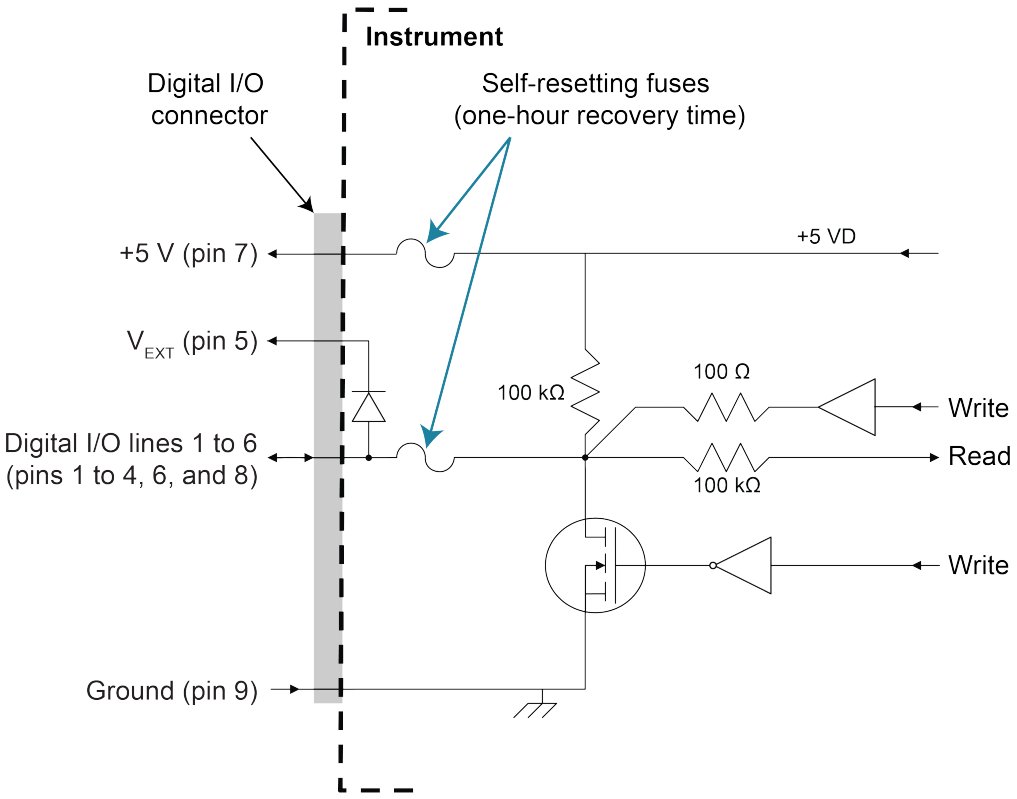
Pin	Description
1	I/O line #1
2	I/O line #2
3	I/O line #3
4	I/O line #4
5	V_{ext} line (relay flyback diode protection; maximum 33 V)
6	I/O line #5
7	+5 V line. Use this pin to drive external logic circuitry. Maximum current output is 500 mA. This line is protected by a self-resetting fuse (one-hour recovery time).
8	I/O line #6
9	Ground

Digital I/O port configuration

The following figure shows the basic configuration of the digital I/O port.

To set a line high (nominally +5 V), write a `1` to it; to set a line low (nominally 0 V), write a `0` to it. To allow an external device to control the state of the line, the line must be set to input mode or open-drain mode. An attached device must be able to sink at least 50 µA from each I/O line.

Figure 125: Digital I/O port configuration



NOTE

For additional details about the digital output, see the 2470 specifications (available at the Keithley Instruments support website (tek.com/support)).

Vext line

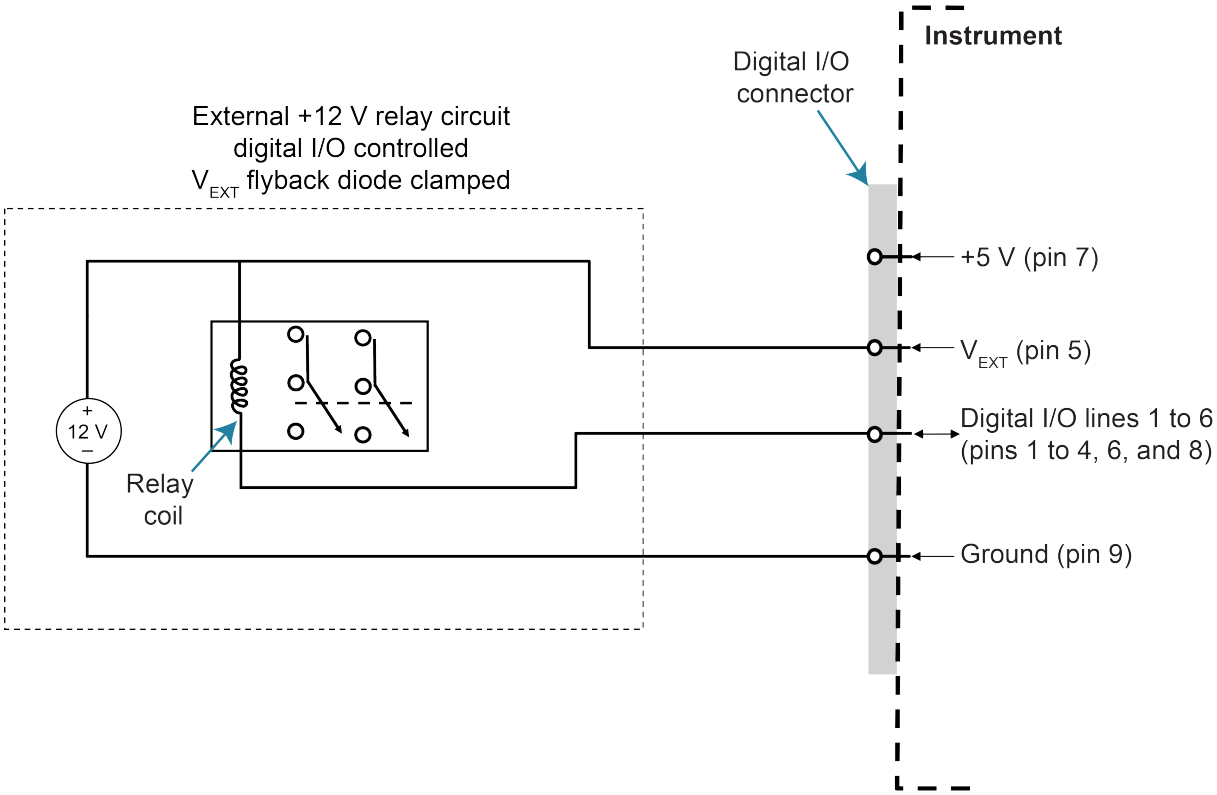
The digital I/O port provides a line (V_{ext}) with a flyback diode clamp that you can use when controlling inductive circuitry such as relay coils or low-power solenoids. You can use the built-in 5 V supply or an external voltage supply for these types of applications. The externally supplied voltage can be up to +33 V.

CAUTION

Do not apply more than 50 mA (maximum current) or exceed +33 V (maximum voltage) on the digital I/O lines. Applying current or voltage exceeding these limits may damage the instrument.

Refer to the following figure for a simplified schematic of a sample control circuit for a relay. You can externally power a different device by replacing the relay coil with the other device. The relay is actuated by configuring the corresponding digital output line. Most of these types of applications use an active-low (set the bit to 0) to turn the relay on (ON = 0 V). In the low state (0 V), the output transistor sinks current through the external device. In the high state, the output transistor is off (transistor switch is open). This interrupts current flow through the external device.

Figure 126: Digital I/O Vext (example external circuit)



+5 V line

The digital I/O port provides a +5 V output. You can use this line to drive external circuitry. The maximum current output for this line is 500 mA. A self-resetting fuse with a one-hour recovery time protects this line.

If you are using this supply to drive a relay, it should be connected to the V_{ext} line so that the relay is protected by the flyback diode clamp.

Digital I/O lines

You can place each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or acceptor

NOTE

When you configure the digital I/O lines for triggering applications, configure the output lines before the input lines. This prevents possible false input trigger detection in certain situations.

Digital control modes

If you are setting a line for digital control, you can set the line to be open-drain, output, or input, as described in the following topics.

Open-drain

When you place a line in open-drain mode, the line is configured to be an open-drain signal with a 100 kΩ pull-up resistor. This makes the line compatible with other instruments that use open-drain digital I/O lines, such as other Keithley Instruments products that only support open-drain for its digital I/O. In this mode, the line can serve as an input, an output, or both. You can read from the line or write to it. When a digital I/O line is used as an input in open-drain mode, you must write a 1 to the line to enable it to detect logic levels that are generated from external sources.

Output

When you place a line in output mode, you can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low. Unlike the input or open-drain modes, it will not respond to externally generated logic levels.

When you read the line, it shows the present output status and an event message is generated.

Input

The input mode is similar to the open-drain mode, except that a line in this mode is intended to be used strictly as an input. When you place a line in input mode, the instrument automatically writes a 1 to the line to enable it to detect externally generated logic levels.

You can read an input line, but you cannot write to it. You also cannot change the logic level while the line is in input mode. If you attempt to change the logic level of a line that is in input mode, an event message is generated.

Trigger control modes

You can use the trigger control modes to synchronize instrument operation with the operation of other instruments. These modes either detect or generate transitions in the state of the line, from high to low (falling edge) or from low to high (rising edge). The input edge detection setting of the instrument determines which type of transition is detected as an input trigger. Output triggers are typically generated in the form of a pulse. The type of transition that occurs on the leading edge of the pulse is determined by an output logic setting. The duration of the pulse is determined by a pulse width setting.

You can use the trigger control modes with interactive triggering or with the trigger model. For more information about the trigger modes and triggering, refer to Triggering (on page 8-3).

Open-drain

When you set the instrument to trigger mode and place a line in open-drain mode, the line is configured to be an open-drain signal with a 100 kΩ pull-up resistor. This makes the line compatible with other instruments that use open-drain trigger signals, such as other Keithley Instruments products that only support open-drain for its digital I/O. In this mode, you can use the line to detect input triggers or generate output triggers, or both. To use this mode successfully, you must carefully configure the input edge and output logic settings because both of these affect the initial state of the trigger line. It is recommended that you reset the line before selecting and configuring this mode.

To use the line only as a trigger input:

1. Reset the line.
2. Set the input trigger edge detection type to falling, rising, or either.

The command that sets the detection type automatically sets the line high. This enables the line to respond to and detect externally generated triggers.

Do not set the output trigger logic type to positive after setting the edge detection type. This sets the line low, which will prevent the line from operating correctly as a trigger input.

To use the line only as a trigger output:

1. Reset the line.
2. Set the output trigger logic type to negative (falling edge) or positive (rising edge).

When you set the logic type to negative, the instrument automatically sets the line high. Setting the logic type to positive automatically sets the line low.

Do not set the input trigger edge detection type after setting the positive logic type. This will set the line high, which will prevent the line from operating correctly as a trigger output.

To use the line as both a trigger input and a trigger output (falling edge triggers only):

1. Reset the line.
2. Set the output trigger logic type to negative (falling edge).
3. Set the input trigger edge detection type to falling, rising, or either.

You can use these settings for triggering applications that use Keithley Instrument products featuring Trigger Link.

Output

When you place a line in output mode, it is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger. Use the positive logic setting when you want to generate a rising edge trigger. You cannot detect incoming triggers on a line configured as a trigger output.

Input

When you place a line in input mode, it is automatically set high to allow it to respond to and detect externally generated triggers. Depending on the input edge detection setting, the line can detect falling-edge triggers, rising-edge triggers, or both.

The line cannot generate an output trigger if it is set to the trigger input mode.

Synchronous triggering

The synchronous triggering modes allow you to:

- Implement bidirectional triggering on a single trigger line
- Start operations on one or more external instruments using a single trigger line
- Wait for all instruments to complete all triggered actions

To coordinate non-Keithley instrumentation with synchronous triggering, the non-Keithley instrument must have a trigger mode that is similar to the synchronous acceptor or synchronous master trigger mode.

To use synchronous triggering, configure the triggering master to synchronous master trigger mode or the non-Keithley equivalent. Configure all other instruments in the test system to the synchronous acceptor trigger mode or equivalent.

Synchronous master

Use the synchronous master trigger mode with the synchronous acceptor mode or its non-Keithley equivalent.

Configure only one instrument as a synchronous master. Configure all other instruments that are connected to the synchronization line as synchronous acceptors.

When a digital I/O line is set to the synchronous master mode, it generates falling edge output triggers and detects rising edge input triggers on the same trigger line.

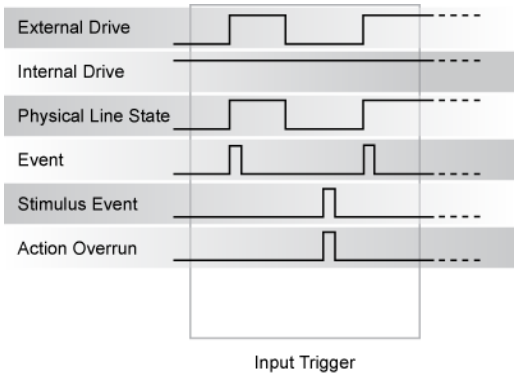
Instruments that are configured as synchronous acceptors detect the falling-edge trigger and begin their triggered actions. At the same time, they latch the line low and hold it in that state until their triggered actions complete. Each instrument configured as an acceptor releases the line upon completion of its triggered actions.

When all instruments have released the line, the line changes state and generates a rising edge trigger. This trigger is detected by the synchronous master, which then performs its next triggered action.

Input characteristics:

- All rising edges are input triggers.
- When all external drives release the physical line, the rising edge is detected as an input trigger.
- A rising edge is not detected until all external drives release the line and the line floats high.

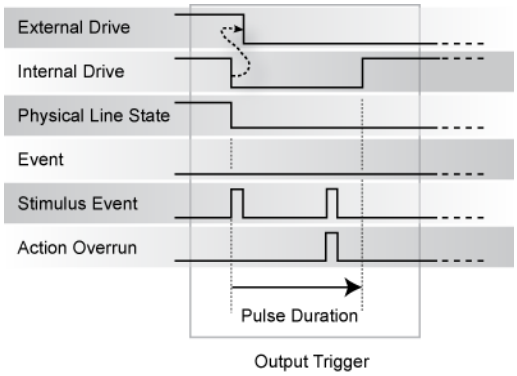
Figure 127: Synchronous master input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the TSP commands `trigger.digout[N].assert()` and `trigger.tsplinkout[N].assert()` generate a low pulse that is similar to the falling-edge trigger mode.
- An action overrun occurs if the physical line state is low when a stimulus event occurs.

Figure 128: Synchronous master output trigger



Synchronous acceptor

Use the synchronous acceptor trigger mode with the synchronous master mode or its non-Keithley equivalent.

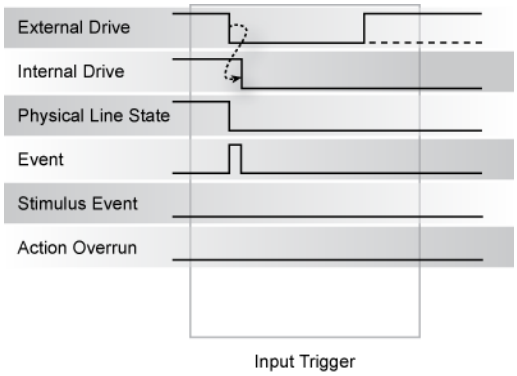
Only one instrument should be configured as a synchronous master. All other instruments connected to the synchronization line must be configured as synchronous acceptor or equivalent.

A line that is set to the synchronous acceptor mode detects falling edge input triggers and generates rising edge output triggers on the same trigger line. When a line that is configured as synchronous acceptor detects the falling edge trigger, it latches the line low and holds it in that state until all triggered actions for that instrument are complete. When the triggered actions are complete, the synchronous acceptor line releases the line. When all connected instruments have released the line, the line changes state and generates a rising edge trigger.

Input characteristics:

- The falling edge is detected as the external drive pulses the line low, and the internal drive latches the line low.

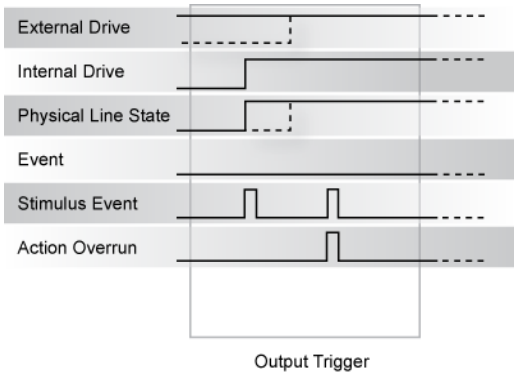
Figure 129: Synchronous acceptor input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the TSP commands `trigger.digout[N].assert()` and `trigger.tsplinkout[N].assert()` also trigger events.
- The physical line state does not change until all drives (internal and external) release the line.
- Action overruns occur if the internal drive is not latched low and a source event is received.

Figure 130: Synchronous acceptor output trigger



Connecting the 2470 to a Trigger Link system

Trigger Link (TLINK) is a Keithley Instruments proprietary trigger bus that is available on many Keithley Instrument products. The 2470 supports TLINK systems. You can connect the 2470 to a TLINK system using the Model 2450-TLINK Trigger Link Cable. Configure the appropriate 2470 digital I/O lines as open-drain inputs or outputs for compatibility with TLINK.

Remote digital I/O commands

Commands for both SCPI and TSP are summarized in the following table. You can use the digital I/O port to do the following actions:

- Perform basic steady-state digital I/O operations, such as reading and writing to individual I/O lines or reading and writing to the entire port
- Trigger the instrument when external trigger pulses are applied to the digital I/O port
- Provide trigger pulses to external devices

SCPI command TSP command	Description
:DIGital:LINE<n>:MODE (on page 12-24) digio.line[N].mode (on page 14-58)	This command sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.
A line reset is not available in SCPI; however, the line is reset when a global reset (*RST) is sent digio.line[N].reset() (on page 14-59)	This command resets digital I/O line values to their factory defaults.
:DIGital:LINE<n>:STATe (on page 12-26) digio.line[N].state (on page 14-61)	This command sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.
:DIGital:READ? (on page 12-27) digio.readport() (on page 14-61)	This command reads the digital I/O port. All six lines must be configured as digital control lines. If not, this command generates an error.
:DIGital:WRITe <n> (on page 12-27) digio.writeport() (on page 14-62)	This command writes to all digital I/O lines. All six lines must be configured as digital control lines. If not, this command generates an error.
:TRIGger:DIGital<n>:IN:CLEar (on page 12-175) trigger.digin[N].clear() (on page 14-223)	This command clears the trigger event on a digital input line.
:TRIGger:DIGital<n>:IN:EDGE (on page 12-175) trigger.digin[N].edge (on page 14-224)	This command sets the edge used by the trigger event detector on the given trigger line.
:TRIGger:DIGital<n>:IN:OVERrun? (on page 12-176) trigger.digin[N].overrun (on page 14-225)	This command returns the event detector overrun status.
Not available in SCPI trigger.digin[N].wait() (on page 14-225)	This command waits for a trigger.
Not available in SCPI trigger.digout[N].assert() (on page 14-226)	This command asserts a trigger pulse on one of the digital I/O lines.
:TRIGger:DIGital<n>:OUT:LOGic (on page 12-177) trigger.digout[N].logic (on page 14-227)	This command sets the output logic of the trigger event generator to positive or negative for the specified line.
:TRIGger:DIGital<n>:OUT:PULSewidth (on page 12-177) trigger.digout[N].pulsewidth (on page 14-228)	This command describes the length of time that the trigger line is asserted for output triggers.
Not available in SCPI trigger.digout[N].release() (on page 14-228)	This command releases an indefinite length or latched trigger.
:TRIGger:DIGital<n>:OUT:STIMulus (on page 12-178) trigger.digout[N].stimulus (on page 14-229)	This command selects the event that causes a trigger to be asserted on the digital output line.

NOTE

To use the trigger model as a stimulus to a digital I/O line, you can use the trigger model Notify block. For information on the Notify block, see Notify block (on page 8-31).

Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in the following table. Line 1 is the least significant bit.

Line #	Bit	Pin	Decimal	Hexadecimal	Binary
1	B1	1	1	0x01	000001
2	B2	2	2	0x02	000010
3	B3	3	4	0x04	000100
4	B4	4	8	0x08	001000
5	B5	6	16	0x10	010000
6	B6	8	32	0x20	100000

Digital I/O programming examples

These examples provide typical methods you can use to work with the digital I/O port.

Outputting a bit pattern

The programming examples below illustrate how to output the bit pattern 110101 at the digital I/O port. Line 1 (bit 1) is the least significant bit.

Using SCPI commands to configure all six lines as digital outputs:

```
:DIGital:LINE1:MODE DIGital, OUT
:DIGital:LINE2:MODE DIGital, OUT
:DIGital:LINE3:MODE DIGital, OUT
:DIGital:LINE4:MODE DIGital, OUT
:DIGital:LINE5:MODE DIGital, OUT
:DIGital:LINE6:MODE DIGital, OUT
```

Using SCPI commands to set the state of each line individually:

```
:DIGital:LINE6:STATe 1
:DIGital:LINE5:STATe 1
:DIGital:LINE4:STATe 0
:DIGital:LINE3:STATe 1
:DIGital:LINE2:STATe 0
:DIGital:LINE1:STATe 1
```

Using SCPI commands to set all six lines at once by writing the decimal equivalent of the bit pattern to the port:

```
:DIGital:WRITe 53
```

Using TSP commands to configure all six lines as digital outputs:

```
-- Send for loop as a single chunk or include in a script.
for i = 1, 6 do
   digio.line[i].mode = digio.MODE_DIGITAL_OUT
end
```

Using TSP commands to set the state of each line individually:

```
digio.line[1].state = digio.STATE_HIGH
digio.line[2].state = digio.STATE_LOW
digio.line[3].state = digio.STATE_HIGH
-- You can use 0 instead of digio.STATE_LOW.
digio.line[4].state = 0
-- You can use 1 instead of digio.STATE_HIGH.
digio.line[5].state = 1
digio.line[6].state = 1
```

Using TSP commands to set all six lines at once by writing the decimal equivalent of the bit pattern to the port:

```
-- You can write binary, decimal or hexadecimal values, as shown below.
-- Use binary value.
digio.writeport(0b110101)
-- Use decimal value.
digio.writeport(53)
-- Use hexadecimal value.
digio.writeport(0x35)
```

Reading a bit pattern

The programming examples below illustrate how to read part or all of a bit pattern that has been applied to the digital I/O port by an external instrument. The binary pattern is 111111 (63 decimal). Line 1 (bit 1) is the least significant bit.

Using SCPI commands:

Configure all six lines as digital inputs:

```
DIGital:LINE1:MODE DIGital, IN
DIGital:LINE2:MODE DIGital, IN
DIGital:LINE3:MODE DIGital, IN
DIGital:LINE4:MODE DIGital, IN
DIGital:LINE5:MODE DIGital, IN
DIGital:LINE6:MODE DIGital, IN
```

Read the state of Line 2:

```
DIGital:LINE2:STATe?
```

Value returned is 1.

Read the state of Line 3:

```
DIGital:LINE3:STATe?
```

Value returned is 1.

Read the value applied to the entire port:

```
DIGital:READ?
```

Value returned is 63, which is the decimal equivalent of the binary bit pattern.

Using TSP commands:

```
-- Configure all six digital I/O lines as digital inputs.
-- You can also use a for loop.
digio.line[1].mode = digio.MODE_DIGITAL_IN
digio.line[2].mode = digio.MODE_DIGITAL_IN
digio.line[3].mode = digio.MODE_DIGITAL_IN
digio.line[4].mode = digio.MODE_DIGITAL_IN
digio.line[5].mode = digio.MODE_DIGITAL_IN
digio.line[6].mode = digio.MODE_DIGITAL_IN
-- Read and then print the state of Line 2 (bit 2).
b2 = digio.line[2].state
print(b2)
```

The value returned is `digio.STATE_HIGH`.

```
-- Print the state of Line 3 (bit 3).
print(digio.line[3].state)
```

The value returned is `digio.STATE_HIGH`.

```
-- Read and then print the value applied to the entire port.
port = digio.readport()
print(port)
```

The value returned is 63, which is the decimal equivalent of the binary bit pattern.

Trigger model

The trigger model controls the sequence in which source and measure actions occur. The 2470 trigger model is flexible, allowing you to control as much or as little as needed for your measurement application.

When you are setting up a trigger model, you can choose the following options:

- Wait for an event to occur before making another measurement
- Notify other equipment and timers that an event has occurred
- Wait for another piece of equipment to signal completion
- Use measure configuration lists to apply different measure settings dynamically during trigger model operation
- Specify delays between events and measurements
- Use source configuration lists to sweep source settings and values
- Turn the source on and off with programmable delays to create pulses
- Store measurements into a given buffer until an event occurs, then switch to another buffer
- Conditionally take actions based on whether the measurement falls within set limits

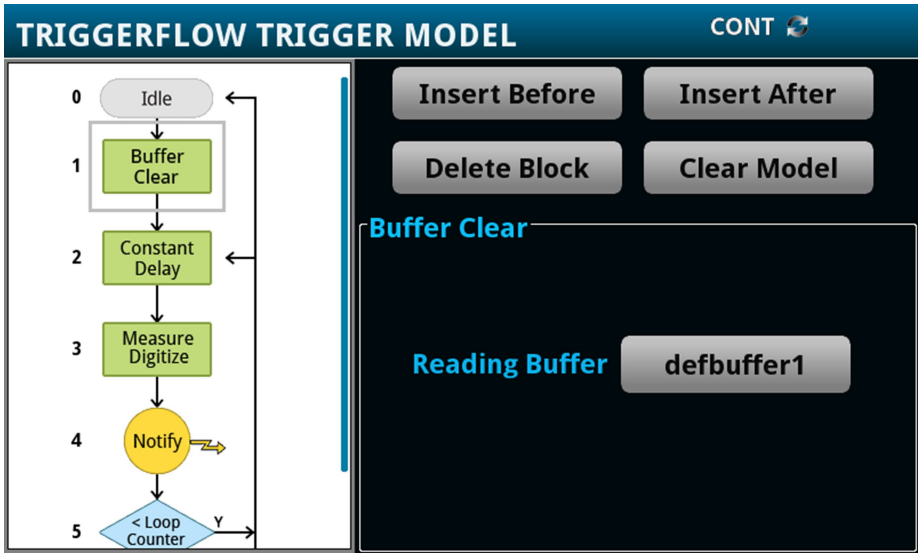
Additional options are detailed in the following sections.

The 2470 includes trigger-model templates to allow you to quickly implement a trigger model. You can also set up your own trigger models.

TriggerFlow Trigger Model

The TriggerFlow® Trigger Model diagram on the front panel provides an interactive visual flowchart of the trigger model. An example of the TriggerFlow diagram when the SimpleLoop trigger template is selected is shown in the following figure.

Figure 131: TriggerFlow Trigger Model screen



You can swipe the TriggerFlow diagram to view the entire trigger model. You can also insert, delete, and adjust settings for the blocks in the trigger model. Block numbers and branching paths are automatically adjusted when you insert or delete blocks.

To insert a block, select a block in the TriggerFlow diagram. Select Insert Before to insert a block immediately above the selected block. Select Insert After to insert a block immediately below the selected block.

To delete block, select a block the TriggerFlow diagram and select Delete Block.

To remove all blocks from the TriggerFlow diagram, select Clear Model.

To change the settings for a trigger block, select the block. The settings that are available for that block are displayed on the right side of the screen. Refer to the descriptions of the blocks in the Trigger model blocks (on page 8-26) for detail on the options that are available for each block.

Trigger-model blocks

Each trigger model consists of blocks that can be combined to create the trigger model. The blocks can be combined from the front panel or by sending remote commands. You can connect a maximum of 63 blocks as needed to control the instrument.

You can combine trigger-model blocks as you would construct a flowchart diagram. Trigger models are created using these fundamental blocks:

- **Wait:** Waits for an event to occur before the flow continues
- **Action:** Starts an action in the instrument, such as making a measurement or clearing a buffer
- **Notify:** Notifies other equipment or timers that an event has occurred
- **Branch:** Branches when a condition has been satisfied

Each type of block is described in the following topics.

Buffer clear block

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

For more information about reading buffers, refer to Reading buffers (on page 6-1).

When you select the buffer clear block, the following option is available.

Setting	Description
Reading Buffer	The name of the buffer to clear.

Measure/Digitize block

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

Digitized measurements are not a feature on the 2470. However, you can use this command to communicate with other Keithley instruments that do offer the digitized measurements feature and to share code with other Keithley instruments.

Firmware versions of the 2470 before version 1.7.0 had a separate measure block. If you have code that is using that block, it works in this version of the 2470 firmware.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1.

When you select the Measure/Digitize block, the following options are available.

Setting	Description
Count	Specifies the number of readings to make before moving to the next block in the trigger model. You can select: ■ A specific number ■ Infinite: Run continuously until stopped. ■ Stop Infinite: Stop the infinite setting. ■ Auto: Use the measure count setting that is active for the function for the trigger model block.
Reading Buffer	The name of the buffer where the readings are stored.

Source Output block

The source output block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as SOURCE_OUTPUT.

When you select the Source Output block, the following options are available.

Setting	Description
Output State	Select **On** to turn the output on or **Off** to turn the output off.

Timing blocks

You can use the timing blocks to control the timing of actions in the trigger model. The timing blocks include the wait and delay blocks.

Wait block

The wait block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to three events for each wait block.

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. By default, when the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action — you cannot combine `AND` and `OR` logic in a single wait block.

When you select the wait block, the following options are available.

Setting	Description
Clear	To clear previously detected trigger events when entering the wait block, select Enter. To immediately act on any previously detected triggers and not clear them, select Never.
Event 1	An event that must occur before the trigger block will continue.
Event 2	Optional. An event that must occur before the trigger block will continue.
Event 3	Optional. An event that must occur before the trigger block will continue.
Event Logic	Optional. Determines if all the defined events must occur or if at least one of the events must occur. Select AND if all the defined events must occur. Select OR if at least one of the events must occur.

The event can be any of the events described in the following table.

Event	Description
Blender	Wait for the events set by an event blender.
Command	A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`

Event	Description
Digital Input	Line edge detected on a digital input line. When you select this option, you select the digital input to monitor. After selecting the digital input line, choose Settings to select the type of edge (falling, rising, or either).
Display TRIGGER Key	Front-panel TRIGGER key press.
LAN Input Trigger	An LXI trigger packet is received on the LAN trigger object. When you select this option, you select the LAN trigger to monitor. After you select the line, choose Settings to select the type of edge (falling, rising, or either).
None	No trigger event.
Source Limit	Source has reached the specified limit.
Timer	A trigger timer expired. When you select this option, you select the timer to monitor. After selecting the timer, choose Settings to define the timer. Refer to Trigger timers (on page 8-5) for detail on the options. For a timer to expire, you must start it. One method to start the timer in the Trigger Model is to include a Notify block before this block. Set the Notify block to use the same timer.
TSP-Link Input	Line edge detected on a TSP-Link synchronization line. When you select this option, you select the TSP-Link line to monitor. After selecting the TSP-Link line, choose Settings to select the type of edge (falling, rising, or either).

If you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block. For information, see Branching blocks (on page 8-36).

Constant delay block

When trigger model execution reaches a delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

When you select the Constant Delay block, the following option is available.

Setting	Description
Delay	The amount of time to delay in seconds.

Dynamic delay block

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Each source function can also have up to five unique user delay times (S1 to S5). The delay time is set by the user-delay command, which is only available over a remote interface. If you are using SCPI, the user delay commands are [:SENSe[1]]:<function>:DELay:USER<n> (on page 12-48) and :SOURce[1]:<function>:DELay:USER<n> (on page 12-76). If you are using TSP, they are smu.measure.userdelay[N] (on page 14-168) and smu.source.userdelay[N] (on page 14-200).

This delay can be different for every index in a configuration list. This makes it possible to have a delay that changes as a configuration list progresses.

When you select the Dynamic Delay block, the following option is available.

Setting	Description
User Delay	The user delay to recall.

Notify block

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

Setting up the notify block using the front panel:

When you set up the notify blocks using the front panel, you select the line or timer to notify. You also set specifics regarding the line. The stimulus and logic for input and output lines are set up automatically. The notify event number is also set automatically and is displayed at the bottom of the Notify definition screen.

When the trigger model executes a notify block, the instrument generates the SCPI event `NOTify<n>` or TSP event `trigger.EVENT_NOTIFY`*N*. You can assign this event to a command that takes an event. For example, if you want a notify block to trigger a digital I/O line, insert a notify block into the trigger model, assign it a notify event and then connect it to the stimulus of the digital I/O line that you want to drive with the notify event.

If you define a LAN trigger from the front panel, you are asked if you want to initiate the LAN connection. You must initiate the connection to use the LAN triggers.

Setting up the notify block using remote commands:

When you set up the notify block using remote commands, you define the notify event number. You need to set up the lines that use the notify event as a stimulus as separate commands.

In the following example, you define trigger model block 5 to be the notify 2 event. You can then assign the notify 2 event to be the stimulus for digital output line 3. To do this, send the following commands in SCPI:

```
:TRIG:BLOC:NOT 5, 2
:TRIG:DIG3:OUT:STIMulus NOTify2
```

In TSP, send the commands:

```
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

If digital I/O line 3 is connected to another instrument, this causes the trigger execution to wait for the other instrument to indicate that it is ready.

Front panel options

When you select the Notify block from the front panel, the following options are available.

Setting	Description
Notify	The line or timer that is notified: ■ Digital Output Line: Select the digital output event (line 1 through line 6) ■ TSP-Link Output Line: Select the TSP-Link output event (line 1 through 3) ■ Timer: Select the timer event (timer 1 through 4) ■ LAN Output: Select the LAN output line (line 1 to 8)
Settings	The available settings depend on the Notify selection. For the digital, TSP-Link, and LAN output lines, select the pulse logic (negative or positive).

Log event block

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

When you select the Log Event block, the following options are available.

Setting	Description
Event Type	The event number or type: ■ Abort Model: Stop the trigger model and log a warning message ■ Information *N*: Logs an information message in the event log ■ Warning *N*: Logs a warning in the event log ■ Error *N*: Logs an error in the event log Where *N* is 1 to 4; you can define up to four of each type.
Message	A message that you define.

Reset Branch Count

This block creates a block in the trigger model that resets a branch counter to 0.

When you select the Reset Branch Count block, the following option is available.

Setting	Description
Counter Block	Enter a number from 1 to 63.

Configuration list blocks

You can use configuration list blocks to recall settings that are stored in a configuration list. When the trigger model reaches a configuration list block, the commands in the configuration list are executed.

The trigger model blocks that recall configuration lists are:

- Configuration recall
- Configuration next
- Configuration previous

For detail on configuration lists, see Configuration lists (on page 4-82).

Config list recall block

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration lists must be defined before you can use this block. If one of the indices for the configuration list changes, verify that the trigger model count is still accurate.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Recall block.

When you select the Config List Recall block, the following options are available.

Setting	Description
Config List	The name of the configuration list to recall the index from.
Recall Index	The index to recall.
Config List 2	The name of the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list.
Disable Config List 2	Sets the second configuration to None.
Recall Index 2	The index to recall from the second configuration list.

Config list next block

The config list next block recalls the settings at the next index of a source or measure configuration list.

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger model block. If the configuration list has recalled an index with the recall block, the next index in the list is recalled instead of the first. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the next block is reached after that recall, index 2 is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Next block.

When you select the Config List Next block, the following option is available.

Setting	Description
Config List	The name of the configuration list from which to recall the next index.
Config List 2	The name of the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list.
Disable Config List 2	Sets the second configuration to None.

Config list prev block

The Config List Prev block defines a trigger model block that recalls the settings stored at the previous index in a source or measure configuration list if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are each recalled from the previous index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger model block. If the configuration list has recalled an index with the recall block, the previous index in the list is called instead of the last. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the previous block is reached after that recall, the last index is recalled. However, if the recall block recalled index 3, the previous block would recall index 2. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Prev block.

When you select the config list prev block, the following option is available.

Setting	Description
Config List	The name of the configuration list from which to recall the previous index.
Config List 2	The name of the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list.
Disable Config List 2	Sets the second configuration to None.

Digital input/output block

The digital I/O block defines a trigger model block that sets the lines on the digital I/O port high or low.

To set the lines on the digital I/O port high or low, you can send an output line bit pattern. The pattern can be specified as an integer value, or, if you are using the TSP command set, a six-bit binary or hexadecimal. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63). If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command). The digital line settings are only available through remote commands.

When you select the digital I/O block, the following options are available.

Setting	Description
Out Line Mask	Sets the output line bit mask (0 to 63).
Out Line Pattern	Sets the value that specifies the output line bit pattern (0 to 63).

Branching blocks

A branch block goes to a trigger block other than the sequential execution block. For example, if you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block to define when the wait block is enabled. You can use the Branch Once block to create a bypass and skip the wait block the first time the trigger model runs. This makes it possible to avoid deadlock when multiple instruments are being synchronized and each one is waiting for notification from the other one to start the trigger model.

Loop Counter block

When trigger model execution reaches a loop counter block, it goes to a specified block until the count value is reached. When the counter exceeds the count value, trigger model execution ignores the branch, continues to the next block in the sequence, and resets the counter.

The counter is reset to 0 when the trigger model starts. It is incremented each time trigger model execution reaches the counter block.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting.

When you select the Loop Counter block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the counter is less than the Target Count value.
Target Count	The number of times to repeat.

On event block

The branch-on-event block branches to a specified block when a specified trigger event occurs. If the trigger event has not yet occurred when trigger model execution reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time trigger model execution reaches the branch-on-event block, it goes to the branching block.

Trigger events are reset when the trigger model is at the start block, so only events that occur after you start trigger model execution are detected by the branch-on-event block. The event is also reset after trigger model execution completes the branching block.

When you select the branch-on-event block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the specified event occurs.
Event	The event that causes this block to branch.

The event can be any of the events described in the following table.

Event	Description
Blender	Wait for the events set by an event blender.
Command	A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`
Digital Input	Line edge detected on a digital input line. When you select this option, you select the digital input to monitor. After selecting the digital input line, choose Settings to select the type of edge (falling, rising, or either).
Display TRIGGER Key	Front-panel TRIGGER key press.
LAN Input Trigger	An LXI trigger packet is received on the LAN trigger object. When you select this option, you select the LAN trigger to monitor. After you select the line, choose Settings to select the type of edge (falling, rising, or either).
None	No trigger event.
Source Limit	Source has reached the specified limit.
Timer	A trigger timer expired. When you select this option, you select the timer to monitor. After selecting the timer, choose Settings to define the timer. Refer to Trigger timers (on page 8-5) for detail on the options. For a timer to expire, you must start it. One method to start the timer in the Trigger Model is to include a Notify block before this block. Set the Notify block to use the same timer.
TSP-Link Input	Line edge detected on a TSP-Link synchronization line. When you select this option, you select the TSP-Link line to monitor. After selecting the TSP-Link line, choose Settings to select the type of edge (falling, rising, or either).

For information on trigger events, see Using trigger events to start actions in the trigger model (on page 8-52).

Constant Limit block

The Branch Constant Limit block defines a trigger model block that goes to a specified block if a measurement meets preset criteria.

The measurement block must be a measurement block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measurement block is used.

If the limit A is more than the limit B, the instrument automatically swaps the values so that the lesser value is used as the lower limit.

You can use this block to create a binning application by having the block branch to a digital I/O block, followed by a branch always block. Multiple tests can be chained together by repeating this.

NOTE

To use limits that vary programmatically, use the branch-on-dynamic-limits block.

When you select the constant limit block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the measurement meets the defined criteria.
High Limit	The upper limit that the measurement is compared against. If the type is set to: ■ Inside: The high limit that the measurement is compared against ■ Above: The measurement must be above this value ■ Below: This value is ignored ■ Outside: The high limit that the measurement is compared against
Limit Type	How the limits are compared: ■ Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value ■ Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected ■ Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected ■ Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
Low Limit	The lower limit that the measurement is compared against. If the type is set to: ■ Inside: The low limit that the measurement is compared against ■ Above: This value is ignored ■ Below: The measurement must be below this value ■ Outside: The low limit that the measurement is compared against
Measure/Digitize Block	The block number of the block that makes the reading to be compared; from the front panel, you can set this to Previous to use the previous measure/digitize block.

NOTE

When you select a Limit Type of Inside or Outside, two buttons display below the Limit Type button. The button on the left is the Low Limit and the button on the right is the High Limit.

Dynamic Limit block

The branch-on-dynamic-limits block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

When you select the dynamic limit block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the measurement meets the defined criteria.
Limit Number	The limit that is used for this block, 1 or 2.
Limit Type	How the limits are compared: ▪ Inside: The measurement is within the limits ▪ Above: The measurement is above the high limit ▪ Below: The measurement is below the low limit ▪ Outside: The measurement is outside the limits
Measure/Digitize Block	The block number of the block that makes the reading to be compared; from the front panel, you can set this to Previous to use the previous measure/digitize block.

Delta block

The branch on delta block defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

When you select the delta block, the following options are available.

Setting	Description
Branch to Block	The block number of the trigger model block to execute when the difference between the measurements is less than or equal to the Target Delta.
Measure/Digitize Block	The block number of the measure/digitize block that makes the measurements to be compared; if this is `0` or undefined, the trigger model uses the previous measure/digitize block.
Target Delta	The value against which the block compares the difference between the measurements.

Always block

When the trigger model reaches a branch-always block, it goes to the block that you specified.

When you select the always block, the following option is available.

Setting	Description
Branch to Block	The block number to execute when the trigger model reaches the always block.

Once block

When the trigger model reaches a branch-once block, it goes to a specified block the first time it is encountered in the trigger model. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

You can use this block to create a bypass. For example, you might place a branch-once block before a wait block to skip the wait block on the first pass of the trigger model.

The once block is reset when the trigger model reaches the idle state. Therefore, the branch-once block will always execute the first time the trigger model encounters this block.

When you select the Once block, the following option is available.

Setting	Description
Branch to Block	The block number to execute the first time the trigger model reaches the once block.

Once excluded block

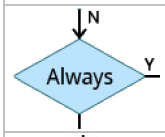
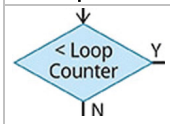
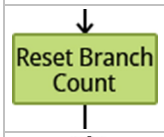
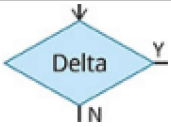
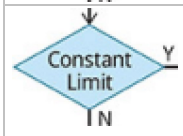
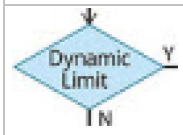
The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

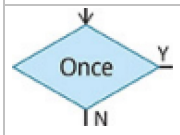
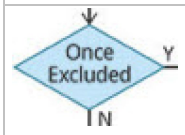
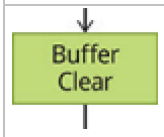
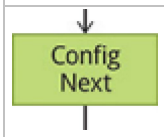
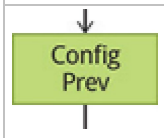
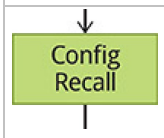
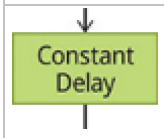
The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

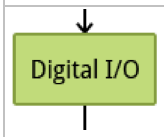
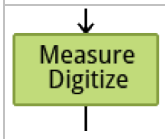
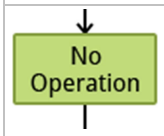
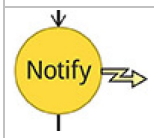
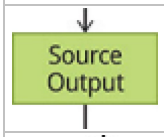
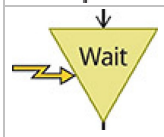
When you select the once excluded block, the following option is available.

Setting	Description
Branch to Block	The block number to execute after the first time the trigger model reaches the once excluded block.

Trigger block summary

Front-panel icon	SCPI command TSP command	Block description
Not applicable	:TRIGger:BLOCk:LIST? (on page 12-166) trigger.model.getblocklist() (on page 14-239)	This returns the settings for all trigger model blocks
	:TRIGger:BLOCk:BRANch:ALWays (on page 12-149) trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS (on page 14-253)	This defines a trigger model block that always goes to a specific block
	:TRIGger:BLOCk:BRANch:COUNter (on page 12-150) trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 14-253)	This defines a trigger model block that branches to a specified block a specified number of times
Not applicable	:TRIGger:BLOCk:BRANch:COUNter:COUNt? (on page 12-151) trigger.model.getbranchcount() (on page 14-240)	This returns the count value of the trigger model counter block
	:TRIGger:BLOCk:BRANch:COUNter:RESet (on page 12-152) trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT (on page 14-274)	This creates a block in the trigger model that resets a branch counter to 0
	:TRIGger:BLOCk:BRANch:DELTa (on page 12-153) trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA (on page 14-254)	This defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria
	:TRIGger:BLOCk:BRANch:EVENt (on page 12-154) trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT (on page 14-258)	This branches to a specified block when a specified trigger event occurs
	:TRIGger:BLOCk:BRANch:LIMit:CONStant (on page 12-155) trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT (on page 14-255)	This defines a trigger model block that goes to a specified block if a measurement meets preset criteria
	:TRIGger:BLOCk:BRANch:LIMit:DYNamic (on page 12-156) trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC (on page 14-257)	This defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCk:BRANch:ONCE (on page 12-158) trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE (on page 14-260)	This causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model
	:TRIGger:BLOCk:BRANch:ONCE:EXCLuded (on page 12-158) trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED (on page 14-260)	This causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time
	:TRIGger:BLOCk:BUFFer:CLEar (on page 12-159) trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR (on page 14-261)	This defines a trigger model block that clears the reading buffer
	:TRIGger:BLOCk:CONFig:NEXT (on page 12-160) trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT (on page 14-262)	This recalls the settings at the next index of a source or measure configuration list
	:TRIGger:BLOCk:CONFig:PREVious (on page 12-161) trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV (on page 14-263)	This defines a trigger model block that recalls the settings stored at the previous index in a source or measure configuration list
	:TRIGger:BLOCk:CONFig:RECall (on page 12-162) trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL (on page 14-265)	This recalls the system settings that are stored in a source or measure configuration list
	:TRIGger:BLOCk:DELay:CONStant (on page 12-163) trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT (on page 14-266)	This adds a constant delay to the execution of a trigger model
	:TRIGger:BLOCk:DELay:DYNamic (on page 12-164) trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 14-267)	This adds a user delay to the execution of the trigger model

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCk:DIGital:IO (on page 12-165) trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO (on page 14-268)	This trigger model block that sets the lines on the digital I/O port high or low
	:TRIGger:BLOCk:LOG:EVENt (on page 12-166) trigger.model.setblock() — trigger.BLOCK_LOG_EVENT (on page 14-269)	This allows you to log an event in the event log when the trigger model is running
	:TRIGger:BLOCk:MDIGitize (on page 12-167) trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE (on page 14-270)	This defines a trigger block that makes or digitizes a measurement, depending on the active function
	:TRIGger:BLOCk:NOP (on page 12-170) trigger.model.setblock() — trigger.BLOCK_NOP (on page 14-272)	This creates a placeholder that performs no action in the trigger model; available only using remote commands
	:TRIGger:BLOCk:NOTify (on page 12-170) trigger.model.setblock() — trigger.BLOCK_NOTIFY (on page 14-273)	This defines a trigger model block that generates a trigger event and immediately continues to the next block
	:TRIGger:BLOCk:SOURce:STATe (on page 12-171) trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT (on page 14-275)	This defines a trigger block that turns the output source on or off
	:TRIGger:BLOCk:WAIT (on page 12-172) trigger.model.setblock() — trigger.BLOCK_WAIT (on page 14-276)	This defines a trigger model block that waits for an event before allowing the trigger model to continue

Trigger-model templates

The 2470 includes trigger-model templates for common applications. You can use these templates without changing them, or you can modify them to meet the needs of your application.

The trigger-model templates include:

- **Empty:** Clears the present trigger model.
- **ConfigList:** Creates a trigger model that loads a source and measure configuration list. The lists are iterated until every index in the configuration list with fewer indexes has been loaded. For example, if the measure list has seven indexes and the source configuration list has 10, it will iterate through seven indexes of the source list and all of the measure list. However, if the source list has three indexes and the measure list has five, it will iterate through three indexes of measure list and all of the indexes in the source list. At each index when the output is turned on, a measurement is made and the output is turned off.
- **LogicTrigger:** Creates a trigger model that waits on an input line, delays, makes a measurement, and sends out a trigger on the output line a specified number of times.
- **SimpleLoop:** Creates a trigger model that makes a specified number of readings. A count parameter defines the number of readings.
- **DurationLoop:** Creates a trigger model that makes continuous measurements for a specified amount of time. When you start this trigger model, the output is turned on.
- **LoopUntilEvent:** Creates a trigger model that makes continuous measurements until a specified event occurs.
- **GradeBinning:** Creates a trigger model that successively measures components and compares their readings to high or low limits to grade components.
- **SortBinning:** Creates a trigger model that successively measures components and compares their readings to high or low limits to sort components.

NOTE

Settings for the GradeBinning and SortBinning templates are shared. If you switch between these two templates, be aware that settings are retained.

Preparations for using a trigger-model template

Before starting the trigger model, you need to set up your instrument for testing, including the source and measure settings and source and measure configuration lists.

When you load a trigger-model template, the instrument overwrites any existing trigger models.

NOTE

If you select Templates and only the Empty trigger model appears, select Empty to see the full list of template options.

Using a trigger-model template to develop a trigger model

The 2470 includes trigger-model templates that you can use as a starting point for developing your trigger model.

After modifying a trigger-model template, you can save it in a saved setup for future use. See Saving setups (on page 3-45) for information on how to save a configuration.

The trigger model is changed to the selected template when you change a setting, press the HOME key, or press the MENU key.

NOTE

If you select a trigger-model template, but then customize it using options on the Trigger Configure menu, Templates displays `Custom`.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Templates**. The TRIGGER MODEL TEMPLATES screen is displayed.
3. Use **Templates** to select the trigger-model template.
4. Change the settings for the template as needed.
5. Select **MENU**.
6. Under Trigger, choose **Configure**. The blocks for the trigger-model template are displayed.
7. Modify the blocks as needed. See Assembling trigger-model blocks (on page 8-47).
8. When the blocks are set up, select **HOME**.
9. Select the measurement method indicator and choose **Manual Trigger Model**. This sets the TRIGGER key to initiate the trigger model. See Measurement method indicator (on page 3-12) for additional information.
10. Press the **TRIGGER** key to initiate the trigger model.

Using SCPI commands:

See the descriptions of the `TRIGger:LOAD` commands for details on the options available for each trigger-model template.

- :TRIGger:LOAD "ConfigList" (on page 12-185)
- :TRIGger:LOAD "DurationLoop" (on page 12-186)
- :TRIGger:LOAD "Empty" (on page 12-187)
- :TRIGger:LOAD "GradeBinning" (on page 12-188)
- :TRIGger:LOAD "LogicTrigger" (on page 12-190)
- :TRIGger:LOAD "LoopUntilEvent" (on page 12-191)
- :TRIGger:LOAD "SimpleLoop" (on page 12-192)
- :TRIGger:LOAD "SortBinning" (on page 12-193)

Using TSP commands:

See the descriptions of the `trigger.model.load()` command for details on the options available for each trigger-model template.

- trigger.model.load() — ConfigList (on page 14-241)
- trigger.model.load() — DurationLoop (on page 14-242)
- trigger.model.load() — Empty (on page 14-243)
- trigger.model.load() — GradeBinning (on page 14-244)
- trigger.model.load() — LogicTrigger (on page 14-245)
- trigger.model.load() — LoopUntilEvent (on page 14-246)
- trigger.model.load() — SimpleLoop (on page 14-248)
- trigger.model.load() — SortBinning (on page 14-250)

Assembling trigger-model blocks

This section describes the basic concepts you need to understand to assemble trigger-model blocks.

Sequencing trigger-model blocks

You can set up trigger-model blocks from the front panel or by using remote commands.

Trigger-model blocks must be sequenced in order; you cannot skip numbers. When the trigger model completes the last block in the trigger model, the trigger model returns to the idle state. Idle is considered to be execution block 0. Branching to block 0 effectively stops the trigger model.

As the trigger model reaches each block, the action defined by that block is started and completed before the trigger model moves to the next block. Blocks do not overlap.

The trigger model steps through the blocks in sequential order. You can set up branching blocks to allow nonsequential actions to occur. See Branching blocks (on page 8-36) for detail on how to use the branching blocks.

If you skip block numbers, when you initiate the trigger model, the trigger model generates an event message that reports the missing block. You can view and delete the missing blocks on the front-panel TriggerFlow® screen. If you delete them using the front-panel options, the remaining blocks are resequenced.

You can have up to 63 blocks in a trigger model.

Working with the trigger model

You can change existing trigger-model blocks through the front panel or by sending a remote command. The blocks are redefined with the new parameters.

When you define the trigger model using remote commands, you can send blocks in any order. For example, you can define block 5 before defining blocks 1 to 4. However, you cannot run a trigger model with undefined blocks.

If you skipped a block, you can use the no operation block to define a block that will not affect the trigger model and save the effort of resequencing the other blocks. The no operation block is available through the remote commands only (SCPI command :TRIGger:BLOCk:NOP (on page 12-170) or TSP command trigger.model.setblock() — trigger.BLOCK_NOP (on page 14-272)).

Determining the structure of the existing trigger model

You can retrieve the existing trigger model structure from the front panel or by using remote commands.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Configure**. The trigger model is displayed.
3. If the trigger model is longer than one screen, swipe the TriggerFlow diagram to scroll up or down.
4. To view the settings for a block, select the block. The settings are displayed on the right.
5. For a description of a setting, press and hold the button and press the **HELP** key.

For additional information on the blocks, refer to the block descriptions under Trigger model blocks (on page 8-26).

Using SCPI commands:

To retrieve the settings for all trigger model blocks, send the command:

```
:TRIGger:BLOCk:LIST?
```

Using TSP commands:

To check the settings for a block, send the command:

```
print(trigger.model.getblocklist())
```

NOTE

To retrieve the TSP code for trigger model blocks that are entered through the front panel, change the Event Log "Command" setting to On. Refer to Using the event log (on page 3-50) for additional information.

Improving the performance of a trigger model

To improve the performance of a trigger model:

- Reduce the number of blocks to less than 15.
- Do not use multiple reading buffers.
- Use four or fewer delay blocks.
- Use four or fewer measure/digitize blocks.
- Do not have multiple blocks waiting on the same event.
- Verify that constant delay blocks are set to less than 254 ms.
- Limit use of configuration list blocks.

Action overruns

An action overrun occurs when a trigger object receives a trigger event and is not ready to act on it. The action overruns of all trigger objects are reported in a command for the associated trigger object. See the appropriate sections on each trigger object for further details on conditions under which an object generates an action overrun.

Some examples of action overruns include the following:

- `trigger.blender[N].overrun`
- `trigger.digin[N].overrun`
- `trigger.extin.overrun`
- `trigger.lanin[N].overrun`
- `trigger.timer[N].overrun`
- `trigger.tsplinkin[N].overrun`

Running the trigger model

You can run the trigger model from the front panel or by using remote commands.

When you run the trigger model, the existing instrument settings are used for any actions unless you assigned configuration lists to the trigger model.

Trigger-model operation is an overlapped process. This means that you can run other commands while a trigger model is running if they do not conflict with trigger-model operation. For example, you can print the buffer contents, but you cannot change the source-measure function.

The initiate command is the overlapped command that starts the process. The command interface is available immediately after the instrument executes the initiate command so that other commands can be executed while the trigger model is running.

If you change from remote to local control, the trigger model measurement method remains selected until you change it. To change the measurement method, see Switching between measurement methods (on page 8-2).

If you change from remote to local control or from local to remote control while a trigger model is running, the trigger model is aborted.

Starting the trigger model

Using the front panel:

1. Press the front-panel **TRIGGER** key for 2 s. A dialog box displays the available trigger methods. The presently selected method is highlighted.
2. Select **Initiate Trigger Model**.
3. If the instrument is controlled remotely, a confirmation screen is displayed. Select **Yes** to change to front-panel control and start the trigger model.

Using SCPI commands:

Send the command:

```
:INITiate
```

Using TSP commands:

Send the command:

```
trigger.model.initiate()
```

Aborting the trigger model

You can stop the trigger model while it is in progress. When you stop the trigger model, all trigger model commands on the instrument are terminated, including sweeps.

Using the front panel:

Press the **TRIGGER** key for two seconds and select **Abort Trigger Model**.

Using SCPI commands:

Send the command:

```
:ABORt
```

Using TSP commands:

Send the command:

```
trigger.model.abort()
```

Pausing and resuming the trigger model

You can pause the trigger model while it is in progress by using the pause command. To restart the trigger model after pausing, use the resume command.

Using SCPI commands:

To pause, send the command:

```
:TRIGger:PAUSe
```

To restart, send the command:

```
:TRIGger:RESume
```

Using TSP commands:

To pause, send the command:

```
trigger.model.pause()
```

To restart, send the command:

```
trigger.model.resume()
```

Checking the state of the trigger model

The trigger model can be in one of several states. The state is shown in the indicator bar on the home screen of the instrument. You can also check the status using remote commands.

The following table describes the trigger model states. This table also describes the indicator that is shown on the front panel and the feedback you get from the remote interface.

Front-panel indicator	SCPI remote command feedback	TSP remote command feedback	Description
N/A	`ABORTED`	`trigger.STATE_ABORTED`	The trigger model was stopped before it completed
CONT	Not available through remote interface	Not available through remote interface	Instrument is not using the trigger model; it is making measurements continuously
IDLE	`IDLE` or `EMPTY`	`trigger.STATE_IDLE` or `trigger.STATE_EMPTY`	Trigger model is stopped, or no blocks are defined
INACT	`INACTIVE`	`trigger.STATE_INACTIVE`	Instrument encountered system settings that do not yield a reading
MAN	Not available through remote interface	Not available through remote interface	Instrument is not using trigger model; makes measurements when you press the front-panel TRIGGER key
RUN	`RUNNING`	`trigger.STATE_RUNNING`	Trigger model is running
WAIT	`WAITING`	`trigger.STATE_WAITING`	The trigger model has been in the wait block for more than 100 ms

Using the front panel

The state of the trigger model is indicated on the status bar with the indicators shown in the previous table.

Using SCPI commands:

Send the command:

```
:TRIGger:STATe?
```

The return shows the state and the block that was last executed.

Using TSP commands:

Send the command:

```
print(trigger.model.state())
```

The return shows the state and the block that was last executed.

Using trigger events to start actions in the trigger model

You can set up trigger blocks to respond to trigger events. Trigger events are signals that can be generated by the instrument or by other system components.

Sources of the trigger event signals can be:

- Front-panel TRIGGER key
- Notify trigger blocks
- Branch-on-event trigger blocks
- Command-interface triggers
- Digital I/O lines
- TSP-Link synchronization lines
- LAN triggers
- Event blenders, which combine other trigger events
- Trigger timers

For information about the options that are not specific to the trigger model, see Triggering (on page 8-3).

Trigger events

To use trigger events, you need to specify the event constant. The tables below show the constants for the trigger events in the system.

Trigger events — SCPI command set

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	`BLENder<n>`
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	`DIGio<n>`
Front-panel TRIGGER key press	`DISPlay`
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	`LAN<n>`
No trigger event	`NONE`
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	`NOTify<n>`
Source limit condition occurs	`SLIMit`
Trigger timer <n> (1 to 4) expired	`TIMer<n>`
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	`TSPLink<n>`

Trigger events – TSP command set

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Trigger events	
Event description	**Event constant**
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`

Using the TRIGGER key to generate an event

You can use the front-panel TRIGGER key to generate a trigger event.

For example, if you set a wait block to advance when the TRIGGER key is pressed, the trigger model will reach the wait block. If the TRIGGER key has already been pressed, the trigger model execution will continue. If the TRIGGER key has not been pressed, the trigger model execution is halted until the TRIGGER key is pressed.

To set a trigger block to respond to the front-panel key press:

- **From the front panel:** Set the event to be Display TRIGGER Key
- **Using SCPI:** Set the event to `DISPlay`
- **Using TSP:** Set the event to `trigger.EVENT_DISPLAY`

There are no action overruns for front-panel TRIGGER key events.

Using the notify block event

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

When the trigger model executes a notify block, the instrument generates the SCPI event `NOTify<n>` or TSP event `trigger.EVENT_NOTIFYN`. You can assign this event to a command that takes an event.

For example, if you want a Notify block to trigger a digital I/O line, insert a Notify block into the trigger model, assign it a notify event and then connect it to the stimulus of the digital I/O line to drive. In the following example, you define trigger model block 5 to be the notify 2 event. You can then assign the notify 2 event to be the stimulus for digital output line 3. To do this, send the following commands in SCPI:

```
:TRIG:BLOC:NOT 5, 2
:TRIG:DIG3:OUT:STIMulus NOTify2
```

In TSP, send the commands:

```
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

Respond to an event with a wait block

The wait building block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to three events for each wait block. The wait block can use any of the system trigger events. See Trigger events (on page 8-53).

To continue the trigger model, it must receive the trigger event that is defined for the wait block.

Using the branch-on-event trigger blocks

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The branch-on-event block can use any of the system trigger events. See Trigger events (on page 8-53).

Section 9

TSP-Link and TSP-Net

In this section:

TSP-Link System Expansion Interface 9-1
TSP-Net 9-13

TSP-Link System Expansion Interface

Keithley Instruments TSP-Link® is a high-speed trigger synchronization and communication bus that test system builders can use to connect multiple instruments in a master and subordinate configuration. Once connected, all the instruments that are equipped with TSP-Link in a system can be programmed and operated under the control of the master instrument or instruments. This allows the instruments to run tests more quickly because they can be decoupled from frequent computer interaction. The test system can have multiple master and subordinate groups, which can be used to handle multi-device testing in parallel. Combining TSP-Link with a flexible programmable trigger model ensures speed.

Using TSP-Link, multiple instruments are connected and can be used as if they are part of the same physical unit for simultaneous multi-channel testing. The test system can be expanded to include up to 32 TSP-Link-enabled instruments.

TSP-Link functionality is only available when using the instrument front panel or the TSP commands to control the instrument. It is not available if you are using SCPI commands.

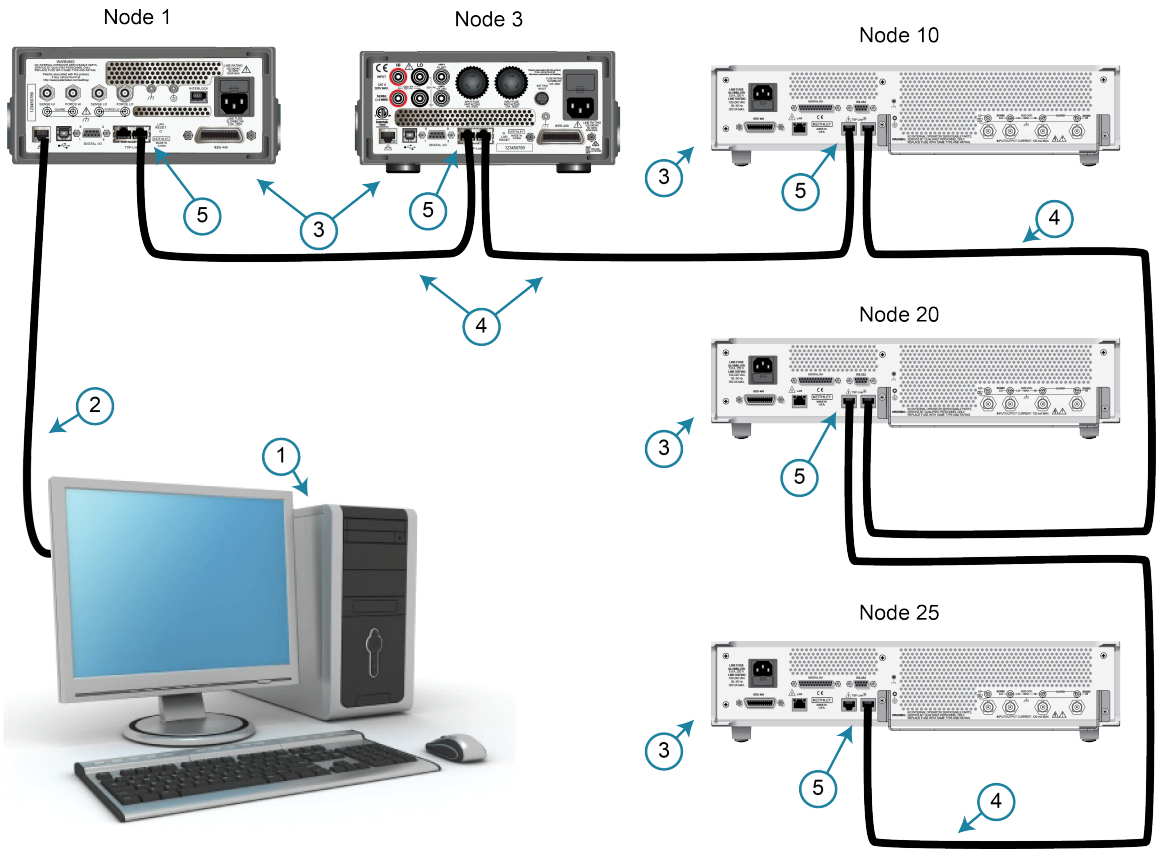
TSP-Link connections

The 2470 has three synchronization lines that are built into the TSP-Link connection. If you are using a TSP-Link network, you do not have to modify any connections.

Example connections for a TSP-Link system are shown in the following figure.

The TSP-Link connectors are on the rear panel of the instruments. All the instruments in the system are connected in a sequence (daisy-chained) using LAN crossover cables.

Figure 132: TSP-Link connections



Item	Description	Notes
1	Controller	Optional. A computer is not needed for stand-alone systems.
2	Communication connection	Optional. Connection from controller to the master node through GPIB, LAN, or USB. Details about these computer communication connections are described in Remote communications interfaces (on page 2-7).
3	Nodes	You can have up to 32 nodes on the TSP-Link system. Each node must have a unique node number from 1 to 63.
4	LAN crossover cable	Type 5e category or higher; 3 meters (9.8 feet) maximum between nodes.
5	TSP-Link connections	Each instrument has two TSP-Link connectors. You can make the connection to either TSP-Link connection.

TSP-Link nodes

Each instrument or enclosure attached to the TSP-Link expansion interface is called a node. Each node must be identified with a unique node number. This identification is called a TSP-Link node number.

An individual node is accessed as `node[N]`, where *N* is the node number assigned to the node. You can access all TSP-accessible remote commands as elements of the specific node. The following attributes are examples of items you can access:

- `node[N].model`: The product model number of the node.
- `node[N].version`: The product version of the node.
- `node[N].serialno`: The product serial number of the node.

Assigning node numbers

Each 2470 instrument is initially assigned as node 2. You can assign node numbers from 1 to 63. However, the system can only include 32 physical nodes.

The node number for each instrument is stored in its nonvolatile memory and remains in storage when the instrument is turned off.

You can assign a node number to an instrument using the front panel or by using a remote command.

To assign a node number using the front panel:

1. Press the **MENU** key.
2. Under System, select **Communication**. The SYSTEM COMMUNICATIONS window opens.
3. Select the **TSP-Link** tab.
4. Next to Node, set the TSP-Link address for this instrument.

To assign a node number using a remote command:

Set the `tsplink.node` attribute of the instrument:

```
tsplink.node = N
```

Where *N* = `1` to `63`

To determine the node number of an instrument, you can read the `tsplink.node` attribute by sending the following command:

```
print(tsplink.node)
```

The above `print` command outputs the node number. For example, if the node number is `1`, a `1` is displayed.

Master and subordinates

In a TSP-Link® system, one of the nodes (instruments) is the master node and the other nodes are the subordinate nodes. The master node in a TSP-Link® system can control the other nodes (subordinates) in the system.

A TSP-Link system can be stand-alone or computer-based.

In a stand-alone system, scripts are loaded into the instruments. You can run a script from the front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation. When the script is finished running, all the nodes in the system return to local operation, and the master/subordinate relationship between nodes is dissolved.

In a computer-based system, you can use a computer and a remote interface to communicate with a single node in the system. This node becomes the interface to the entire system. When a command is sent through this node, all nodes go into remote operation. The node that receives the command becomes the master and can control all other nodes, which become its subordinates. In a computer-based system, the master/subordinate relationship between nodes can only be dissolved by performing an abort operation. For more information about remote interfaces, see Remote communications interfaces (on page 2-7).

NOTE

When linking with earlier models of Keithley instruments such as the Model 2600B, make sure to use the 2470 as the master node and the earlier instruments as subordinates.

Initializing the TSP-Link system

The TSP-Link® system must be initialized after configuration changes. You need to initialize the system after you:

- Turn off power or reboot any instrument in the system
- Change node numbers on any instrument in the system
- Rearrange or disconnect the TSP-Link cable connections between instruments

If initialization is not successful, you can check the event log for error messages that indicate the problem. Some typical problems include:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument performing the initialization
- One or more of the instruments in the system is turned off
- The actual number of nodes is less than the expected number

From the front panel:

1. Power on all instruments connected to the TSP-Link network.
2. Press the **MENU** key.
3. Under System, select **Communication**. The SYSTEM COMMUNICATIONS window opens.
4. Select the **TSP-Link** tab.
5. Select **Initialize**.

Using TSP commands:

To initialize the TSP-Link system, send the command:

```
tsplink.initialize()
```

To check the state of the TSP-Link system, send the command:

```
print(tsplink.state)
```

If initialization was successful, `online` is returned. If initialization was not successful, `offline` is returned.

Sending commands to TSP-Link nodes

You can send remote commands to any instrument on the TSP-Link system by adding `node[`*N*`].` to the beginning of the remote command, where *N* is the node number.

For example, to sound the beeper on node 10, you would send the command:

```
node[10].beeper.beep(2, 2400)
```

To send a command to the master, you can interact with it as if it were a single instrument.

Using the reset() command

Most TSP-Link® system operations target a single node in the system, but the `reset()` command affects the system as a whole by resetting all nodes to their default settings:

```
-- Reset all nodes in a TSP-Link system to their default state.
reset()
```

NOTE

Using the `reset()` command in a TSP-Link network differs from using the tsplink.initialize() (on page 14-296) command. The `tsplink.initialize()` command reinitializes the TSP-Link network and turns off the output of any TSP-linked instrument. It may change the state of individual nodes in the system.

Use `node[N].reset()` or `localnode.reset()` to reset only one of the nodes. The other nodes are not affected. The following programming example shows this type of reset operation with code that is run on node 1.

```
-- Reset node 1 only.
node[1].reset()
-- Reset the node you are connected to (in this case, node 1).
localnode.reset()
-- Reset node 4 only.
node[4].reset()
```

Terminating scripts on the TSP-Link system

You can terminate a script that is executing on a TSP-Link system.

To terminate an executing script and return all nodes to local control, send the following command:

```
abort
```

This dissolves the master/subordinate relationships between nodes.

You can also abort an executing script and turn off the source outputs on all source-measure units in the TSP-Link system. To do this, press the OUTPUT ON/OFF switch on any source-measure instrument in the system.

From the front panel, you can abort a script by pressing the TRIGGER key for a few seconds and selecting **Abort Trigger Model** from the dialog box that is displayed.

Triggering using TSP-Link trigger lines

The 2470 has three trigger lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

Using TSP-Link trigger lines for digital I/O

Each trigger line is an open-drain signal. When using the TSP-Link® trigger lines for digital I/O, any node that sets the programmed line state to zero (0) causes all nodes to read 0 from the line state. This occurs regardless of the programmed line state of any other node. Refer to the table in the Digital I/O bit weighting (on page 8-23) topic for digital bit weight values.

Running simultaneous test scripts

Running test scripts simultaneously improves functional testing, provides higher throughput, and expands system flexibility. You can use TSP-Link and TSP scripting to run simultaneous test scripts. You can also manage the resources that are allocated to test scripts that are running simultaneously.

In addition, you can use the data queue to do real-time communication between nodes on the TSP-Link system.

To run test scripts simultaneously, you can set up your TSP-Link network in one of the following configurations:

- Multiple TSP-Link networks
- Single TSP-Link network with groups

Using groups to manage nodes on a TSP-Link system

TSP-Link groups allow each group to run a different test script simultaneously. This method requires one TSP-Link network and a single remote connection to the computer that is connected to the master node.

A group can consist of one or more nodes. You must assign group numbers to each node using remote commands. If you do not assign a node to a group, it defaults to group 0, which will always be grouped with the master node (regardless of the group to which the master node is assigned).

The following table shows an example of the functions of groups on a single TSP-Link network. Each group in this example runs a different test script than the other groups, which allows the system to run multiple tests simultaneously.

TSP-Link network group functions

Group number	Group members	Present function
0	Master node 1	Initiates and runs a test script on node 2 Initiates and runs a test script on node 6 In addition, the master node can execute scripts and process run commands
1	Group leader node 2	Runs the test script initiated by the master node Initiates remote operations on node 3 through node 5
	Node 3 through node 5	Performs remote operations initiated by node 2
2	Group leader node 6	Runs the test script initiated by the master node Initiates remote operations on node 7 through node *n*
	Node 7 through node *n*	Performs remote operations initiated by node 6

Master node overview

You can assign the master node to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to group 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

The master node is always the node that coordinates activity on the TSP-Link network.

The master node:

- Is the only node that can use the `execute()` command on a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation (a command that continues to operate after the command that initiated it has finished running)
- Can execute the `waitcomplete()` command to wait for the group to which the master node belongs; to wait for another group; or to wait for all nodes on the TSP-Link network to complete overlapped operations (overlapped commands allow the execution of subsequent commands while device operations of the overlapped command are still in progress)

Group leader overview

Each group has a dynamic group leader. The last node in a group that performs any operation initiated by the master node is the group leader.

The group leader:

- Performs operations initiated by the master node
- Initiates remote operations on any node with the same group number
- Cannot initiate remote operations on any node with a different group number
- Can use the `waitcomplete()` command without a parameter to wait for all overlapped operations running on nodes in the same group

Assigning groups

Group numbers can range from zero (0) to 64. The default group number is 0. You can change the group number at any time. You can also add or remove a node to or from a group at any time.

Each time the power for a node is turned off, the group number for that node changes to 0.

The following example code dynamically assigns a node to a group:

```
-- Assign node 3 to group 1.
node[3].tsplink.group = 1
```

Running test scripts and programs on remote nodes

You can send the `execute()` command from the master node to initiate a test script and Lua code on a remote node. The `execute()` command places the remote node in the overlapped operation state. As a test script runs on the remote node, the master node continues to process other commands simultaneously.

Use the following code to send the `execute()` command for a remote node. The *N* parameter represents the node number that runs the test script (replace *N* with the node number).

To set the global variable "setpoint" on node N to 2.5:

```
node[N].execute("setpoint = 2.5")
```

The following code demonstrates how to run a test script that is defined on the local node. For this example, *scriptVar* is defined on the local node, which is the node that initiates the code to run on the remote node. The local node must be the master node.

To run scriptVar on node N:

```
node[N].execute(scriptVar.source)
```

The programming example below demonstrates how to run a test script that is defined on a remote node. For this example, *scriptVar* is defined on the remote node.

To run a script defined on the remote node:

```
node[N].execute("scriptVar()")
```

It is recommended that you copy large scripts to a remote node to improve system performance.

Coordinating overlapped operations in remote groups

All overlapped operations on all nodes in a group must have completed before the master node can send a command to the group. If you send a command to a node in a remote group when an overlapped operation is running on any node in that group, errors will occur.

You can execute the `waitcomplete()` command on the master node or group leader to wait for overlapped operations. The action of `waitcomplete()` depends on the parameters specified.

If you want to wait for completion of overlapped operations for:

- **All nodes in the local group:** Use `waitcomplete()` without a parameter from the master node or group leader.
- **A specific group:** Use `waitcomplete(N)` with a group number as the parameter from the master node. This option is not available for group leaders.
- **All nodes in the system:** Use `waitcomplete(0)` from the master node. This option is not available for group leaders.

For additional information, refer to waitcomplete() (on page 14-319).

The following code shows two examples of using the `waitcomplete()` command from the master node:

```
-- Wait for each node in group N to complete all overlapped operations.
waitcomplete(N)
-- Wait for all groups on the TSP-Link network to complete overlapped operations.
waitcomplete(0)
```

A group leader can issue the `waitcomplete()` command to wait for the local group to complete all overlapped operations.

The following code is an example of how to use the `waitcomplete()` command from a group leader:

```
-- Wait for all nodes in the local group to complete all overlapped operations.
waitcomplete()
```

Using the data queue for real-time communication

Nodes that are running test scripts at the same time can store data in the data queue for real-time communication. Each instrument has an internal data queue that uses the first-in, first-out (FIFO) structure to store data. You can use the data queue to post numeric values, strings, and tables.

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link® network at any time

You cannot access the reading buffers or global variables from any node in a remote group while a node in that group is performing an overlapped operation. However, you can use the data queue to retrieve data from any node in a group that is performing an overlapped operation. In addition, the master node and the group leaders can use the data queue to coordinate activities.

Tables in the data queue consume one entry. When a node stores a table in the data queue, a copy of the data in the table is made. When the data is retrieved from the data queue, a new table is created on the node that is retrieving the data. The new table contains a separate copy of the data in the original table, with no references to the original table or any subtables.

You can access data from the data queue even if a remote group or a node has overlapped operations in process. See the `dataqueue` commands in the TSP command reference (on page 14-1) for more information.

Remote TSP-Link commands

Commands that control and access the TSP-Link® synchronization port are summarized in the following table. See the TSP command reference (on page 14-1) for complete details on these commands.

Use the commands in the following table to perform basic steady-state digital I/O operations; for example, you can program the 2470 to read and write to a specific TSP-Link synchronization line or to the entire port.

TSP-Link commands

Command	Description
trigger.tsplinkin[N].clear() (on page 14-289)	Clears the event detector for a trigger
trigger.tsplinkin[N].edge (on page 14-289)	Indicates which trigger edge controls the trigger event detector for a trigger line
trigger.tsplinkin[N].overrun (on page 14-290)	Indicates if the event detector ignored an event while in the detected state
trigger.tsplinkin[N].wait() (on page 14-291)	Waits for a trigger
trigger.tsplinkout[N].assert() (on page 14-291)	Simulates the occurrence of the trigger and generates the corresponding trigger event
trigger.tsplinkout[N].logic (on page 14-292)	Defines the trigger output with output logic for a trigger line
trigger.tsplinkout[N].pulsewidth (on page 14-293)	Sets the length of time that the trigger line is asserted for output triggers
trigger.tsplinkout[N].release() (on page 14-293)	Releases a latched trigger on the given TSP-Link trigger line
trigger.tsplinkout[N].stimulus (on page 14-294)	Specifies the event that causes the synchronization line to assert a trigger
tsplink.group (on page 14-296)	The group number of the TSP-Link node
tsplink.initialize() (on page 14-296)	Initializes all instruments and enclosures in the TSP-Link system
tsplink.line[N].mode (on page 14-298)	Defines the trigger operation of a TSP-Link line as digital in or out or trigger in or out
tsplink.line[N].reset() (on page 14-298)	Resets some of the TSP-Link trigger attributes to their defaults
tsplink.line[N].state (on page 14-299)	Reads or writes the digital state of a TSP-Link synchronization line
tsplink.master (on page 14-300)	Reads the node number assigned to the master node
tsplink.node (on page 14-301)	Defines the node number
tsplink.readport() (on page 14-301)	Reads the TSP-Link synchronization lines as a digital I/O port
tsplink.state (on page 14-302)	Describes the TSP-Link online state
tsplink.writeport() (on page 14-303)	Writes to all TSP-Link synchronization lines as a digital I/O port

TSP-Link synchronization programming example

The programming example below illustrates how to set bit B1 of the TSP-Link digital I/O port high, and then read the entire port value:

```
tsplink.line[1].mode = tsplink.MODE_DIGITAL_OPEN_DRAIN
-- Set bit B1 high.
tsplink.line[1].state = 1
-- Read I/O port.
data = tsplink.readport()
print(data)
```

The output would be similar to:

```
7
```

To read bit B1 only:

```
-- To read bit B1 only
data = tsplink.line[1].state
print(data)
```

The output would be similar to:

```
tsplink.STATE_HIGH
```

Using 2470 TSP-Link commands with other TSP-Link products

If you are connecting the 2470 in a system with other TSP-Link products, be aware that some of the TSP-Link commands may be different.

Commands that are the same in all TSP-Link products:

- `tsplink.group`
- `tsplink.master`
- `tsplink.node`
- `tsplink.readport()`
- `tsplink.state`
- `tsplink.writeport()`

2470 TSP-Link command	Replaces this command in other TSP-Link products
`trigger.tsplinkin[N].clear()`	`tsplink.trigger[N].clear()`

`trigger.tsplinkin[N].edge`
`trigger.tsplinkout[N].logic`
`tsplink.line[N].mode`
	`tsplink.trigger[N].mode`

`trigger.tsplinkin[N].overrun`	`tsplink.trigger[N].overrun`
`trigger.tsplinkin[N].wait()`	`tsplink.trigger[N].wait()`
`trigger.tsplinkout[N].assert()`	`tsplink.trigger[N].assert()`
`trigger.tsplinkout[N].pulsewidth`	`tsplink.trigger[N].pulsewidth`
`trigger.tsplinkout[N].release()`	`tsplink.trigger[N].release()`
`trigger.tsplinkout[N].stimulus`	`tsplink.trigger[N].stimulus`
`tsplink.initialize()`	`tsplink.reset()`
`tsplink.line[N].reset()`	`tsplink.trigger[N].reset()`
`tsplink.line[N].state`	`tsplink.readbit()` `tsplink.writebit()`
Not applicable	`tsplink.writeprotect`

TSP-Net

TSP-Net provides a simple socket-like programming interface to Test Script Processor (TSP) enabled instruments. Using the TSP-Net library, the 2470 can control ethernet-enabled devices directly through its LAN port. This enables the 2470 to communicate directly with a device that is that is not TSP-enabled without the use of a controlling computer.

Using TSP-Net library methods, you can transfer string data to and from a remote instrument, transfer and format data into Lua variables, and clear input buffers. The TSP-Net library is only accessible using commands from a remote command interface when you are using the TSP command language.

While you can use TSP-Net commands to communicate with any ethernet-enabled instrument, specific TSP-Net commands exist for TSP-enabled instruments to allow for support of features unique to the TSP scripting engine. These features include script downloads, reading buffer access, wait completion, and handling of TSP scripting engine prompts.

Using TSP-Net commands with TSP-enabled instruments, a 2470 can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The 2470 can read the data from the remote instrument and either manipulate the data or send the data to a different remote instrument on the LAN.

You can use TSP-Net to connect to a computer; you can use a script on the instrument to transfer data directly to your computer hard drive.

With TSP-Net, you can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the 2470.

Using TSP-Net with any ethernet-enabled instrument

NOTE

Refer to TSP command reference (on page 14-1) for details about the commands presented in this section.

The 2470 LAN port is auto-sensing (Auto-MDIX), so you can use either a LAN crossover cable or a LAN straight-through cable to connect directly from the 2470 to an ethernet device or to a hub.

To set up communication to a remote ethernet-enabled instrument that is TSP® enabled:

1. Send the following command to configure TSP-Net to send an abort command when a connection to a TSP instrument is established:

```
tspnet.tsp.abortonconnect = 1
```

   If the scripts are allowed to run, the connection is made, but the remote instrument may be busy.

2. Send the command:

```
connectionID = tspnet.connect(ipAddress)
```

   Where:

   - *connectionID* is the connection ID that will be used as a handle in all other TSP-Net function calls.
   - *ipAddress* is the IP address, entered as a string, of the remote instrument.

See tspnet.connect() (on page 14-304) for additional detail.

To set up communication to a remote ethernet-enabled device that is not TSP enabled:

Send the command:

```
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

Where:

- *connectionID* is the connection ID that will be used as a handle in all other `tspnet` function calls.
- *ipAddress* is the IP address, entered as a string, of the remote device.
- *portNumber* is the port number of the remote device.
- *initString* is the initialization string that is to be sent to *ipAddress*.

See tspnet.connect() (on page 14-304) for additional detail.

To communicate to a remote ethernet device from the 2470:

1. Connect to the remote device using one of the above procedures. If the 2470 cannot make a connection to the remote device, it generates a timeout event. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 s.
2. Use `tspnet.write()` or `tspnet.execute()` to send strings to a remote device. If you use:
   - `tspnet.write()`: Strings are sent to the device exactly as indicated, and you must supply any needed termination characters.
   - `tspnet.execute()`: The 2470 appends termination characters to all strings that are sent. Use `tspnet.termination()` to specify the termination character.
3. To retrieve responses from the remote instrument, use `tspnet.read()`. The 2470 suspends operation until the remote device responds or a timeout event is generated. To check if data is available from the remote instrument, use `tspnet.readavailable()`.
4. Disconnect from the remote device using the `tspnet.disconnect()` function. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote device that is not TSP® enabled, and send and receive data from this device:

```
-- Set tspnet timeout to 5 s.
tspnet.timeout = 5
--  Establish connection to another device with IP address 192.168.1.51
-- at port 1394.
id_instr = tspnet.connect("192.168.1.51", 1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set the termination character to CRLF. You must do this
-- for each connection after the connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device.
tspnet.write(id_instr, "login admin\r\n")
-- Read the data available, then print it.
tspnet.write(id_instr, "*idn?\r\n")
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

This example produces a return such as:

```
ID is:    1
instrument write/read returns:      SUCCESS: Logged in
instrument write/read returns:      KEITHLEY INSTRUMENTS,MODEL 2470,04089762,1.6.3d
```

Remote instrument events

If the 2470 is connected to a TSP-enabled instrument through TSP-Net, all events that occur on the remote instrument are transferred to the event log of the 2470. The 2470 indicates events from the remote instrument by prefacing these events with “Remote Error.” For example, if the remote instrument generates event code 4909, "Reading buffer not found within device," the 2470 generates the string “Remote Error: (4909) Reading buffer not found within device.”

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

tspnet.clear() (on page 14-303)
tspnet.connect() (on page 14-304)
tspnet.disconnect() (on page 14-305)
tspnet.execute() (on page 14-306)
tspnet.idn() (on page 14-307)
tspnet.read() (on page 14-308)
tspnet.readavailable() (on page 14-309)
tspnet.reset() (on page 14-309)
tspnet.termination() (on page 14-310)
tspnet.timeout (on page 14-311)
tspnet.write() (on page 14-314)

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

tspnet.tsp.abort() (on page 14-311)
tspnet.tsp.abortonconnect (on page 14-312)
tspnet.tsp.rbtablecopy() (on page 14-313)
tspnet.tsp.runscript() (on page 14-314)

Example: Using tspnet commands

```
function telnetConnect(ipAddress, userName, password)
   -- Connect through Telnet to a computer.
   id = tspnet.connect(ipAddress, 23, "")
   -- Read the title and login prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the login name.
   tspnet.write(id, userName .. "\r\n")
   -- Read the login echo and password prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the password information.
   tspnet.write(id, password .. "\r\n")
   -- Read the telnet banner from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
end

function test_tspnet()
   tspnet.reset()
   -- Connect to a computer using Telnet.
   telnetConnect("192.0.2.1", "my_username", "my_password")
   -- Read the prompt back from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   -- Change directory and read the prompt back from the computer.
   tspnet.write(id, "cd c:\\\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Make a directory and read the prompt back from the computer.
   tspnet.write(id, "mkdir TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Change to the newly created directory.
   tspnet.write(id, "cd c:\\TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- if you have data print it to the file.
   -- 11.2 is an example of data collected.
   cmd = "echo " .. string.format("%g", 11.2) .. " >> datafile.dat\r\n"
   tspnet.write(id, cmd)
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   tspnet.disconnect(id)
end
test_tspnet()
```

Section 10

Maintenance

In this section:

Introduction 10-1
Line fuse replacement 10-1
Lithium battery 10-2
Front-panel display 10-2
Upgrading the firmware 10-3

Introduction

The information in this section describes routine maintenance of the instrument that the operator can perform.

Line fuse replacement

A fuse on the 2470 rear panel protects the power line input of the instrument. Follow the below instructions to replace the fuse. You do not need to return your instrument for service if the fuse is damaged.

⚠ WARNING

Disconnect the line cord at the rear panel and remove all test leads connected to the instrument before replacing a line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.

Use only the correct fuse type. Failure to do so could result in injury, death, or instrument damage.

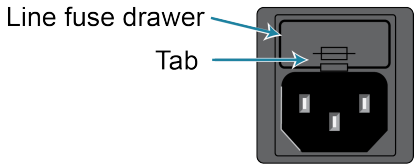
Use a 5 x 20 mm slow-blow fuse rated at 250 V at 2 A.

To replace the fuse, you will need a small, flat-bladed screwdriver.

Complete the following steps to replace the line fuse:

1. Power off the instrument.
2. Remove all test leads connected to the instrument.
3. Remove the line cord.
4. Locate the fuse drawer, which is above the AC receptacle, as shown in the figure below.

Figure 133: 2470 line fuse



5. Use the screwdriver to lift the tab from the fuse drawer.
6. Slide the fuse drawer out. The fuse drawer does not pull completely out of the power module.
7. Snap the fuse out of the drawer.
8. Replace the fuse.
9. Push the fuse drawer back into the module.

If a fuse continues to become damaged, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

Lithium battery

The 2470 contains a CR2032 cell ($LiMnO_2$) battery. Perchlorate material may require special handling. See Hazardous waste - perchlorate (dtsc.ca.gov/hazardouswaste/perchlorate).

This battery is not replaceable by the user.

Front-panel display

Do not use sharp metal objects, such as tweezers or screwdrivers, or pointed objects, such as pens or pencils, to touch the touchscreen. It is strongly recommended that you use only fingers to operate the instrument. Use of clean-room gloves to operate the touchscreen is supported.

Cleaning the front-panel display

If you need to clean the front-panel LCD touchscreen display, use a soft dry cloth.

CAUTION

Do not use liquids to clean the display.

Abnormal display operation

If the display area is pushed hard during operation, you may see abnormal display operation. To restore normal operation, turn the instrument off and then back on.

Removing ghost images or contrast irregularities

If the display has been operating for a long time with the same display patterns, the display patterns may remain on the screen as ghost images and a slight contrast irregularity may appear. Note that if this occurs, it does not adversely affect the performance reliability of the display.

To regain normal operation, stop using the front-panel display for some time. You can turn off the front-panel display while continuing operation using remote commands and the virtual front panel.

To turn off the front-panel display using a SCPI command:

Send the command:

```
DISPlay:LIGHt:STATe OFF
```

To turn off the front-panel display using a TSP command:

Send the command:

```
display.lightstate = display.STATE_LCD_OFF
```

Upgrading the firmware

To upgrade the 2470 firmware, you load an upgrade file into the instrument. You can load the file from the front-panel USB port using either a remote interface or the front panel of the instrument. If you are using Test Script Builder (TSB), you can upgrade the firmware from TSB using a file saved to the computer on which TSB is running.

During the upgrade process, the instrument verifies that the version you are loading is newer than what is on the instrument. If the version is older or at the same revision level, no changes are made.

If you want to return to a previous version or reload the present version of the firmware, select **Downgrade to Older**. This forces the instrument to load the firmware regardless of the version.

The upgrade process normally takes about five minutes.

Upgrade files are available on tek.com/keithley.

CAUTION

Disconnect the input and output terminals before you upgrade or downgrade.

Do not remove power from the 2470 or remove the USB flash drive while an upgrade or downgrade is in progress. Wait until the instrument completes the procedure and shows the opening display. If you are upgrading a 2470-NFP instrument, the LAN and 1588 LEDs on the front panel blink in unison during the upgrade and stop when the upgrade is complete.

Do not initialize or reset TSP-Link before starting the upgrade.

Before upgrading, turn the instrument power off, wait a few seconds, then turn the instrument power on.

From the front panel

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

NOTE

The firmware file must be in the root subdirectory of the flash drive and must be the only firmware file in that location. You can upgrade or downgrade the firmware from the front panel or from the virtual front panel. Refer to Using the 2470 virtual front panel (on page 2-30) for information.

From the front panel or virtual front panel:

1. Copy the firmware file (`.upg` file) to a USB flash drive.
2. Verify that the firmware file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect any input and output terminals that are attached to the instrument.
4. Turn the instrument power off. Wait a few seconds.
5. Turn the instrument power on.
6. Insert the flash drive into the USB port on the front panel of the instrument.
7. From the instrument front panel, press the **MENU** key.
8. Under System, select **Info/Manage**.
9. Choose an upgrade option:
   - To upgrade to a newer version of firmware, select **Upgrade to New**.
   - To return to a previous version of firmware, select **Downgrade to Older**.
10. If the instrument is controlled remotely, a message is displayed. Select **Yes** to continue.
11. When the upgrade is complete, reboot the instrument.

A message is displayed while the upgrade is in progress.

Using SCPI

There are no SCPI commands that you can use to upgrade the firmware. To upgrade the firmware, you must either use the front panel, virtual front panel, or switch the command set to TSP.

To use the front panel to upgrade the firmware, see From the front panel (on page 10-4).

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

If you need to upgrade the firmware from a remote interface and you are using a SCPI command set:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect the input and output terminals that are attached to the instrument.
4. Power on the instrument.
5. Change the command set to TSP by sending the command:
   `*LANG TSP`
6. Turn the instrument off and then turn it on again.
7. Insert the flash drive into the USB port on the front panel of the instrument.
8. To upgrade to a newer version of firmware, send:
   `upgrade.unit()`
9. To return to a previous version of firmware, send:
   `upgrade.previous()`
10. After completion of the upgrade, turn the instrument off and then turn it on again.
11. To return to the SCPI command set, send the command:
    `*LANG SCPI`
12. Turn the instrument off and then turn it on again.

A message is displayed on the front panel of the instrument while the upgrade is in process. In addition, the LEDs in the upper right of the front panel blink while the upgrade is in process.

Using TSP

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

Using TSP over a remote interface:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect the input and output terminals that are attached to the instrument.
4. Turn the instrument power off. Wait a few seconds.
5. Turn the instrument power on.
6. Insert the flash drive into the USB port on the front panel of the instrument.
7. To upgrade to a newer version of firmware, send:
   `upgrade.unit()`
8. To return to a previous version of firmware, send:
   `upgrade.previous()`
9. After completion of the upgrade, reboot the instrument.

A message is displayed on the front panel of the instrument while the upgrade is in progress. In addition, the LEDs in the upper right of the front panel blink while the upgrade is in process.

Using TSB

CAUTION

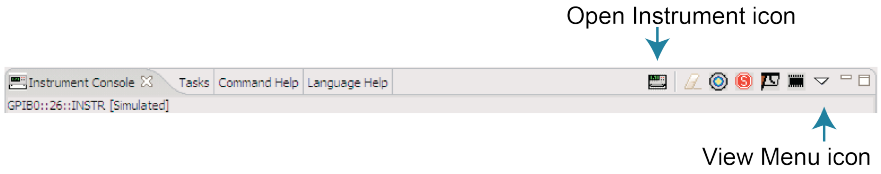
Do not turn off power or remove the USB flash drive until the upgrade process is complete.

You can use Test Script Builder (TSB) to upgrade the firmware of your instrument.

To upgrade the firmware using Test Script Builder:

1. Disconnect the input and output terminals that are attached to the instrument.
2. Turn the instrument power off. Wait a few seconds.
3. Turn the instrument power on.
4. Start Test Script Builder.
5. On the Instrument Console toolbar, select the **Open Instrument** icon.

Figure 134: TSB Instrument Console toolbar



6. Select your communication interface from the Select Instrument dialog box. See Connecting an instrument in TSB (on page 13-35) for details on opening communications.
7. On the Instrument Console toolbar, choose the View Menu icon. Select **Instrument**, then select **Flash**.
8. From the Select a Firmware Image File dialog box, use the browser to select the file name of the new firmware or enter the path and file name.
9. If you are upgrading the firmware, replace the existing firmware with a newer version of firmware.
10. If you are downgrading the firmware, replace the existing firmware with an older version of firmware or repair the same version.
11. Select **OK**. A Progress Information bar is displayed on the instrument during the update. In addition, the LEDs in the upper right of the front panel blink while the upgrade is in process.
12. Wait until the instrument indicates that the firmware upgrade is complete (TSB may indicate that the upgrade is complete before it is finalized on the instrument).
13. Reboot the instrument.

Section 11

Introduction to SCPI commands

In this section:

Introduction to SCPI 11-1
SCPI command programming notes 11-3
Acquiring readings using SCPI commands 11-8

Introduction to SCPI

The Standard Commands for Programmable Instruments (SCPI) standard is a syntax and set of commands that is used to control test and measurement devices.

The following information describes some basic SCPI command information and how SCPI is used with the 2470 and presented in the 2470 documentation.

This section also contains general information about using SCPI.

Command execution rules

Command execution rules are as follows:

- Commands execute in the order that they are presented in the command message.
- An invalid command generates an event message and is not executed.
- Valid commands that precede an invalid command in a command message are executed.
- Valid commands that follow an invalid command in a command message are ignored.

Command messages

A command message is made up of one or more command words sent by the controller to the instrument.

SCPI commands contain several command words that are structured to create command messages. The command words are separated by colons (`:`). For example, to configure an ethernet connection, the command words are:

```
:SYSTem:COMMunication:LAN:CONFigure
```

Many commands have query options. If there is a query option, it is created by adding a question mark (?) to the command. For example, to query the present ethernet settings, send:

```
:SYSTem:COMMunication:LAN:CONFigure?
```

Commands often take parameters. Parameters follow the command words and a space. For example, to set the instrument to automatically detect the ethernet settings, send:

```
:SYSTem:COMMunication:LAN:CONFigure "AUTO"
```

SCPI can also use common commands, which consist of an asterisk (*) followed by three or four letters. For example, you can reset the instrument by sending the following command:

```
*RST
```

The examples above show commands that are sent individually. You can also group command messages when you send them to the instrument. To group a set of commands, separate them with semicolons and include a colon before each command (unless it starts with an *).

For example, to reset the instrument, enable relative offset for the current function, and set a relative offset of 0.5 for the current function, send the command:

```
*RST; :SENSe:CURRent:REL:STAT ON; :SENSe:CURRent:RELative 0.5
```

If commands are not combined, the colon (:) at the beginning of a command is optional. For example, the following commands are equivalent:

```
:SENSe:CURRent:REL:STAT ON
SENSe:CURRent:REL:STAT ON
```

If the next command in a multiple command message is on the same path, you do not need to send the colon or the path to reset the path parsing of the command. For example, the following examples for returning the system time and system version are equivalent:

```
:SYST:TIME?;:SYST:VERSION?
:SYST:TIME?;VERSION?
```

Both return:

```
1569191196;1996.0
```

You can also do multiple queries in a single command message with or without resetting the path. For example, to query for the current relative offset and state, you can send:

```
:SENSe:CURRent:RELative?; :SENSe:CURRent:REL:STAT?
```

You can also send:

```
SENSe:CURRent:RELative?; rel:STAT?
```

Each new command message resets the parser path as if it was sent with the leading colon. The output for both queries is:

```
0.5;1
```

A command string sent to the instrument must terminate with a <new line> character. The IEEE-488.2 EOI (end-or-identify) message is interpreted as a <new line> character and can be used to terminate a command string in place of a <new line> character. A <carriage return> followed by a <new line> is also accepted. Command string termination will always reset the current SCPI command path to the root level.

SCPI command programming notes

This section contains general information about using Standard Commands for Programmable Instruments (SCPI).

SCPI command formatting

This section describes the formatting that this manual uses when discussing SCPI commands.

SCPI command short and long forms

This documentation shows SCPI commands with both uppercase and lowercase letters. The uppercase letters are the required elements of a command. The lowercase letters are optional. However, if you choose to include the letters that are shown in lowercase letters, you must include all of them.

When you send a command to the instrument, letter case is not important — you can mix uppercase and lowercase letters in program messages.

For example, you can send the command `SENSe:COUNt?` in any of the following formats:

```
SENSe:COUNt?
sense:count?
SENS:COUN?
Sens:Coun?
```

Optional command words

If a command word is enclosed in brackets (`[ ]`), the command word is optional. Do not include the brackets if you send the optional command word to the instrument.

For example, you can send the command `:SYSTem:BEEPer[:IMMediate] <n1>, <n2>` in any of the following formats:

```
:SYSTem:BEEPer:IMMediate 500, 1
:SYSTem:BEEPer 500, 1
:SYST:BEEP:IMMediate 500, 1
:SYST:BEEP 500, 1
```

MINimum, MAXimum, and DEFault

You can use `MINimum`, `MAXimum`, or `DEFault` instead of a parameter for some commands.

For example, you can set the parameter for the command `[:SENSe[1]]:RESistance:NPLCycles` to the minimum, maximum, or default value. To set NPLC to the minimum value, you can send either of these commands:

```
:SENSe1:RESistance:NPLCycles MINimum
:SENS:RES:NPLC MIN
```

Queries

SCPI queries have a question mark (?) after the command. You can use the query to determine the present value of the parameters of the command or to get information from the instrument.

For example, to determine what the present setting for NPLC is, you can send:

```
:SENSe1:RESistance:NPLCycles?
```

This query returns the present setting.

If the command has `MINimum`, `MAXimum`, and `DEFault` options, you can use the query command to determine what the minimum, maximum, and default values are. In these queries, the `?` is placed before the `MINimum`, `MAXimum`, or `DEFault` parameter. For example, to determine the default value for NPLC, you can send:

```
:SENSe1:RESistance:NPLCycles? DEFault
```

If you send two query commands without reading the response from the first, and then attempt to read the second response, you may receive some data from the first response followed by the complete second response. To avoid this, do not send a query command without reading the response. When you cannot avoid this situation, send a device clear before sending the second query command.

When you query a Boolean option, the instrument returns a `0` or `1`, even if you sent `OFF` or `ON` when you originally sent the command.

SCPI parameters

The parameters of the SCPI commands are shown in angle brackets (< >). For example:

```
:SYSTem:BEEPer[:IMMediate] <frequency>, <duration>
```

The type of information that you can use to replace `<frequency>` and `<duration>` is defined in the Usage section of the command description. For this example, the Usage is:

`<frequency>`	The frequency of the beep (20 Hz to 8000 Hz)
`<duration>`	The amount of time to play the tone (0.001 s to 100 s)

For this example, you can generate an audible sound by sending:

```
:SYSTem:BEEPer 500, 1
```

Note that you do not include the angle brackets when sending the command.

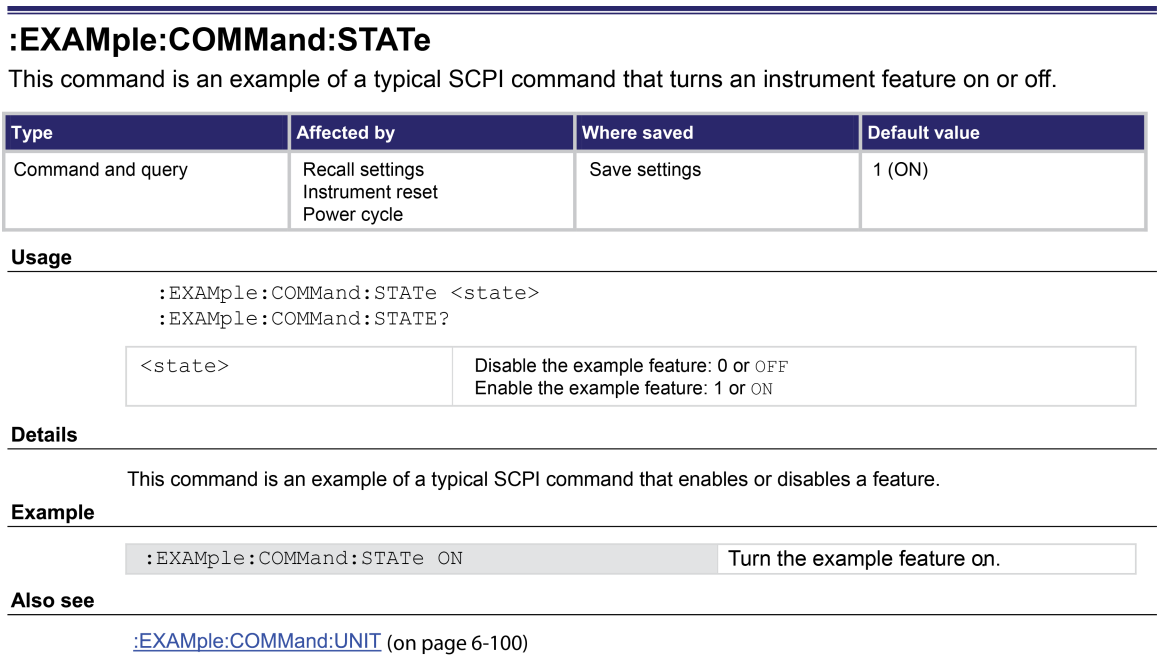
Sending strings

If you are sending a string, it must begin and end with matching quotes (either single quotes or double quotes). If you want to include a quote character as part of the string, type it twice with no characters in between.

Using the SCPI command reference

The SCPI command reference contains detailed descriptions of each of the SCPI commands that you can use to control your instrument. Each command description is broken into several standard subsections. The figure below shows an example of a command description.

Figure 135: SCPI command description example



:EXAMple:COMMand:STATe

This command is an example of a typical SCPI command that turns an instrument feature on or off.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1 (ON)

Usage

```
:EXAMple:COMMand:STATe <state>
:EXAMple:COMMand:STATE?
```

`<state>`	Disable the example feature: 0 or `OFF` Enable the example feature: 1 or `ON`

Details

This command is an example of a typical SCPI command that enables or disables a feature.

Example

`:EXAMple:COMMand:STATe ON`	Turn the example feature on.

Also see

:EXAMple:COMMand:UNIT (on page 6-100)

Each command listing is divided into five subsections that contain information about the command:

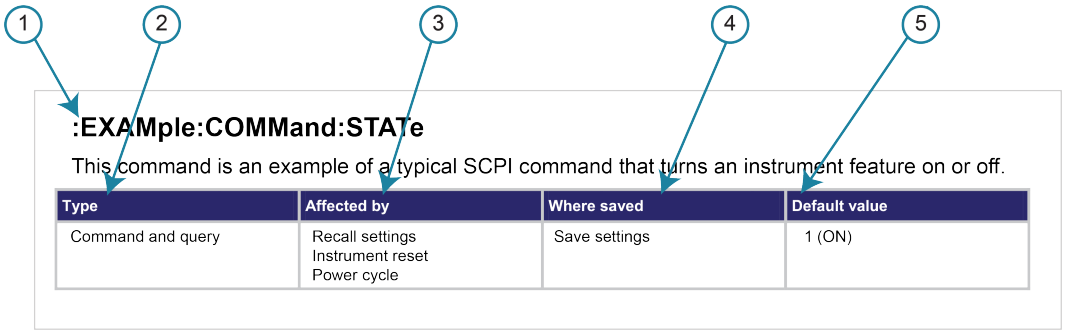
- Command name and summary table
- Usage
- Details
- Example
- Also see

The content of each of these subsections is described in the following topics.

Command name and summary table

Each instrument command description starts with the command name, followed by a table with relevant information for each command. Definitions for the numbered items below are listed following the figure.

Figure 136: SCPI command name and summary table



1 **Instrument command name**. Signals the beginning of the command description and is followed by a brief description of what the command does.

2 **Type of command**. Options are:
- **Command only**. There is a command but no query option for this command.
- **Command and query**. The command has both a command and query form.
- **Query only**. This command is a query.

3 **Affected by**. Commands or actions that have a direct effect on the instrument command.
- **Recall settings.** If you send `*RCL` to recall the system settings, this setting is changed to the saved value.
- **Instrument reset.** When you reset the instrument, this command is reset to its default value. Reset can be done from the front panel or when you send `*RST`.
- **Power cycle.** When you power cycle the instrument, this command is reset to its default value.
- **Source configuration list**. If you recall a source configuration list, this setting changes to the stored setting.
- **Measure configuration list.** If you recall a measure configuration list, this setting changes to the stored setting.

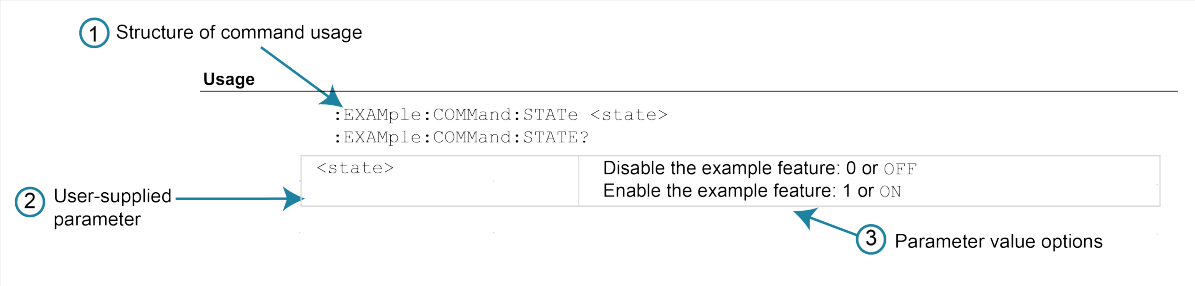
4 **Where saved**. Indicates where the command settings reside once they are used on an instrument. Options include:
- **Not saved.** Command is not saved and must be sent each time you use it.
- **Nonvolatile memory.** The command is stored in a storage area in the instrument where information is saved even when the instrument is turned off.
- **Save settings.** This command is saved when you send the `*SAV` command.
- **Source configuration list.** This command is stored in source configuration lists.
- **Measure configuration list.** This command is stored in measure configuration lists.

5 **Default value:** Lists the default value for the command. The parameter values are defined in the Usage or Details sections of the command description.

Command usage

The Usage section of the remote command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage; all possible command usage options are shown here.

Figure 137: SCPI command description usage identification



1. **Structure of command usage:** Shows how the parts of the command should be organized.
2. **User-supplied parameters:** Indicated by angle brackets (< >).

NOTE

Some commands have optional parameters. Optional parameters are presented on separate lines in the Usage section, presented in the required order with each valid permutation of optional parameters. For example:

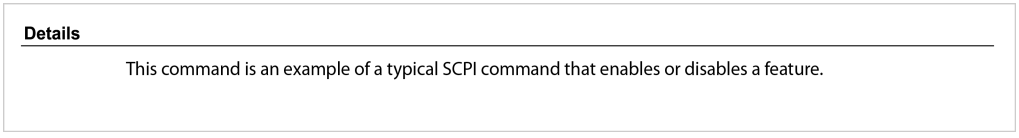
```
:SYSTem:COMMunication:LAN:CONFigure AUTO
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress, NETmask
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress, NETmask, GATeway
:SYSTem:COMMunication:LAN:CONFigure?
```

3. **Parameter value options:** Descriptions of the options that are available for the parameter.

Command details

This section lists additional information you need to know to successfully use the command.

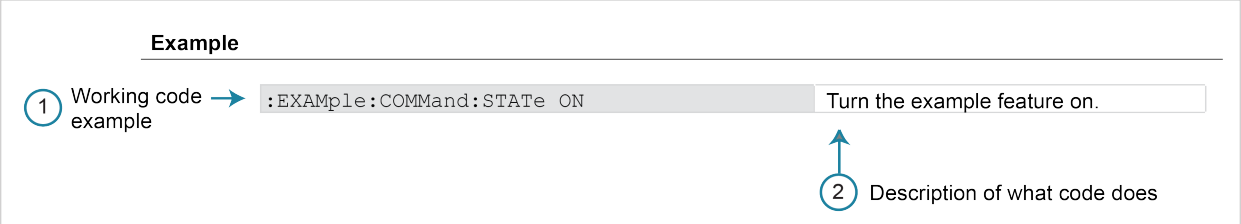
Figure 138: Details section of command listing



Example section

The Example section of the command description shows some simple examples of how the command can be used.

Figure 139: SCPI command description code examples



1. Example code that you can copy from this table and paste into your own application. Examples are generally shown using the short forms of the commands.
2. Description of the code and what it does. This may also contain the output of the code.

Related commands list

The **Also see** section of the remote command description provides links to commands that are related to the command that is being described.

Figure 140: SCPI related commands list example



Acquiring readings using SCPI commands

The following table summarizes SCPI commands that acquire and return readings.

Command	Description
FETCh?	Returns the specified data elements from the most recent reading. This command does not trigger source-measure operations. This command can repeatedly return the same readings. Until there are new readings, this command continues to return the old readings. See :FETCh? (on page 12-1) for more information about the FETCh? command.
READ?	Makes measurements, places them in a reading buffer, and returns the specified data elements from the latest reading. The READ? query returns the same information as the TRACe:TRIGger and FETCh? commands. Do not use INITiate with the READ? command. For example, send the following command to obtain the source voltage, measured current, and relative timestamp: READ? 1, 10, "defbuffer1", SOUR, READ, REL See :READ? (on page 12-6) for more information about the READ? command.
MEASure?	Same as READ?. See :MEASure? (on page 12-3) for more information.
TRACe:DATA?	Returns specified data elements from a specified reading buffer. Use this command if SENSe:COUNT > 1. Use TRACe:TRIGger to start making measurements if you are not using the trigger model. For example, send the following command to get the source voltage, measured current, and relative timestamp: TRAC:DATA? 1, 10, "defbuffer1", SOUR, READ, REL See :TRACe:DATA? (on page 12-121) for information about the TRACe:DATA? command.

Section 12

SCPI command reference

In this section:

:FETCh? 12-1
:MEASure? 12-3
:READ? 12-6
*RCL 12-8
*SAV 12-9
CALCulate subsystem 12-10
DIGital subsystem 12-24
DISPlay subsystem 12-28
FORMat subsystem 12-34
OUTPut subsystem 12-37
ROUTe subsystem 12-41
SCRipt subsystem 12-42
SENSe1 subsystem 12-43
SOURce subsystem 12-68
STATus subsystem 12-97
SYSTem subsystem 12-103
TRACe subsystem 12-117
TRIGger subsystem 12-145

:FETCh?

This command requests the latest reading from a reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:FETCh?
:FETCh? "<bufferName>"
:FETCh? "<bufferName>", <bufferElements>
```

`<bufferName>`	The name of the buffer where the reading is stored; if nothing is specified, `defbuffer1` is used
`<bufferElements>`	See **Details**; default is `READing`

Details

This command requests the last available reading from a reading buffer. If you send this command more than once and there are no new readings, the returned values are the same. If the reading buffer is empty, an error is returned.

NOTE

To change the number of digits returned in a remote command reading, use the `:FORMat:ASCii:PRECision` command.

You can send `:FETCh?` while a trigger model is running.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include any or all of the buffer elements listed below in a single list. You can repeat elements if the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
`DATE`	The date when the data point was measured
`FORMatted`	The measured value as it appears on the front panel
`FRACtional`	The fractional seconds for the data point when the data point was measured
`READing`	The measurement reading based on the `[:SENSe[1]]:FUNCtion[:ON]` setting; if no buffer elements are defined, this option is used
`RELative`	The relative time when the data point was measured
`SEConds`	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
`SOURce`	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 12-85))
`SOURFORMatted`	The source value as it appears on the display
`SOURSTATus`	The status information associated with sourcing. The values returned indicate the status of the following conditions: ▪ Overvoltage protection was active ▪ Measured source value was read ▪ Overtemperature condition existed ▪ Source function level was limited ▪ Four-wire sense was used ▪ Output was on
`SOURUNIT`	The unit of value associated with the source value
`STATus`	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
`TIME`	The time for the data point
`TSTamp`	The timestamp for the data point
`UNIT`	The unit of measure associated with the measurement

The output of `:FETCh?` is affected by the data format selected by `:FORMat[:DATA]`. If you set `FORMat[:DATA]` to `REAL` or `SREAL`, you will have fewer options for buffer elements. The only buffer elements available are `READing`, `RELative`, `SOURce`, and `EXTRa`. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The STATus buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the 2470, this will always be 0 (main)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

`FETCh? "defbuffer1", DATE, READ`	Retrieve the date and measurement value for the most recent data captured in `defbuffer1`. Example output: `03/21/2019,-1.375422E-11`

Also see

:FORMat[:DATA] (on page 12-36)
:INITiate[:IMMediate] (on page 12-145)
:MEASure? (on page 12-3)
:READ? (on page 12-6)
:TRACe:DATA? (on page 12-121)
:TRACe:TRIGger (on page 12-138)

:MEASure?

This command makes measurements, places them in a reading buffer, and returns the last reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:MEASure?
:MEASure:<function>?
:MEASure:<function>? "<bufferName>"
:MEASure:<function>? "<bufferName>", <bufferElements>
:MEASure? "<bufferName>"
:MEASure? "<bufferName>", <bufferElements>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<bufferName>`	The name of the buffer where the reading is stored; if nothing is specified, `defbuffer1` is used
`<bufferElements>`	See **Details**

Details

This command makes a measurement using the specified function and stores the reading in a reading buffer.

If you do not define the function parameter, the instrument uses the presently selected measure function.

This query makes the number of readings specified by `[:SENSe[1]]:COUNt`. When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

To get multiple readings, use the `:TRACe:DATA?` command.

Sending this command changes the measurement function to the one specified by `<function>`. This function remains selected after the measurement is complete.

`:MEASure?` performs the same function as `READ?`.

`:MEASure:<function>?` performs the same function as sending `:SENSe:FUNCtion`, then `READ?`.

NOTE

To change the number of digits returned in a remote command reading, use the `:FORMat:ASCii:PRECision` command.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include any or all of the buffer elements listed below in a single list. You can repeat elements if the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
`DATE`	The date when the data point was measured
`FORMatted`	The measured value as it appears on the front panel
`FRACtional`	The fractional seconds for the data point when the data point was measured
`READing`	The measurement reading based on the `[:SENSe[1]]:FUNCtion[:ON]` setting; if no buffer elements are defined, this option is used
`RELative`	The relative time when the data point was measured
`SEConds`	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
`SOURce`	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 12-85))
`SOURFORMatted`	The source value as it appears on the display

Option	Description
SOURSTATus	The status information associated with sourcing. The values returned indicate the status of the following conditions: ■ Overvoltage protection was active ■ Measured source value was read ■ Overtemperature condition existed ■ Source function level was limited ■ Four-wire sense was used ■ Output was on
SOURUNIT	The unit of value associated with the source value
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time for the data point
TSTamp	The timestamp for the data point
UNIT	The unit of measure associated with the measurement

The output of :MEASure? is affected by the data format selected by :FORMat[:DATA]. If you set FORMat[:DATA] to REAL or SREAL, you will have fewer options for buffer elements. The only buffer elements available are READing, RELative, SOURce, and EXTRa. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The STATus buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the 2470, this will always be 0 (main)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
TRACe:MAKE "voltMeasBuffer", 10000
MEAS:VOLT? "voltMeasBuffer", FORM, DATE, READ
```

Create a buffer named voltMeasBuffer. Make a voltage measurement and store it in the buffer voltMeasBuffer and return the formatted reading, the date, and the reading elements from the buffer.

Example output:

```
-00.0024 mV,05/16/2018,-2.384862E-06
```

Also see

:FORMat[:DATA] (on page 12-36)
:READ? (on page 12-6)
[:SENSe[1]]:FUNCtion[:ON] (on page 12-68)
:TRACe:DATA? (on page 12-121)

:READ?

This command makes measurements, places them in a reading buffer, and returns the last reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:READ?
:READ? "<bufferName>"
:READ? "<bufferName>", <bufferElements>
```

`<bufferName>`	The name of the buffer where the reading is stored; if nothing is specified, `defbuffer1` is used
`<bufferElements>`	See **Details**; if nothing is specified, `READing` is used

Details

This query makes the number of readings specified by `[:SENSe[1]]:COUNt`. If multiple readings are made, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command. To get multiple readings, use the `:TRACe:DATA?` command.

NOTE

To change the number of digits returned in a remote command reading, use the `:FORMat:ASCii:PRECision` command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include any or all of the buffer elements listed below in a single list. You can repeat elements if the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
`DATE`	The date when the data point was measured
`FORMatted`	The measured value as it appears on the front panel
`FRACtional`	The fractional seconds for the data point when the data point was measured
`READing`	The measurement reading based on the `[:SENSe[1]]:FUNCtion[:ON]` setting; if no buffer elements are defined, this option is used
`RELative`	The relative time when the data point was measured
`SEConds`	The seconds in UTC (Coordinated Universal Time) format when the data point was measured

Option	Description
SOURce	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 12-85))
SOURFORMatted	The source value as it appears on the display
SOURSTATus	The status information associated with sourcing. The values returned indicate the status of the following conditions: ■ Overvoltage protection was active ■ Measured source value was read ■ Overtemperature condition existed ■ Source function level was limited ■ Four-wire sense was used ■ Output was on
SOURUNIT	The unit of value associated with the source value
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time for the data point
TSTamp	The timestamp for the data point
UNIT	The unit of measure associated with the measurement

The output of :READ? is affected by the data format selected by :FORMat[:DATA]. If you set FORMat[:DATA] to REAL or SREAL, you will have fewer options for buffer elements. The only buffer elements available are READing, RELative, SOURce, and EXTRa. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The STATus buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the 2470, this will always be 0 (main)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
:TRACe:MAKE "voltMeasBuffer", 10000
:SENSe:FUNCtion "VOLTage"
:COUN 10
:READ? "voltMeasBuffer", FORM, DATE, READ
:TRAC:DATA? 1, 10, "voltMeasBuffer"
```

Create a buffer named `voltMeasBuffer`.
Set the measurement function to voltage.
Set the count to 10.
Make the measurements and store them in the buffer `voltMeasBuffer`. Return the last reading as displayed on the front panel with the date, along with the unformatted reading.
Return all 10 readings from the reading buffer.
Example output is:

```
-000.06580 mV,10/14/2018,-6.580474E-05
-1.322940E-05,-7.876178E-05,-7.798489E-05,-7.201674E-05,-9.442933E-05,-7.653603E-0
    6,-7.916663E-05,-8.177242E-05,-6.187183E-05,-6.580474E-05
```

Also see

:FETCh? (on page 12-1)
[:SENSe[1]]:COUNt (on page 12-67)
:TRACe:DATA? (on page 12-121)
:TRACe:TRIGger (on page 12-138)

*RCL

This command returns the instrument to the setup that was saved with the *SAV command.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*RCL <n>
```

`<n>`	An integer from `0` to `4` that represents the saved setup

Details

Restores the state of the instrument from a copy of user-saved settings that are stored in setup memory. The settings are saved using the *SAV command.

If you view the user-saved settings from the front panel of the instrument, these are stored as scripts named `Setup0<n>`.

Example

`*RCL 3`	Restores the settings stored in memory location 3.

Also see

Saving setups (on page 3-45)
*SAV (on page 12-9)

*SAV

This command saves the present instrument settings as a user-saved setup.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Nonvolatile memory	Not applicable

Usage

```
*SAV <n>
```

<n>	An integer from 0 to 4

Details

Save the present instrument settings as a user-saved setup. You can restore the settings with the *RCL command.

Most commands that are affected by *RST can be saved with the *SAV command.

You can save up to five user-saved setups. Any settings that had been stored previously as <n> are overwritten.

If you view the user-saved setups from the front panel of the instrument, they are stored as scripts named Setup0<n>.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press HOME and ENTER to save an image of the settings with the screen capture feature. Refer to Save screen captures to a USB flash drive (on page 3-44) for additional information.

Example

*SAV 2	Saves the instrument settings in memory location 2.

Also see

Saving setups (on page 3-45)
*RCL (on page 12-8)

CALCulate subsystem

The commands in this subsystem configure and control the math and limit operations.

:CALCulate[1]:<function>:MATH:FORMat

This command specifies which math operation is performed on measurements when math operations are enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PERC

Usage

```
:CALCulate[1]:<function>:MATH:FORMat <operation>
:CALCulate[1]:<function>:MATH:FORMat?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<operation>`	The name of the math operation: ■ y = mx+b: `MXB` ■ Percent: `PERCent` ■ Reciprocal: `RECiprocal`

Details

This specifies which math operation is performed on measurements for the selected measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Displays measurements as the percentage of deviation from a specified reference constant.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is `1/X`, where `X` is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

NOTE

If you send this command without the `<function>` parameter, it will set the state of the math format for all measure functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STAT ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 4-50)
:CALCulate[1]:<function>:MATH:MBFactor (on page 12-12)
:CALCulate[1]:<function>:MATH:MMFactor (on page 12-13)
:CALCulate[1]:<function>:MATH:PERCent (on page 12-15)
:CALCulate[1]:<function>:MATH:STATe (on page 12-16)

:CALCulate[1]:<function>:MATH:MBFactor

This command specifies the offset, b, for the y = mx + b operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
:CALCulate[1]:<function>:MATH:MBFactor <n>
:CALCulate[1]:<function>:MATH:MBFactor <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:MBFactor?
:CALCulate[1]:<function>:MATH:MBFactor? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<n>`	The offset for the y = mx + b operation; the valid range is –1e12 to +1e12
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This attribute specifies the offset (b) for an mx + b operation.

The mx + b math operation lets you manipulate normal display readings (x) mathematically based on the calculation:

$$y = mx + b$$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

NOTE

If you send this command without the `<function>` parameter, it will set the scale factor for all measure functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STAT ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 4-50)
:CALCulate[1]:<function>:MATH:FORMat (on page 12-10)
:CALCulate[1]:<function>:MATH:MMFactor (on page 12-13)
:CALCulate[1]:<function>:MATH:STATe (on page 12-16)

:CALCulate[1]:<function>:MATH:MMFactor

This command specifies the scale factor, m, for the y = mx + b math operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
:CALCulate[1]:<function>:MATH:MMFactor <value>
:CALCulate[1]:<function>:MATH:MMFactor <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:MMFactor?
:CALCulate[1]:<function>:MATH:MMFactor? <MIN|MAX|DEF>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<value>`	The scale factor; the valid range is –1e12 to +1e12
`<MIN|MAX|DEF>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the scale factor (m) for an mx + b operation for the selected measurement function.

The mx + b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$$y = mx + b$$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

NOTE

If you send this command without the <function> parameter, it will set the scale factor for all measure functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STAT ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 4-50)
:CALCulate[1]:<function>:MATH:FORMat (on page 12-10)
:CALCulate[1]:<function>:MATH:MBFactor (on page 12-12)
:CALCulate[1]:<function>:MATH:STATe (on page 12-16)

:CALCulate[1]:<function>:MATH:PERCent

This command specifies the reference constant that is used when math operations are set to percent.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
:CALCulate[1]:<function>:MATH:PERCent <value>
:CALCulate[1]:<function>:MATH:PERCent <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:PERCent?
:CALCulate[1]:<function>:MATH:PERCent? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<value>`	The reference used when the math operation is set to percent; the range is –1e12 to +1e12
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This is the constant that is used when the math operation is set to percent.

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

$$\text{Percent} = \left(\frac{\text{input - reference}}{\text{reference}}\right) \times 100\%$$

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

NOTE

If you send this command without the `<function>` parameter, it will set the constant for all measure functions.

Example

```
CALC:VOLT:MATH:FORM PERC
CALC:VOLT:MATH:PERC 50
CALC:VOLT:MATH:STAT ON
```

Set the math operations for voltage to percent.
Set the percentage value to 50.
Enable math operations.

Also see

Calculations that you can apply to measurements (on page 4-50)
:CALCulate[1]:<function>:MATH:FORMat (on page 12-10)
:CALCulate[1]:<function>:MATH:STATe (on page 12-16)

:CALCulate[1]:<function>:MATH:STATe

This command enables or disables math operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
:CALCulate[1]:<function>:MATH:STATe <n>
:CALCulate[1]:<function>:MATH:STATe?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<n>`	Disable math operations: `OFF` or `0` Enable math operations: `ON` or `1`

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

NOTE

If you send this command without the `<function>` parameter, it sets the state of math operations for all measure functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STAT ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

:CALCulate[1]:<function>:MATH:FORMat (on page 12-10)
Calculations that you can apply to measurements (on page 4-50)

:CALCulate2:<function>:LIMit<Y>:AUDible

This command determines if the instrument beeper sounds when a limit test passes or fails.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	NONE

Usage

```
:CALCulate2:<function>:LIMit<Y>:AUDible <state>
:CALCulate2:<function>:LIMit<Y>:AUDible?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`
`<state>`	When the beeper sounds: ■ Never: `NONE` ■ On test failure: `FAIL` ■ On test pass: `PASS`

Details

The tone and length of beeper cannot be adjusted.

NOTE

If you send this command without the `<function>` parameter, it will set the limit for all measure functions.

Example

`:CALC2:VOLT:LIM1:CLE:AUTO OFF`	Set limit autoclear off.
`:CALC2:VOLT:LIM1:AUD FAIL`	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
`:CALC2:VOLT:LIM1:LOW 0.25`	Set lower limit 1 for voltage to 0.25 V.
`:CALC2:VOLT:LIM1:UPP 2.5`	Set upper limit 1 for voltage to 2.5 V.
`:CALC2:VOLT:LIMIT1:STAT ON`	Enable limit 1 testing for voltage.
`:READ?`	Make a reading; the limit is checked and results display on the front panel.
`:CALC2:VOLT:LIMIT1:FAIL?`	Return the test results; example output if the test fails on the low limit: `LOW`
`:CALC2:VOLT:LIM1:CLE`	Clear the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:STATe (on page 12-22)

:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO

This command indicates if the test result for limit *Y* should be cleared automatically or not.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO <state>
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`
`<state>`	The auto clear setting: ■ Disable: `OFF` or `0` ■ Enable: `ON` or `1`

Details

When auto clear is set to on, limit conditions are cleared automatically after each measurement. If you are making a series of measurements, the instrument shows the limit test result of the last measurement for the pass or fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the auto clear setting to off. When this is set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The auto clear setting affects both the high and low limits.

NOTE

If you send this command without the `<function>` parameter, it will set autoclear for all measure functions.

Example

`:CALC2:VOLT:LIM1:CLE:AUTO ON`	Set limit autoclear on.
`:CALC2:VOLT:LIM1:AUD FAIL`	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
`:CALC2:VOLT:LIM1:LOW 0.25`	Set lower limit 1 for voltage to 0.25 V.
`:CALC2:VOLT:LIM1:UPP 2.5`	Set upper limit 1 for voltage to 2.5 V.
`:CALC2:VOLT:LIMIT1:STAT ON`	Enable limit 1 testing for voltage.
`:READ?`	Make a reading; the limit is checked and results display on the front panel.
`:CALC2:VOLT:LIMIT1:FAIL?`	Return the test results; example output if the test fails on the low limit: `LOW` The test results are automatically cleared.

Also see

:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate] (on page 12-19)

:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate]

This command clears the results of the limit test defined by *Y*.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate]
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`

Details

Use this command to clear the test results of limit *Y* when the limit auto clear option is turned off. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, turn the auto clear option on.

NOTE

If you send this command without the `<function>` parameter, it clears the limit for all measure functions.

Example

`:CALC2:VOLT:LIM1:CLE:AUTO OFF`	Set limit autoclear off.
`:CALC2:VOLT:LIM1:AUD FAIL`	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
`:CALC2:VOLT:LIM1:LOW 0.25`	Set lower limit 1 for voltage to 0.25 V.
`:CALC2:VOLT:LIM1:UPP 2.5`	Set upper limit 1 for voltage to 2.5 V.
`:CALC2:VOLT:LIMIT1:STAT ON`	Enable limit 1 testing for voltage.
`:READ?`	Make a reading; the limit is checked and results display on the front panel.
`:CALC2:VOLT:LIMIT1:FAIL?`	Return the test results; example output if the test fails on the low limit: `LOW`
`:CALC2:VOLT:LIM1:CLE`	Clear the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 12-18)
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 12-21)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 12-23)

:CALCulate2:<function>:LIMit<Y>:FAIL?

This command queries the results of a limit test.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:CALCulate2:<function>:LIMit<Y>:FAIL?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`

Details

This command queries the result of a limit test for the selected measurement function.

The response message indicates if the limit test passed or how it failed (on the high or low limit).

If autoclear is set to off, reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command. To automatically clear the results, set auto clear on.

If auto clear is set to on and you are making a series of measurements, the last measurement limit determines the fail indication for the limit. If auto clear is turned off, the results return a test fail if any of one of the readings failed.

To use this attribute, you must set the limit state to on.

The results of the limit test for limit *Y*:

- `NONE`: Test passed; the measurement is between the upper and lower limits
- `HIGH`: Test failed; the measurement exceeded the upper limit
- `LOW`: Test failed; the measurement exceeded the lower limit
- `BOTH`: Test failed; the measurement exceeded both limits

Example

`:CALC2:VOLT:LIM1:CLE:AUTO OFF`	Set limit autoclear off.
`:CALC2:VOLT:LIM1:AUD FAIL`	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
`:CALC2:VOLT:LIM1:LOW 0.25`	Set lower limit 1 for voltage to 0.25 V.
`:CALC2:VOLT:LIM1:UPP 2.5`	Set upper limit 1 for voltage to 2.5 V.
`:CALC2:VOLT:LIMIT1:STAT ON`	Enable limit 1 testing for voltage.
`:READ?`	Make a reading; the limit is checked and results display on the front panel.
`:CALC2:VOLT:LIMIT1:FAIL?`	Return the test results; example output if the test fails on the low limit: `LOW`
`:CALC2:VOLT:LIM1:CLE`	Clear the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 12-18)
:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate] (on page 12-19)
:CALCulate2:<function>:LIMit<Y>:STATe (on page 12-22)
Limit testing and binning (on page 4-66)

:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]

This command specifies the lower limit for limit tests.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	-1

Usage

```
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <n>
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <DEF|MIN|MAX>
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`
`<n>`	The low limit value of limit *Y* (-9.99999E+11 to 9.99999E+11)
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the lower limit for the limit *Y* test for the selected measure function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

NOTE

If you send this command without the `<function>` parameter, it will set the limit for all measure functions.

Default is 0.3 for limit 1 when the diode function is selected. The default for limit 2 for the diode function is –1.

Example

`:CALC2:VOLT:LIM1:CLE:AUTO OFF`
`:CALC2:VOLT:LIM1:AUD FAIL`
`:CALC2:VOLT:LIM1:LOW 0.25`
`:CALC2:VOLT:LIM1:UPP 2.5`
`:CALC2:VOLT:LIMIT1:STAT ON`
`:READ?`
`:CALC2:VOLT:LIMIT1:FAIL?`
`:CALC2:VOLT:LIM1:CLE`
	Set limit autoclear off.
	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
	Set lower limit 1 for voltage to 0.25 V.
	Set upper limit 1 for voltage to 2.5 V.
	Enable limit 1 testing for voltage.
	Make a reading; the limit is checked and results display on the front panel.
	Return the test results; example output if the test fails on the low limit:
	`LOW`
	Clear the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 12-23)

:CALCulate2:<function>:LIMit<Y>:STATe

This command enables or disables a limit test on the measurement from the selected measure function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
:CALCulate2:<function>:LIMit<Y>:STATe <state>
:CALCulate2:<function>:LIMit<Y>:STATe?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`
`<state>`	Disable the limit test: `OFF` or `0` Enable the limit test: `ON` or `1`

Details

This command enables or disables a limit test for the selected measurement function. When this attribute is enabled, the limit *Y* testing occurs on each measurement made by the instrument. Limit *Y* testing compares the measurements to the high-limit and low-limit values. If a measurement falls outside these limits, the test fails.

NOTE

If you send this command without the `<function>` parameter, it sets the state of math operations for all math functions.

Example

`:CALC2:VOLT:LIM1:CLE:AUTO OFF`	Set limit autoclear off.
`:CALC2:VOLT:LIM1:AUD FAIL`	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
`:CALC2:VOLT:LIM1:LOW 0.25`	Set lower limit 1 for voltage to 0.25 V.
`:CALC2:VOLT:LIM1:UPP 2.5`	Set upper limit 1 for voltage to 2.5 V.
`:CALC2:VOLT:LIMIT1:STAT ON`	Enable limit 1 testing for voltage.
`:READ?`	Make a reading; the limit is checked and results display on the front panel.
`:CALC2:VOLT:LIMIT1:FAIL?`	Return the test results; example output if the test fails on the low limit: `LOW`
`:CALC2:VOLT:LIM1:CLE`	Clear the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 12-18)
:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate] (on page 12-19)
:CALCulate2:<function>:LIMit<Y>:FAIL? (on page 12-20)
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 12-21)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 12-23)

:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]

This command specifies the upper limit for a limit test.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <value>
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <DEF|MIN|MAX>
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]?
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<Y>`	Limit number: `1` or `2`
`<value>`	The value of the upper limit (–9.99999e+11 to +9.99999e+11) or `DEFault`, `MINimum`, or `MAXimum`
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the high limit for the limit *Y* test for the selected measurement function. When limit *Y* testing is enabled, the instrument generates a fail indication when the measurement value is more than this value.

NOTE

If you send this command without the `<function>` parameter, it will set the limit for all measure functions.

Example

`:CALC2:VOLT:LIM1:CLE:AUTO OFF`	Set limit autoclear off.
`:CALC2:VOLT:LIM1:AUD FAIL`	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
`:CALC2:VOLT:LIM1:LOW 0.25`	Set lower limit 1 for voltage to 0.25 V.
`:CALC2:VOLT:LIM1:UPP 2.5`	Set upper limit 1 for voltage to 2.5 V.
`:CALC2:VOLT:LIMIT1:STAT ON`	Enable limit 1 testing for voltage.
`:READ?`	Make a reading; the limit is checked and results display on the front panel.
`:CALC2:VOLT:LIMIT1:FAIL?`	Return the test results; example output if the test fails on the low limit: `LOW`
`:CALC2:VOLT:LIM1:CLE`	Clear the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 12-21)
:CALCulate2:<function>:LIMit<Y>:STATe (on page 12-22)

DIGital subsystem

The commands in the DIGital subsystem control the digital I/O lines.

:DIGital:LINE<n>:MODE

This command sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	DIG, IN

Usage

```
:DIGital:LINE<n>:MODE <lineType>, <lineDirection>
:DIGital:LINE<n>:MODE?
```

`<n>`	The digital I/O line: 1 to 6
`<lineType>`	Sets the digital line control type; the options are: ■ Allow direct digital control of the line: `DIGital` ■ Configure for trigger control: `TRIGger` ■ Configure as a synchronous master or acceptor: `SYNChronous`

<lineDirection>	Sets the line direction; the options are: ■ Input: IN ■ Output: OUT ■ Open drain: OPENdrain ■ Master: MASTer ■ Acceptor: ACCeptor See **Details** for valid combinations with line type.

Details

You can specify the line type and line direction parameters to configure each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

Set <lineDirection> to one of the values shown in the following table.

Value	Description
IN	If the type is digital control, this automatically detects externally generated logic levels. You can read an input line, but you cannot write to it. If the type is trigger control, the line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This mode uses the edge setting specified by :TRIGger:DIGital<n>:IN:EDGE.
OUT	If the type is digital control, you can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low. If the type is trigger control, it is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
OPENdrain	Configures the line to be an open-drain signal. This makes the line compatible with other instruments that use open-drain digital I/O lines or trigger signals, such as other Keithley Instruments products. If the type is digital control, the line can serve as an input, an output, or both. You can read from the line or write to it. When a digital I/O line is used as an input in open-drain mode, you must write a 1 to it. If the type is trigger control, you can use the line to detect input triggers or generate output triggers. This mode uses the edge setting specified by :TRIGger:DIGital<n>:IN:EDGE.
ACCeptor	Only available with the SYNChronous trigger type. This value detects a falling-edge trigger as an input trigger and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line.
MASTer	Only available with the SYNChronous trigger type. This value detects a rising-edge trigger as an input. It asserts a TTL-low pulse for output.

Example

`:DIG:LINE1:MODE DIG, OUT`	Set digital I/O line 1 as a digital output line.

Also see

Digital I/O lines (on page 8-16)
Digital I/O port configuration (on page 8-14)
:TRIGger:DIGital<n>:IN:EDGE (on page 12-175)

:DIGital:LINE<n>:STATe

This command sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	See **Details**

Usage

```
:DIGital:LINE<n>:STATe <state>
:DIGital:LINE<n>:STATe?
```

`<n>`	The digital I/O line: 1 to 6
`<state>`	Clear the bit (bit low): `0` Set the bit (bit high): `1`

Details

When the line mode for a digital I/O line is set to digital output (`:DIG:LINE<n>:MODE DIG, OUT`), you can set the line high or low using the `<state>` parameter. When the line mode is set to digital input (`:DIG:LINE<n>:MODE DIG, IN`), you can query the state of the digital input line.

When a reset occurs, the digital line state can be read as high because the digital line is reset to a digital input. A digital input floats high if nothing is connected to the digital line.

This returns the integer equivalent values of the binary states on all six digital I/O lines.

Set the state to zero (0) to clear the bit; set the state to one (1) to set the bit.

Example 1

`:DIG:LINE1:MODE DIG, OUT` `:DIG:LINE1:STAT 1`	Set digital I/O line 1 as a digital output line. Sets line 1 (bit B1) of the digital I/O port high.

Example 2

`:DIG:LINE1:MODE DIG, IN` `:DIG:LINE1:STAT?`	Set digital I/O line 1 as a digital input line. Query the state of line 1 on the digital I/O port. Output: `1`

Also see

Digital I/O port configuration (on page 8-14)
:DIGital:LINE<n>:MODE (on page 12-24)
:DIGital:READ? (on page 12-27)
:DIGital:WRITe <n> (on page 12-27)
:TRIGger:DIGital<n>:IN:EDGE (on page 12-175)

:DIGital:READ?

This command reads the digital I/O port.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:DIGital:READ?
```

Details

The binary equivalent of the returned value indicates the value of the input lines on the digital I/O port. The least significant bit (bit B1) of the binary number corresponds to digital I/O line 1; bit B6 corresponds to digital I/O line 6.

For example, a returned value of 42 has a binary equivalent of 101010, which indicates that lines 2, 4, 6 are high (1), and the other lines are low (0).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

`:DIG:READ?`	Assume lines 2, 4, and 6 are set high when the I/O port is read. Output: 42 This is binary 101010

Also see

Digital I/O bit weighting (on page 8-23)
Digital I/O port configuration (on page 8-14)

:DIGital:WRITe <n>

This command writes to all digital I/O lines.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:DIGital:WRITe <n>
```

`<n>`	The value to write to the port (`0` to `63`)

Details

This function writes to the digital I/O port by setting the binary state of each digital line from an integer equivalent value.

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a *data* value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other three lines are set low).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

`:DIG:WRIT 63`	Sets digital I/O lines 1 through 6 high (binary 111111).

Also see

Digital I/O bit weighting (on page 8-23)
Digital I/O port configuration (on page 8-14)

DISPlay subsystem

This subsystem contains commands that control the front-panel display.

:DISPlay:BUFFer:ACTive

This command determines which buffer is used for measurements that are displayed on the front panel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1

Usage

```
:DISPlay:BUFFer:ACTive "<bufferName>"
:DISPlay:BUFFer:ACTive?
```

`<bufferName>`	The name of the buffer to make active

Details

The buffer defined by this command is used to store measurements data and is shown in the reading buffer indicator on the home screen of the instrument.

Example

`:DISP:BUFF:ACT "buffer2"`	Set the front panel to use `buffer2` as the active reading buffer.

Also see

None

:DISPlay:CLEar

This command clears the text from the front-panel USER swipe screen.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:DISPlay:CLEar
```

Example

`DISP:CLE`
`DISP:SCR SWIPE_USER`
`DISP:USER1:TEXT "Batch A122"`
`DISP:USER2:TEXT "Test running"`
	Clear the USER swipe screen.
	Display the USER swipe screen.
	Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

:DISPlay:USER<n>:TEXT[:DATA] (on page 12-33)

:DISPlay:<function>:DIGits

This command determines the number of digits that are displayed for measurements on the front panel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Measure: 5

Usage

```
:DISPlay:<function>:DIGits <value>
:DISPlay:<function>:DIGits <DEF|MIN|MAX>
:DISPlay:<function>:DIGits?
:DISPlay:<function>:DIGits? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<value>`	Display digits: ■ 6.5: 6 ■ 5.5: 5 ■ 4.5: 4 ■ 3.5: 3
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

The change in digits occurs the next time a measurement is made.

To change the number of digits returned in a remote command reading, use `:FORMat:ASCii:PRECision`.

NOTE

If you send this command without the `<function>` parameter, the digits values for all measure functions are changed.

Example

`:DISP:CURR:DIG 5`	Set the front panel to display current measurements with 5½ digits.

Also see

:FORMat:ASCii:PRECision (on page 12-34)

:DISPlay:LIGHt:STATe

This command sets the light output level of the front-panel display.

Type	Affected by	Where saved	Default value
Command and query	Power cycle	Not applicable	ON50

Usage

```
:DISPlay:LIGHt:STATe <brightness>
:DISPlay:LIGHt:STATe?
```

`<brightness>`	The brightness of the display: ■ Full brightness: `ON100` ■ 75% brightness: `ON75` ■ 50% brightness: `ON50` ■ 25% brightness: `ON25` ■ Display off: `OFF` ■ Display and all indicators off: `BLACkout`

Details

This command changes the light output of the front panel when a test requires different instrument illumination levels.

The change in illumination is temporary. The normal backlight settings are restored after a power cycle. You can use this to reset a display that is already dimmed by the front-panel Backlight Dimmer.

NOTE

Screen life is affected by how long the screen is on at full brightness. The higher the brightness setting and the longer the screen is bright, the shorter the screen life.

Example

`DISP:LIGH:STAT ON50`	Set the display brightness to 50%.

Also see

Adjust the backlight brightness and dimmer (on page 3-7)

:DISPlay:READing:FORMat

This command determines the format that is used to display measurement readings on the front-panel display of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	PREF

Usage

```
:DISPlay:READing:FORMat <format>
:DISPlay:READing:FORMat?
```

`<format>`	Use exponent format: `EXPonent` Add a prefix to the units symbol, such as k, m, or µ: `PREFix`

Details

This setting persists through `*RST` and power cycles.

When Prefix is selected, prefixes are added to the units symbol, such as k (kilo) or m (milli). When Exponent is selected, exponents are used instead of prefixes. When the prefix option is selected, very large or very small numbers may be displayed with exponents.

Example

`DISP:READ:FORM EXP`	Change front-panel display to show readings in exponential format.

Also see

Setting the display format (on page 3-41)

:DISPlay:SCReen

This command changes which front-panel screen is displayed.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:DISPlay:SCReen <screenName>
```

`<screenName>`	The screen to display: ■ Home screen: `HOME` ■ Home screen with large readings: `HOME_LARGe_reading` ■ Reading table: `READing_table` ■ Graph screen (opens last selected tab): `GRAPh` ■ Histogram screen: `HISTogram` ■ GRAPH swipe screen: `SWIPE_GRAPh` ■ SETTINGS swipe screen: `SWIPE_SETTings` ■ SOURCE swipe screen: `SOURce` ■ STATISTICS swipe screen: `SWIPE_STATistics` ■ USER swipe screen: `SWIPE_USER` ■ Open a screen that uses minimal CPU resources: `PROCessing`

Example

```
DISP:CLE
DISP:USER1:TEXT "Batch A122"
DISP:USER2:TEXT "Test running"
DISP:SCR SWIPE_USER
```

Clear the USER swipe screen.
Set the first line of the USER swipe screen to read "Batch A122" and the second line to display "Test running".
Display the USER swipe screen.



Also see

None

:DISPlay:USER<n>:TEXT[:DATA]

This command defines the text that is displayed on the front-panel USER swipe screen.

Type	Affected by	Where saved	Default value
Command only	Power cycle Instrument reset	Not applicable	Not applicable

Usage

```
:DISPlay:USER<n>:TEXT[:DATA] "<textMessage>"
```

`<n>`	The line of the USER swipe screen on which to display text: ■ Top line: `1` ■ Bottom line: `2`
`<textMessage>`	String that contains the message; up to 20 characters for `USER1` and 32 characters for `USER2`

Details

This command defines text messages for the USER swipe screen.

If you enter too many characters, the instrument displays a warning event and shortens the message to fit.

When the instrument is reset, the user test is removed and the USER swipe screen is hidden until another message is defined.

Example

`DISP:CLE`
`DISP:SCR SWIPE_USER`
`DISP:USER1:TEXT "Batch A122"`
`DISP:USER2:TEXT "Test running"`
	Clear the USER swipe screen
	Display the USER swipe screen.
	Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

:DISPlay:SCReen (on page 12-32)

FORMat subsystem

The commands for this subsystem select the data format that is used to transfer instrument readings over the remote interface.

:FORMat:ASCii:PRECision

This command sets the precision (number of digits) for all numbers returned in the ASCII format.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:FORMat:ASCii:PRECision <value>
:FORMat:ASCii:PRECision <DEF|MIN|MAX>
:FORMat:ASCii:PRECision?
:FORMat:ASCii:PRECision? <DEF|MIN|MAX>
```

`<value>`	The precision: ■ Automatic: `0` ■ Specific value: `1` to `16`
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This attribute specifies the precision (number of digits) for queries.

Note that the precision is the number of significant digits. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

`:FORM:ASC:PREC 10`	Set a precision of 10 digits. An example of the output is: `-6.999999881E-01`

Also see

:FORMat[:DATA] (on page 12-36)

:FORMat:BORDer

This command sets the byte order for the IEEE Std 754 binary formats.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	SWAP

Usage

```
:FORMat:BORDer <name>
:FORMat:BORDer?
```

<name>	The binary byte order: ■ Normal byte order: NORMal ■ Reverse byte order for binary formats: SWAPped

Details

This attribute selects the byte order in which data is written.

The SWAPped byte order must be used when transmitting binary data to a computer with a Microsoft Windows operating system.

The ASCII data format can only be sent in the normal byte order. If the ASCII format is selected, the SWAPped selection is ignored.

When you select NORMal byte order, the data format for each element is sent as follows:

```
Byte 1 Byte 2 Byte 3 Byte 4
```

(Single precision)

When you select SWAPped, the data format for each element is sent as follows:

```
Byte 4 Byte 3 Byte 2 Byte 1
```

(Single precision)

The #0 header is not affected by this command. The header is always sent at the beginning of the data string for each measurement conversion.

Example

FORM:BORD NORM	Use the normal byte order.

Also see

:FORMat[:DATA] (on page 12-36)

:FORMat[:DATA]

This command selects the data format that is used when transferring readings over the remote interface.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	ASC

Usage

```
:FORMat[:DATA] <type>
:FORMat[:DATA]?
```

<type>	The data format, which can be one of the following: ■ ASCII format: ASCii ■ IEEE Std. 754 double-precision format: REAL ■ IEEE Std. 754 single-precision format: SREal

Details

This command affects the output of READ?, FETCh?, MEASure:<function>?, and TRACe:DATA? queries over a remote interface. All other queries are returned in the ASCII format.

NOTE

The 2470 only responds to input commands using the ASCII format, regardless of the data format that is selected for output strings.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with #0 and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

If you set this to REAL or SREAL, you have fewer options for buffer elements with the TRACe:DATA?, READ?, MEASURE:<function>?, and FETCh? commands. The only buffer elements available are READing, RELative, SOURce, and EXTRa. If you request a buffer element that is not available, you see the event code 1133, "Parameter 4, Syntax error, expected valid name parameter."

Example

FORM REAL	Set the format to double-precision format.

Also see

:TRACe:DATA? (on page 12-121)

OUTPut subsystem

The output subsystem provides information and settings that control the output of the selected source.

:OUTPut[1]:<function>:SMODe

This command defines the state of the source when the output is turned off.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	NORM

Usage

```
:OUTPut[1]:<function>:SMODe <state>
:OUTPut[1]:<function>:SMODe?
```

`<function>`	The function to which this setting applies: ■ Current: `CURRent[:DC]` ■ Voltage: `VOLTage[:DC]`
`<state>`	The output-off setting; set to one of the following values (see the **Details** below for specifics regarding each of option): ■ `NORMal` ■ `HIMPedance` ■ `ZERO` ■ `GUARd`

Details

This command sets the state of the output when the source is off for the selected function.

When the 2470 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10% of the full scale of the present measurement function autorange value
- If source readback is off, Output Off is displayed in the home screen Source area
- If source readback is on, the actual measurement is displayed in the home screen Source area
- If measurement is set to resistance, dashes (--.----) are shown in the home screen Source area
- The Source button on the home screen shows the value that will be sourced when the output is turned on again

When the high-impedance output-off state is selected and the output is turned off:

- The measurement sense is set to 2-wire
- The output relay opens, disconnecting the instrument as a load

Opening the relay disconnects external circuitry from the inputs and outputs of the instrument. To prevent excessive wear on the output relay, do not use this output-off state for tests that turn the output off and on frequently.

The high-impedance output-off state should be used when the instrument is connected to a power source or another source-measure instrument. In some cases, it may also be appropriate for devices such as capacitors.

When the zero output-off state is selected and you turn off the output:

- The measurement sense is changed to 2-wire
- The voltage source is selected and set to 0 V
- The range is set to the presently selected range (turn off autorange)
- If the source is voltage, the current limit is not changed
- If the source is current, the current limit is set to the programmed source current value or to 10% full scale of the present current range, whichever is greater

When the zero output-off state is selected, you can use the instrument as an ammeter because it is outputting 0 V.

When the guard output-off state is selected and the output is turned off, the following actions occur:

- The measurement sense is changed to 2-wire
- The current source is selected and set to 0 A if the source is set to current (amps); otherwise, the output remains a voltage source when the output is turned off
- The voltage limit is set to 10% full scale of the present voltage range

Note that the front-panel display does not reflect all of the changes. For example, the 4-wire display indicator continues to display when the output is off, even though the sense is changed to 2-wire.

NOTE

If you send this command without the `<function>` parameter, it sets the output-off state for all functions.

Example

`:OUTP:CURR:SMOD HIMP`	Sets the output-off state for the current function so that the instrument opens the output relay when the output is turned off.

Also see

Output-off state (on page 4-16)
:OUTPut[1][:STATe] (on page 12-41)

:OUTPut[1]:INTerlock:STATe

This command determines if the output can be turned on when the interlock is not engaged.

Type	Affected by	Where saved	Default value
Command and query	None	Nonvolatile memory	OFF

Usage

```
:OUTPut[1]:INTerlock:STATe <state>
:OUTPut[1]:INTerlock:STATe?
```

`<state>`	Allow the output to be turned on when the interlock is not engaged: `OFF` or `0` Only allow the output to be turned on if the interlock is engaged: `ON` or `1`

Details

The instrument provides an interlock circuit on the rear panel. You must enable this circuit in order for the instrument to set source voltages greater than ±42 V DC.

When the safety interlock signal is asserted, the following actions occur:

- All voltage ranges of the instrument are available.
- The green front-panel INTERLOCK indicator is on.

The action when the interlock signal is not asserted depends on the Interlock setting. If Interlock is set to Off, if the safety interlock signal is not asserted, the following occurs:

- The nominal output is limited to less than ±42 V.
- The front-panel INTERLOCK indicator is not illuminated.
- You can output voltages less than ±42.

If Interlock is set to On, when the safety interlock signal is not asserted, the following occurs:

- You cannot turn on the source output.
- The front-panel INTERLOCK indicator is not illuminated.
- Whenever the interlock changes state (from asserted to not asserted or vice versa), the output is turned off.

If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, event code 5074, “Output voltage limited by interlock,” is generated.

⚠ WARNING

The 2470 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

Example

`:OUTP:INT:STAT ON`	Only allow the Source Output to be turned on when the interlock is asserted.

Also see

:OUTPut[1]:INTerlock:TRIPped? (on page 12-40)

:OUTPut[1]:INTerlock:TRIPped?

This command indicates that the interlock has been tripped.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:OUTPut[1]:INTerlock:TRIPped?
```

Details

This command gives you the status of the interlock. When the safety interlock signal is asserted, all voltage ranges of the instrument are available. However, when the safety interlock signal is not asserted, the 200 V and 1000 V ranges are hardware limited to a nominal output of less than ±42 V.

When the interlock is not asserted:

- The front-panel INTERLOCK indicator is off.
- High voltage ranges are disabled.
- If you attempt to turn on the source with a voltage more than ±21 V, an event message is generated.

If `1` is returned, the interlock signal is asserted and all voltage ranges are available.

If `0` is returned, the interlock is not asserted and the 200 V and 1000 V ranges are limited. Lower voltage ranges are available.

Example

`OUTP:INT:TRIP?`	If the interlock is not asserted, returns `0`. If the interlock is asserted, returns `1`.

Also see

None

:OUTPut[1][:STATe]

This command enables or disables the source output.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0 (OFF)

Usage

```
:OUTPut[1][:STATe] <state>
:OUTPut[1][:STATe]?
```

`<state>`	Turn source off: `0` or `OFF` Turn source on: `1` or `ON`

Details

When the output is switched on, the instrument sources either voltage or current, as set by `:SOURce[1]:FUNCtion[:MODE]`.

Example

```
:OUTP ON
```

Switch the source output of the instrument to on.

Also see

:SOURce[1]:FUNCtion[:MODE] (on page 12-80)

ROUTe subsystem

The ROUTe subsystem selects which set of input and output terminals to enable (front panel or rear panel).

:ROUTe:TERMinals

This command describes which set of input and output terminals the instrument is using.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	FRON

Usage

```
:ROUTe:TERMinals <location>
:ROUTe:TERMinals?
```

`<location>`	Use the front-panel input and output terminals: `FRONt` Use the rear-panel input and output terminals: `REAR`

Details

This command selects which set of input and output terminals the instrument uses. You can select front panel or rear panel terminals.

If the output is on when you change from one set of terminals to the other, the output is turned off.

Example

`:ROUT:TERM REAR` `:ROUT:TERM?`	Set the instrument to use the rear-panel terminals and query to verify. Output: REAR

Also see

None

SCRipt subsystem

The SCRipt subsystem controls macro or instrument setup scripts. For additional information on macro scripts, refer to Saving front-panel settings into a macro script (on page 3-52).

:SCRipt:RUN

This command runs a script.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
SCRipt:RUN "<scriptName>"
```

`<scriptName>`	The name of the script

Details

The script must be available in the instrument to be used by this command.

Example

`SCR:RUN "bufferCreate"`	Runs a script named `bufferCreate`.

Also see

Saving front-panel settings into a macro script (on page 3-52)
Scripts menu (on page 3-33)

SENSe1 subsystem

The SENSe1 subsystem commands configure and control the measurement functions of the instrument.

Many of these commands are set for a specific function (current, voltage, or resistance). For example, you can program a range setting for each function. The settings are saved with that function.

[:SENSe[1]]:<function>:AVERage:COUNt

This command sets the number of measurements that are averaged when filtering is enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	10

Usage

```
[:SENSe[1]]:<function>:AVERage:COUNt <n>
[:SENSe[1]]:<function>:AVERage:COUNt <DEF|MIN|MAX>
[:SENSe[1]]:<function>:AVERage:COUNt?
[:SENSe[1]]:<function>:AVERage:COUNt? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<n>`	The number of readings required for each filtered measurement (1 to 100)
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. When the filter count is larger, more filtering is done, and the data is less noisy.

NOTE

If you send this command without the `<function>` parameter, it sets the filter count for all functions.

Example 1

`CURR:AVER:COUNT 10`
`CURR:AVER:TCON MOV`
`CURR:AVER ON`
	For current measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 2

`RES:AVER:COUNT 10`
`RES:AVER:TCON MOV`
`RES:AVER ON`
	For resistance measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 3

`VOLT:AVER:COUNT 10`
`VOLT:AVER:TCON MOV`
`VOLT:AVER ON`
	For voltage measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 5-24)
[:SENSe[1]]:<function>:AVERage[:STATe] (on page 12-44)
[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 12-45)

[:SENSe[1]]:<function>:AVERage[:STATe]

This command enables or disables the averaging filter for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
[:SENSe[1]]:<function>:AVERage[:STATe] <state>
[:SENSe[1]]:<function>:AVERage[:STATe]?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<state>`	The filter status; set to one of the following values: ■ Disable the averaging filter: `OFF` or `0` ■ Enable the averaging filter: `ON` or `1`

Details

This command enables or disables the averaging filter. When this is enabled, the reading returned by the instrument is an averaged value, taken from multiple measurements. The settings of the filter count and filter type for the selected measure function determines how the reading is averaged.

NOTE

If you send this command without the `<function>` parameter, it sets the state of the averaging filter for all functions.

Example 1

`CURR:AVER:COUNT 10`
`CURR:AVER:TCON MOV`
`CURR:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 2

`RES:AVER:COUNT 10`
`RES:AVER:TCON MOV`
`RES:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 3

`VOLT:AVER:COUNT 10`
`VOLT:AVER:TCON MOV`
`VOLT:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 5-24)
[:SENSe[1]]:<function>:AVERage:COUNt (on page 12-43)
[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 12-45)

[:SENSe[1]]:<function>:AVERage:TCONtrol

This command sets the type of averaging filter that is used for the selected measure function when the measurement filter is enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	REP

Usage

```
[:SENSe[1]]:<function>:AVERage:TCONtrol <type>
[:SENSe[1]]:<function>:AVERage:TCONtrol?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<type>`	The filter type to use when filtering is enabled; set to one of the following values: ■ Repeating filter: `REPeat` ■ Moving filter: `MOVing`

Details

This command selects the type of averaging filter: Repeating average or moving average.

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed, and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

NOTE

When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

NOTE

If you send this command without the `<function>` parameter, it sets the filter type for all functions.

Example 1

`CURR:AVER:COUNT 10`
`CURR:AVER:TCON MOV`
`CURR:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 2

`RES:AVER:COUNT 10`
`RES:AVER:TCON REP`
`RES:AVER ON`
	Set the averaging filter type to repeating average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 5-24)
[:SENSe[1]]:<function>:AVERage:COUNt (on page 12-43)
[:SENSe[1]]:<function>:AVERage[:STATe] (on page 12-44)

[:SENSe[1]]:<function>:AZERo[:STATe]

This command enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
[:SENSe[1]]:<function>:AZERo[:STATe] <state>
[:SENSe[1]]:<function>:AZERo[:STATe]?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<state>`	The status of autozero: ■ Disable autozero: `OFF` or `0` ■ Enable autozero: `ON` or `1`

Details

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The 2470 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

The time to make the reference measurements is in addition to the normal measurement time. If timing is critical, such as in sweeps, you can disable autozero to avoid this time penalty.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

NOTE

If you send this command without the `<function>` parameter, it sets autozero for all functions.

Example

`VOLT:AZER OFF`	Sets autozero off for voltage measurements.

Also see

Automatic reference measurements (on page 4-44)
[:SENSe[1]]:AZERo:ONCE (on page 12-60)

[:SENSe[1]]:<function>:DELay:USER<n>

This command sets a user-defined delay that you can use in the trigger model.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list Function change	Save settings Measure configuration list	0 (0 s)

Usage

```
[:SENSe[1]]:<function>:DELay:USER<n> <delayTime>
[:SENSe[1]]:<function>:DELay:USER<n> <DEF|MIN|MAX>
[:SENSe[1]]:<function>:DELay:USER<n>?
[:SENSe[1]]:<function>:DELay:USER<n>? <DEF|MIN|MAX>
```

<function>	The measure function: ■ Current: CURRent[:DC] ■ Resistance: RESistance ■ Voltage: VOLTage[:DC]
<n>	The user delay to which this time applies (1 to 5)
<delayTime>	The delay (0 for no delay, or 167 ns to 10 ks)
<DEF|MIN|MAX>	The DEFault, MINimum, or MAXimum value

Details

To use this command in a trigger model, assign the delay to the dynamic delay block using the corresponding MEAS<n> parameter that matches the delay number specified here (see the Example below).

The delay is specific to the selected function.

Example

:CURRent:DELay:USER1 0.2 :TRIGger:BLOCk:DELay:DYNamic 6, MEAS1	Set user delay 1 to 0.2 s for current measurements. Set trigger block 6 to be a dynamic delay that is set to user delay 1 for the function being measured.

Also see

:TRIGger:BLOCk:DELay:DYNamic (on page 12-164)

[:SENSe[1]]:<function>:NPLCycles

This command sets the time that the input signal is measured for the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[:SENSe[1]]:<function>:NPLCycles <n>
[:SENSe[1]]:<function>:NPLCycles <DEF|MIN|MAX>
[:SENSe[1]]:<function>:NPLCycles?
[:SENSe[1]]:<function>:NPLCycles? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<n>`	The number of power-line cycles for each measurement: 0.01 to 10
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified as the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50). For 60 Hz, if you set the NPLC to 0.1, the measure time is 1.667 ms.

This command is set for the measurement of specific functions (current, resistance, or voltage).

The shortest amount of time results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

NOTE

If you send this command without the `<function>` parameter, it sets the NPLCs for all functions.

Example 1

`CURR:NPLC 0.5`	Sets the measurement time for current measurements to 0.0083 s (0.5/60).

Example 2

`RES:NPLC 0.5`	Sets the measurement time for resistance measurements to 0.0083 s (0.5/60).

Example 3

`VOLT:NPLC 0.5`	Sets the measurement time for voltage measurements to 0.0083 s (0.5/60).

Also see

Using NPLCs to adjust speed and accuracy (on page 5-9)

[:SENSe[1]]:<function>:OCOMpensated

This command determines if offset compensation is used.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
[:SENSe[1]]:<function>:OCOMpensated <state>
[:SENSe[1]]:<function>:OCOMpensated?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<state>`	Disable offset compensation: `OFF` or `0` Enable offset compensation: `ON` or `1`

Details

The voltage offsets caused by the presence of thermoelectric EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

This feature is only applied to resistance measurements or when `[:SENSe[1]]:CURRent:UNIT` or `[:SENSe[1]]:VOLTage:UNIT` is set to `OHM`.

Example

`*RST`
`:SENS:FUNC "RES"`
`:SENS:RES:RANG:AUTO ON`
`:RES:OCOM ON`
`:COUNT 5`
`:OUTP ON`
`:TRAC:TRIG "defbuffer1"`
`:TRAC:DATA? 1, 5, "defbuffer1", SOUR, READ`
`:OUTP OFF`
	Reset the instrument.
	Set the measurement function to resistance and set the range to automatic.
	Turn offset-compensated ohms on.
	Set the measurement count to 5.
	Turn the output on.
	Make measurements and store them in `defbuffer1`.
	Retrieve readings 1 to 5 with the source value and measurement values.
	Turn the output off.

Also see

Offset-compensated ohms (on page 4-32)

[:SENSe[1]]:<function>:RANGe:AUTO

This command determines if the measurement range is set manually or automatically for the selected measure function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO <state>
[:SENSe[1]]:<function>:RANGe:AUTO?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<state>`	Set the measurement range manually: `OFF` or `0` Set the measurement range automatically: `ON` or `1`

Details

Autorange selects the best range in which to measure the signal that is applied to the input terminals of the instrument. When autorange is enabled, the range increases at 100 percent of range. The range decreases occur when the reading is <10 percent of nominal range.

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

`RES:RANG:AUTO ON`	Set the range to be selected automatically for resistance measurements.

Also see

[:SENSe[1]]:<function>:RANGe[:UPPer] (on page 12-55)
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 12-52)

[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit

This command selects the lower limit for measurements of the selected function when the range is selected automatically.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Current: 10 nA Voltage: 0.2 V

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit <n>
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit?
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<n>`	The lower limit: ■ Current: 10 nA to 100 mA ■ Resistance: 2 Ω to 200 MΩ ■ Voltage: 0.2 V to 200 V
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

You can use this command when automatic range selection is enabled. It prevents the instrument from selecting a range that is below this limit. Because the lowest ranges generally require longer settling times, setting the low limit that is appropriate for your application but above the lowest possible range can make measurements require less settling time.

The lower limit must be less than the upper limit.

Though you can send any value when you send this command, the instrument selects the next highest range value. For example, if you send 15 for the lowest voltage range, the instrument will be set to the 20 V range as the low limit.

Example

`:VOLT:RANG:AUTO:LLIM 15` `:VOLT:RANG:AUTO:LLIM?`	Set the low range for voltage measurements to 20 V. Output: `2.000000E+01`

Also see

Ranges (on page 4-39)
[:SENSe[1]]:<function>:RANGe:AUTO (on page 12-51)

[:SENSe[1]]:<function>:RANGe:AUTO:REBound

This command determines if the instrument restores the measure range to match the limit range after making a measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO:REBound <state>
[:SENSe[1]]:<function>:RANGe:AUTO:REBound?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Voltage: `VOLTage[:DC]`
`<state>`	Do not restore the measurement range after each measurement: `OFF` or `0` Restore the measurement range after each measurement: `ON` or `1`

Details

The effective source limit is the lesser of either the programmed source limit or 105% of the active measure range. If you use fixed measure ranges, the instrument prevents you from selecting different limit and measure ranges. However, if measure autorange is selected, it is possible for the autorange process to cause the ranges to differ because the instrument may go down to a range that is lower than the one on which the source limit is programmed. This causes the effective source limit to drop to 105% of the newly selected measure range. For example, the output may be limited if you make a measurement that causes the range to be lowered and then increase the source level or make a change to the device under test or an external source. In this situation, the source voltage or current is limited to conform with the lower measurement range until another measurement causes the instrument to select a higher range to accommodate the new source value. If no further measurement is made, the source limit may remain limited to the present measurement range indefinitely.

When autorange rebound is enabled, it prevents the source from being limited to a value that is below the source limit. After an autoranged measurement is made, the measure range is restored to match the limit range once the autoranged measurement is complete. This ensures that the source does not limit at less than the full limit setting.

Example

`CURR:RANG:AUTO ON` `CURR:RANG:AUTO:REB ON`	Set the range to be selected automatically for current measurements. Set the measure range to be automatically restored to match the source limit value after each measurement.

Also see

[:SENSe[1]]:<function>:RANGe:AUTO (on page 12-51)

[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit

When autorange is selected, this command represents the highest measurement range that is used when the instrument selects the measurement range automatically.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Resistance: 200e6

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit <n>
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit?
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit? <DEF|MIN|MAX>
```

`<function>`	The measurement function to which this setting applies: ■ Current (query only): `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage (query only): `VOLTage[:DC]`
`<n>`	The upper limit: ■ Current: 1e-8 A to 1 A ■ Resistance: 2 Ω to 200 MΩ ■ Voltage: 0.2 V to 1000 V
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command can be written to and read for resistance measurements. For current and voltage measurements, it can only be read.

For current and voltage measurements, the upper limit is controlled by the current or voltage limit.

For resistance measurements, you can use this command when automatic range selection is enabled to put an upper bound on the range that is used for resistance measurements.

The upper limit must be more than the lower limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`:SENSe:RESistance:RANGe:AUTO:ULIMit 20`	Set the upper limit to 20 Ω.

Also see

[:SENSe[1]]:<function>:RANGe:AUTO (on page 12-51)
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 12-52)

[:SENSe[1]]:<function>:RANGe[:UPPer]

This command determines the positive full-scale measure range.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Current: 10 nA Resistance: 200 MΩ Voltage: 0.2 V

Usage

```
[:SENSe[1]]:<function>:RANGe[:UPPer] <n>
[:SENSe[1]]:<function>:RANGe[:UPPer] <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RANGe[:UPPer]?
[:SENSe[1]]:<function>:RANGe[:UPPer]? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<n>`	Set this command to a specific value or a preset value: ■ Current: 10 nA to 1 A ■ Resistance: 2 Ω to 200 MΩ ■ Voltage: 0.20 V to 1000 V
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

You can assign any real number using this command. The instrument selects the closest fixed range that is large enough to measure the entered number. For example, for current measurements, if you expect a reading of approximately 9 mA, set the range to 9 mA to select the 10 mA range. When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using.

This command is primarily intended to eliminate the time that is required by the instrument to automatically search for a range.

When a range is fixed, any signal greater than the entered range generates an overrange condition. When an overrange condition occurs, the front panel displays "Overflow" and the remote interface returns `9.9e+37`.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range, regardless of measurement range setting. However, the setting for the measure range is retained, and when the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

If you change the range while the output is off, the instrument does not update the hardware settings, but if you read the range setting, the return is the setting that will be used when the output is turned on. If you set a range while the output is on, the new setting takes effect immediately.

NOTE

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the selected measurement function (if supported by that function).

The range for measure functions defaults to autorange for all measure functions. If you switch from a fixed range to autorange, autorange is set to off. The range remains at the fixed range until a measurement is made, at which time the range is set to accommodate the new measurement.

Example 1

`:SENS:CURR:RANG 10E-6`	Select the 10 μA range.

Example 2

`:SENS:RES:RANG 2E6`	Select the 2 MΩ range.

Example 3

`:SENS:VOLT:RANG 200e-3`	Select the 200 mV range.

Also see

Ranges (on page 4-39)
[:SENSe[1]]:<function>:RANGe:AUTO (on page 12-51)

[:SENSe[1]]:<function>:RELative

This command contains the relative offset value.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
[:SENSe[1]]:<function>:RELative <n>
[:SENSe[1]]:<function>:RELative <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RELative?
[:SENSe[1]]:<function>:RELative? <DEF|MIN|MAX>
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`

<n>	The relative offset value: ■ Current: -1.05 to 1.05 ■ Resistance: –1e10 to 1e10 ■ Voltage: -1100 to 1100
<DEF|MIN|MAX>	The DEFault, MINimum, or MAXimum value

Details

This command specifies the relative offset value that can be applied to new measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

You can set this value, or have the instrument acquire a value. If the instrument acquires the value, read this setting to return the value that was measured internally.

Example

CURR:REL 0.5 CURR:REL:STAT ON	Set the relative offset for current measurements to 0.5. Enable relative offset.

Also see

Relative offset (on page 4-47)
[:SENSe[1]]:<function>:RELative:ACQuire (on page 12-57)
[:SENSe[1]]:<function>:RELative:STATe (on page 12-58)

[:SENSe[1]]:<function>:RELative:ACQuire

This command acquires a measurement and stores it as the relative offset value.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:<function>:RELative:ACQuire
```

<function>	The measure function: ■ Current: CURRent[:DC] ■ Resistance: RESistance ■ Voltage: VOLTage[:DC]

Details

This command triggers the instrument to make a new measurement for the selected function. This measurement is then stored as the new relative offset level.

When you send this command, the instrument does not apply any math, limit test, or filter settings to the measurement, even if they are set. It is a measurement that is made as if these settings are disabled.

You must change to the function for which you want to acquire a value before sending this command.

The instrument must have relative offset enabled to use the acquired relative offset value.

After executing this command, you can use the [:SENSe[1]]:<function>:RELative? command to return the last relative level value that was acquired or set.

Example

`FUNC "RES"`
`RES:REL:ACQ`
`RES:REL?`
`RES:REL:STAT ON`
	Switch to resistance measurements. Acquire a relative offset value for resistance measurements.
	Query for the offset value.
	Turn relative offset on.
	Example output:
	`-5.4017E-10`

Also see

[:SENSe[1]]:<function>:RELative (on page 12-56)
[:SENSe[1]]:<function>:RELative:STATe (on page 12-58)

[:SENSe[1]]:<function>:RELative:STATe

This command enables or disables the application of a relative offset value to the measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
[:SENSe[1]]:<function>:RELative:STATe <state>
[:SENSe[1]]:<function>:RELative:STATe?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<state>`	Disable the relative offset: `OFF` or `0` Enable the relative offset: `ON` or `1`

Details

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value. You can enter a relative offset value or have the instrument acquire a relative offset value.

Each returned measured relative reading is the result of the following calculation:

Displayed reading = Actual measured reading - Relative offset value

Example

`:SENS:FUNC "VOLT"`
`:SENS:VOLT:REL 5`
`:SENSe:VOLT:REL:STATe ON`
	Set the measurement function to volts with a relative offset of 5 V and enable the relative offset function.

Also see

Relative offset (on page 4-47)
[:SENSe[1]]:<function>:RELative (on page 12-56)
[:SENSe[1]]:<function>:RELative:ACQuire (on page 12-57)

[:SENSe[1]]:<function>:RSENse

This command selects local (2-wire) or remote (4-wire) sensing.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
[:SENSe[1]]:<function>:RSENse <state>
[:SENSe[1]]:<function>:RSENse?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`
`<state>`	Disable remote sensing (2-wire): `0` or `OFF` Enable remote sensing (4-wire): `1` or `ON`

Details

This command determines if 2-wire (local) or 4-wire (remote) sensing is used.

When you use 4-wire sensing, voltages are measured at the device under test (DUT). For the source voltage, if the sensed voltage is lower than the programmed amplitude, the voltage source increases the voltage until the sensed voltage is the same as the programmed amplitude. This compensates for IR drop in the output test leads.

Using 4-wire sensing with voltage measurements eliminates any voltage drops that may be in the test leads between the 2470 and the DUT.

When you are using 2-wire sensing, voltage is measured at the output connectors.

When you are measuring resistance, you can enable 4-wire sensing to make 4-wire resistance measurements.

When the output is off, 4-wire sensing is disabled and the instrument uses 2-wire sense, regardless of the sense setting. When the output is on, the selected sense setting is used.

If the output is on when you change the sense setting, the output is turned off.

Example

`VOLT:RSEN ON`	Set the remote sense for voltage measurements.

Also see

Two-wire local sense connections (on page 4-8)
Four-wire remote sense connections (on page 4-10)

[:SENSe[1]]:<function>:UNIT

This command sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Current: AMP Voltage: VOLT

Usage

```
[:SENSe[1]]:<function>:UNIT <unitOfMeasure>
[:SENSe[1]]:<function>:UNIT?
```

`<function>`	The measure function: ■ Current: `CURRent[:DC]` ■ Voltage: `VOLTage[:DC]`
`<unitOfMeasure>`	Current: `OHM`, `WATT`, or `AMP` Voltage: `OHM`, `WATT`, or `VOLT`

Details

The change in measurement units is displayed when the next measurement is made.

Example

`VOLT:UNIT WATT`	Changes the front-panel display and buffer readings for voltage measurements to be displayed as power readings in watts.

Also see

None

[:SENSe[1]]:AZERo:ONCE

This command causes the instrument to refresh the reference and zero measurements once.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:AZERo:ONCE
```

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting for the selected function.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

`FUNC "VOLT"` `AZER:ONCE`	Do a one-time refresh of the reference and zero measurements for the voltage function.

Also see

Automatic reference measurements (on page 4-44)
[:SENSe[1]]:<function>:AZERo[:STATe] (on page 12-47)

[:SENSe[1]]:CONFiguration:LIST:CATalog?

This command returns the name of one measure configuration list that is stored on the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:CATalog?
```

Details

You can use this command to retrieve the names of measure configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns an empty string when there are no more names to return. If the command returns an empty string the first time you send it, no measure configuration lists have been created for the instrument.

Example

`CONF:LIST:CAT?`	Send this command to retrieve the name of one measure configuration list. To get all stored lists, send it again until it returns an empty string.

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62)

[:SENSe[1]]:CONFiguration:LIST:CREate

This command creates an empty measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:CREate "<name>"
```

`<name>`	A string that represents the name of a measure configuration list

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. To save a configuration list, use a saved setup to store the instrument settings, which include defined configuration lists.

Example

```
:SENS:CONF:LIST:CRE "MyMeasList"
```

Creates a measure configuration list named `MyMeasList`.

Also see

*SAV (on page 12-9)
Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:STORe (on page 12-66)

[:SENSe[1]]:CONFiguration:LIST:DELete

This command deletes a measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:DELete "<name>"
[:SENSe[1]]:CONFiguration:LIST:DELete "<name>", <index>
```

`<name>`	A string that represents the name of a measure configuration list
`<index>`	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

`:SENSe:CONF:LIST:DELete "myMeasList"`	Deletes a configuration list named `myMeasList`.
`:SENSe:CONF:LIST:DELete "myMeasList", 2`	Deletes configuration index 2 in a configuration list named `myMeasList`.

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62)

[:SENSe[1]]:CONFiguration:LIST:QUERy?

This command returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:QUERy? "<name>", <index>
[:SENSe[1]]:CONFiguration:LIST:QUERy? "<name>", <index>, <fieldSeparator>
```

`<name>`	A string that represents the name of a measure configuration list
`<index>`	A number that defines a specific configuration index in the configuration list
`<fieldSeparator>`	A separator for the data: ■ Comma (default): `COMMa` or `1` ■ Semicolon: `SEMicolon` or `2` ■ New line: `NEWLine or 3`

Details

This command recalls data for one configuration index.

For additional information about the information this command recalls when using a configuration list query command, see Settings stored in a measure configuration index (on page 4-83).

Example

```
:SENS:CONF:LIST:QUER? "MyMeasList", 2, NEWL
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62)
*SAV (on page 12-9)
TSP command reference (on page 14-1)

[:SENSe[1]]:CONFiguration:LIST:RECall

This command recalls a configuration index in a measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:RECall "<name>"
[:SENSe[1]]:CONFiguration:LIST:RECall "<name>", <index>
```

`<name>`	A string that represents the name of a measure configuration list
`<index>`	A number that defines a specific configuration index in the measure configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a measure configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified measure configuration list.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, event code 2790, "Configuration list, error, does not exist" is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. A single index stores the settings associated with a single measure function. To see the settings that will be recalled with an index, use the `[:SENSe[1]]:CONFiguration:LIST:QUERy?` command.

NOTE

If you are going to recall a source configuration list separately (not with this command), recall the source configuration list before the measure configuration list. This order ensures that dependencies between source and measure settings will be properly handled.

For additional information about the information this command recalls when using a configuration list query command, see Settings stored in a measure configuration index (on page 4-83).

Example

`:SENSe:CONF:LIST:RECall "MyMeasList", 5`	Recalls configuration index 5 in a configuration list named `MyMeasList`.
`:SENSe:CONF:LIST:RECall "MyMeasList"`	Because an index was not specified, this command recalls configuration index 1 from a configuration list named `MyMeasList`.

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62)
*SAV (on page 12-9)
[:SENSe[1]]:CONFiguration:LIST:QUERy? (on page 12-63)
[:SENSe[1]]:CONFiguration:LIST:STORe (on page 12-66)
:SOURce[1]:CONFiguration:LIST:RECall (on page 12-72)

[:SENSe[1]]:CONFiguration:LIST:SIZE?

This command returns the size (number of configuration indexes) of a measure configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:SIZE? "<name>"
```

`<name>`	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration indexes) of a measure configuration list. The size of the list is equal to the number of configuration indexes in a configuration list.

Example

`:SENSe:CONF:LIST:SIZE? "MyMeasList"`	Returns the number of configuration indexes in a measure configuration list named `MyMeasList`. Example output: 3

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62)
*SAV (on page 12-9)

[:SENSe[1]]:CONFiguration:LIST:STORe

This command stores the active measure settings into the named configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Saved settings	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:STORe "<name>"
[:SENSe[1]]:CONFiguration:LIST:STORe "<name>", <index>
```

`<name>`	A string that represents the name of a measure configuration list
`<index>`	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active measure settings to a configuration index in a configuration list. If the index parameter is not provided, the new settings are appended to the end of the list. The index only stores the active settings for a single active measure function.

Configuration lists are not saved when the instrument is turned off or reset. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Refer to Settings stored in a measure configuration index (on page 4-83) for a complete list of measure settings that the instrument stores.

Example

`:SENS:CONF:LIST:STOR "MyConfigList"`	Stores the active settings of the instrument to the end of the configuration list named `MyConfigList`.
`:SENS:CONF:LIST:STOR "MyConfigList", 5`	Stores the active settings of the instrument to the configuration list named `MyConfigList` in configuration index 5.

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 12-62)
*SAV (on page 12-9)

[:SENSe[1]]:COUNt

This command sets the number of measurements to make when a measurement is requested.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[:SENSe[1]]:COUNt <n>
[:SENSe[1]]:COUNt <DEF|MIN|MAX>
[:SENSe[1]]:COUNt?
[:SENSe[1]]:COUNt? <DEF|MIN|MAX>
```

`<n>`	The number of measurements (1 to 300,000 or buffer capacity)
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the number of measurements that are made when a measurement is requested. This command does not affect the trigger model.

This command sets the count for all measure functions.

If you set the count to a value that is larger than the capacity of the reading buffer and the buffer fill mode is set to continuous, the buffer wraps until the number of readings specified have occurred. The earliest readings in the count are overwritten. If the buffer is set to fill once, readings stop when the buffer is filled, even if the count is not complete.

NOTE

To get better performance from the instrument, use the SimpleLoop trigger-model template instead of using the count command.

Example

```
:SENS:FUNC "CURR"
:TRAC:CLEAR
:COUN 10
:MEAS?
:TRAC:DATA? 1, 10
```

Clear data from the reading buffer.
Set the count to 10.
Make ten measurements.
Returns the last measurement.
Example output:

```
-5.693831E-05
```

Read all ten measurements.
Example output:

```
-7.681046E-05,-2.200288E-04,-9.086048E-05,-6.388056E-05,-7.212282E-05,-4.874761E-0
    5,-4.741654E-04,-6.811028E-05,-5.110232E-05,-5.693831E-05
```

Also see

:MEASure? (on page 12-3)
:TRACe:DATA? (on page 12-121)
:TRIGger:LOAD "SimpleLoop" (on page 12-192)

[:SENSe[1]]:FUNCtion[:ON]

This command selects the active measure function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	CURR

Usage

```
[:SENSe[1]]:FUNCtion[:ON] "<function>"
[:SENSe[1]]:FUNCtion[:ON]?
```

`<function>`	A string that contains the measure function: ■ Current: `CURRent[:DC]` ■ Resistance: `RESistance` ■ Voltage: `VOLTage[:DC]`

Details

Set this command to the type of measurement you want to make.

Reading this command returns the measure function that is presently active.

Example

`:FUNC "VOLTage"`	Make the voltage measurement function the active function.

Also see

Making resistance measurements (on page 4-26)
Source and measure using SCPI commands (on page 4-33)

SOURce subsystem

The commands in the SOURce subsystem configure and control the current source and voltage source.

:SOURce[1]:CONFiguration:LIST:CATalog?

This command returns the name of one source configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:CATalog?
```

Details

You can use this command to see the names of source configuration lists stored on the instrument.

This command returns one name each time you send it. This command returns an empty string if there are no more names to return. If the command returns an empty string the first time you send it, no source configuration lists have been created for the instrument.

Example

`:SOUR:CONF:LIST:CAT?`	Send this command to return the name of one source configuration list stored on the instrument. To get all stored configuration lists, resend this command until it returns an empty string.

Also see

Configuration lists (on page 4-82)
:SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)

:SOURce[1]:CONFiguration:LIST:CREate

This command creates an empty source configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:CREate "<name>"
```

`<name>`	A string that represents the name of a source configuration list

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list, use a saved setup to store the instrument settings, which include defined configuration lists.

Configuration list names must be unique. For example, the name of a source configuration list cannot be the same as the name of a measure configuration list.

Example

```
:SOUR:CONF:LIST:CRE "MySourceList"
```

Creates a source configuration list named `MySourceList`.

Also see

Configuration lists (on page 4-82)
*SAV (on page 12-9)
:SOURce[1]:CONFiguration:LIST:STORe (on page 12-73)

:SOURce[1]:CONFiguration:LIST:DELete

This command deletes a source configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:DELete "<name>"
:SOURce[1]:CONFiguration:LIST:DELete "<name>", <index>
```

`<name>`	A string that represents the name of a source configuration list
`<index>`	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

`:SOURce:CONF:LIST:DEL "MySourceList"`	Deletes a configuration list named `MySourceList`.
`:SOURce:CONF:LIST:DEL "MySourceList", 2`	Deletes configuration index 2 in the configuration list named `MySourceList`.

Also see

Configuration lists (on page 4-82)
:SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)

:SOURce[1]:CONFiguration:LIST:QUERy?

This command returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:QUERy? "<name>", <point>
:SOURce[1]:CONFiguration:LIST:QUERy? "<name>", <point>, <fieldSeparator>
```

`<name>`	A string that represents the name of a source configuration list
`<point>`	A number that defines a specific configuration index in the configuration list
`<fieldSeparator>`	A separator for the data: ■ Comma (default): `COMMa` or `1` ■ Semicolon: `SEMicolon` or `2` ■ New line: `NEWLine or 3`

Details

This command can only return data for one configuration index. To get data for additional configuration indexes, resend the command and specify different configuration indexes.

Refer to Settings stored in a source configuration list (on page 4-83) for a complete list of source settings that the instrument stores in a source configuration list.

Example

```
:SOUR:CONF:LIST:QUER? "MySourceList", 2
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Also see

Configuration lists (on page 4-82)
:SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)

:SOURce[1]:CONFiguration:LIST:RECall

This command recalls a specific configuration index in a specific source configuration list and an optional measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:RECall "<name>", <index>
:SOURce[1]:CONFiguration:LIST:RECall "<name>", <index>, "<measureListName>"
:SOURce[1]:CONFiguration:LIST:RECall "<name>", <index>, "<measureListName>",
   <measureIndex>
```

`<name>`	A string that represents the name of a source configuration list
`<index>`	A number that defines a specific configuration index in the source configuration list
`<measureListName>`	A string that represents the name of a measure configuration list
`<measureIndex>`	A number that defines a specific configuration index in the measure configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a specific source configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified source configuration list of that index.

You can optionally specify a measure configuration list and index to recall with the source settings. If you do not specify a measure index, the measure index defaults to match the source index. Specify a source and measure list together with this command to allow the instrument to coordinate the application of the settings in the two lists appropriately. If you do not need have the application of the source and measure configuration lists coordinated, you can specify just the source configuration list with this command and use the [:SENSe[1]]:CONFiguration:LIST:RECall (on page 12-64) command to recall measure settings separately in your application.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, event code 2790, "Configuration list, error, does not exist" is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. To see the settings that are recalled with an index, use the :SOURce[1]:CONFiguration:LIST:QUERy? (on page 12-71) command.

NOTE

To recall a source configuration list separately (not with this command), recall the source configuration list before the measure configuration list. This order ensures that dependencies between source and measure settings will be properly handled.

Example

`:SOURce:CONF:LIST:REC "MySourceList", 5`	Recalls configuration index 5 in a configuration list named `MySourceList`.
`:SOURce:CONF:LIST:RECall "MySourceList"`	Because an index was not specified, this command recalls configuration index 1 from a configuration list named `MySourceList`.

Also see

Configuration lists (on page 4-82)
:SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)

:SOURce[1]:CONFiguration:LIST:SIZE?

This command returns the number of configuration indexes of a source configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Query

```
:SOURce[1]:CONFiguration:LIST:SIZE? "<name>"
```

`<name>`	A string that represents the name of a source configuration list

Details

The size of the list is equal to the number of configuration indexes in a configuration list.

Example

`:SOUR:CONF:LIST:SIZE? "MySourceList"`	Returns the number of configuration indexes in a source configuration list named `MySourceList`.

Also see

Configuration lists (on page 4-82)
:SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)

:SOURce[1]:CONFiguration:LIST:STORe

This command stores the active source settings into the named configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle Source configuration list	Saved settings	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:STORe "<name>", <index>
```

`<name>`	A string that represents the name of a source configuration list
`<index>`	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active source settings to a configuration index in a configuration list. If the index is defined, the configuration list is stored in that index. If the index is not defined, the configuration index is appended to the end of the list. If a configuration index already exists for the specified index, the new configuration overwrites the existing configuration index.

Refer to Settings stored in a source configuration index (on page 4-83) for information about the settings this command stores.

Example

`:SOURce:CONF:LIST:CRE "biasLevel"`
`:SOURce:FUNC VOLT`
`:SOURce:VOLT:LEV 5`
`:SOURce:CONF:LIST:STORE "biasLevel"`
	Create a configuration list named `biasLevel`.
	Set the source function to voltage and the source voltage level to 5 V.
	Store the configuration list and append it to the end of the `biasLevel` configuration list.

Also see

:SOURce[1]:CONFiguration:LIST:CREate (on page 12-69)

:SOURce[1]:<function>:DELay

This command contains the source delay.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0

Usage

```
:SOURce[1]:<function>:DELay <value>
:SOURce[1]:<function>:DELay <DEF|MIN|MAX>
:SOURce[1]:<function>:DELay?
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<value>`	The delay in seconds (0 to 10 ks)
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets a delay for the selected source function. This delay is in addition to normal settling times.

After the programmed source is turned on, this delay allows the source level to settle before a measurement is made.

If you set a specific source delay (`smu.source.delay`), source autodelay is turned off.

When source autodelay is turned on, the manual source delay setting is overwritten with the autodelay setting.

When either a source delay or autodelay is set, the delay is applied to the first source output and then only when the magnitude of the source changes.

NOTE

If you send this command without the `<function>` parameter, it sets the delay for all functions.

Example

`SOUR:VOLT:DEL DEF`	Set the delay for the voltage source to the default value.

Also see

:SOURce[1]:<function>:DELay:AUTO (on page 12-75)

:SOURce[1]:<function>:DELay:AUTO

This command enables or disables the automatic delay that occurs when the source is turned on.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	1 (ON)

Usage

```
:SOURce[1]:<function>:DELay:AUTO <state>
:SOURce[1]:<function>:DELay:AUTO?
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<state>`	Disable the source auto delay: `OFF` or `0` Enable the source auto delay: `ON` or `1`

Details

When autodelay is turned on, the actual delay that is set depends on the range.

When source autodelay is on, if you set a source delay, the autodelay is turned off.

When source autodelay is on, the manual source delay setting is overwritten with the autodelay setting.

The delay is applied to the first source output and then only when the magnitude of the source changes.

Example

`SOUR:CURR:DEL:AUTO OFF`	Turn off auto delay when current is being sourced.

Also see

:SOURce[1]:<function>:DELay (on page 12-74)

:SOURce[1]:<function>:DELay:USER<n>

This command sets a user-defined delay that you can use in the trigger model.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0

Usage

```
:SOURce[1]:<function>:DELay:USER<n> <delayTime>
:SOURce[1]:<function>:DELay:USER<n> <DEF|MIN|MAX>
:SOURce[1]:<function>:DELay:USER<n>?
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<n>`	The number that identifies this user delay (1 to 5)
`<delayTime>`	The time of the delay in seconds (0 to 10,000)
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.

The delay is specific to the selected function.

Example

`:SOUR:VOLT:DEL:USER1 5`	Set user delay for source 1 to 5 s.
`:TRIG:BLOC:SOUR:STAT 1, ON`	Set trigger block 1 to turn the source output on.
`:TRIG:BLOC:DEL:DYN 2, SOUR1`	Set trigger block 2 to a dynamic delay that calls source user delay 1.
`:TRIG:BLOC:MDIG 3`	Set trigger block 3 to make a measurement.
`:TRIG:BLOC:SOUR:STAT 4, OFF`	Set trigger block 4 to turn the source output off.
`:TRIG:BLOC:BRAN:COUN 5, 10, 1`	Set trigger block 5 to branch to block 1 ten times.
`:INIT`	Start the trigger model.

Also see

:TRIGger:BLOCk:DELay:DYNamic (on page 12-164)

:SOURce[1]:<function>:HIGH:CAPacitance

This command enables or disables high-capacitance mode.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0 (OFF)

Usage

```
:SOURce[1]:<function>:HIGH:CAPacitance <state>
:SOURce[1]:<function>:HIGH:CAPacitance?
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<state>`	Turn high capacitance off: `OFF` or `0` Turn high capacitance on: `ON` or `1`

Details

When the instrument is measuring low current and is driving a capacitive load, you may see overshoot, ringing, and instability. You can enable the high capacitance mode to minimize these problems.

Example

`SOUR:CURR:HIGH:CAP ON`	Turn the high capacitance mode on when sourcing current.

Also see

High-capacitance operation (on page 5-23)

:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]

This command immediately selects a fixed amplitude for the selected source function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0

Usage

```
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] <value>
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] <DEF|MIN|MAX>
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]?
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]? <DEF|MIN|MAX>
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<value>`	The amplitude: ■ Current: –1.05 A to 1.05 A ■ Voltage: –1100 V to 1100 V
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the output level of the voltage or current source. If the output is on, the new level is sourced immediately.

The sign of the source level dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

If a manual source range is selected, the level cannot exceed the specified range. For example, if the voltage source is on the 2 V range, you cannot set the voltage source level to 3 V. When autorange is selected, the amplitude can be set to any level supported by the instrument.

Example

`SOUR:FUNC VOLT` `SOUR:VOLT 1`	Set the instrument to source voltage and set it to source 1 V.

Also see

:SOURce[1]:<function>:RANGe (on page 12-82)
:SOURce[1]:<function>:RANGe:AUTO (on page 12-84)

:SOURce[1]:<function>:<x>LIMit[:LEVel]

This command selects the source limit for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	Voltage: 105 µA Current: 210 mV

Usage

```
:SOURce[1]:CURRent:VLIMit[:LEVel] <value>
:SOURce[1]:CURRent:VLIMit[:LEVel] <DEF|MIN|MAX>
:SOURce[1]:CURRent:VLIMit[:LEVel]?
:SOURce[1]:CURRent:VLIMit[:LEVel]? <DEF|MIN|MAX>
:SOURce[1]:VOLTage:ILIMit[:LEVel] <value>
:SOURce[1]:VOLTage:ILIMit[:LEVel] <DEF|MIN|MAX>
:SOURce[1]:VOLTage:ILIMit[:LEVel]?
:SOURce[1]:VOLTage:ILIMit[:LEVel]? <DEF|MIN|MAX>
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<x>`	The function to which the limit applies: ■ Current: `I` ■ Voltage: `V`
`<value>`	The limit: ■ Current source function: -1.05 A to 1.05 A ■ Voltage source function: -1100 V to 1100 V
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command sets the source limit for measurements. The values that can be set for this command are limited by the setting for the overvoltage protection limit.

The 2470 cannot source levels that exceed this limit.

If you change the measure range to a range that is not appropriate for this limit, the instrument changes the source limit to a limit that is appropriate to the range and a warning is generated. Depending on the source range, your actual maximum limit value could be lower. The instrument makes adjustments to stay in the operating boundaries.

This value can also be limited by the measurement range. If a specific measurement range is set, the limit must be 10.6% or higher of the measurement range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

Limits are absolute values.

You can use `:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped?` to check the limit state of the source output.

Example

`:SOUR:CURR:VLIM 15`	Set the voltage limit to 15 V.

Also see

:SOURce[1]:<function>:PROTection[:LEVel] (on page 12-81)
:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped? (on page 12-80)

:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped?

This command indicates if the source exceeded the limits that were set for the selected measurements.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CURRent:VLIMit[:LEVel]:TRIPped?
:SOURce[1]:VOLTage:ILIMit[:LEVel]:TRIPped?
```

Details

You can use this command to check the limit state of the source.

If the limits were exceeded, the instrument clamps the source to keep the source within the set limits.

If the source did not exceed the set limits, the return is `0`. If the source did exceed the set limits, the return is `1`.

Example 1

`SOUR:CURR:VLIM:TRIP?`	Returns a value that indicates whether or not the source exceeded the current limits.

Example 2

`SOUR:VOLT:ILIM:TRIP?`	Return value indicates whether or not the source has exceeded the voltage limits.

Also see

:SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 12-79)

:SOURce[1]:FUNCtion[:MODE]

This command contains the source function, which can be voltage or current.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	VOLT

Usage

```
:SOURce[1]:FUNCtion[:MODE] <function>
:SOURce[1]:FUNCtion[:MODE]?
```

`<function>`	Voltage source function: `VOLTage` Current source function: `CURRent`

Details

When you set this command, it configures the instrument as either a voltage source or a current source.

Example

`SOUR:FUNC CURR` `SOUR:FUNC?`	Set the source function of the instrument to be a current source and query the source function. Output: `CURR`

Also see

None

:SOURce[1]:<function>:PROTection[:LEVel]

This command sets the overvoltage protection setting of the source output.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	NONE

Usage

```
:SOURce[1]:VOLTage:PROTection[:LEVel] <n>
:SOURce[1]:VOLTage:PROTection[:LEVel]?
```

`<n>`	The overvoltage protection level, set as `<n>`, where `<n>` is `PROT20`, `PROT40`, `PROT100`, `PROT200`, `PROT300`, `PROT400`, `PROT500`, or `NONE`

Details

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced.

This protection is in effect for both positive and negative output voltages.

When this attribute is used in a test sequence, it should be set before turning the source on.

⚠ WARNING

Even with the overvoltage protection set to the lowest value (20 V), never touch anything connected to the terminals of the 2470 when the instrument is on. Always assume that a hazardous voltage (greater than 30 V_{RMS}) is present when the instrument is on. To prevent damage to the device under test or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

Example

`SOUR:VOLT:PROT PROT40` `SOUR:VOLT:PROT?`	Set the voltage source protection to 40 V and query the value. The output is: `PROT40`

Also see

Overvoltage protection (on page 4-35)

:SOURce[1]:<function>:PROTection[:LEVel]:TRIPped?

This command indicates if the overvoltage source protection feature is active.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:VOLTage:PROTection[:LEVel]:TRIPped?
```

Details

When overvoltage protection is active, the instrument restricts the maximum voltage level that the instrument can source.

If the voltage source does not exceed the set limits, the return is `0`. If the voltage source exceeds the set limits, the return is `1`.

Example

`SOUR:VOLT:PROT:TRIP?`	If overvoltage protection is active, the output is: `1`

Also see

Overvoltage protection (on page 4-35)
:SOURce[1]:<function>:PROTection[:LEVel] (on page 12-81)

:SOURce[1]:<function>:RANGe

This command selects the range for the source for the selected source function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	Current: 1e-8 A Voltage: 0.2 V

Usage

```
:SOURce[1]:<function>:RANGe <value>
:SOURce[1]:<function>:RANGe <DEF|MIN|MAX>
:SOURce[1]:<function>:RANGe?
```

`<function>`	The source function to which this setting applies: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<value>`	Set to the maximum expected voltage or current to be sourced; see **Details** for values; the ranges are: ■ Current: –1.05 A to 1.05 A ■ Voltage: –1050 V to 1050 V
`<DEF|MIN|MAX>`	The `DEFault`, `MINimum`, or `MAXimum` value

Details

This command manually selects the measurement range for the specified source.

To select the range, specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 3 V, set the source level to 3. The 2470 selects the 20 V range.

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 200 mV, 2 V, 20 V, 200 V, and 1000 V.

When you read this value, the instrument returns the positive full-scale value that the instrument is presently using.

This command is intended to eliminate the time required by the automatic range selection.

To select the range, you can specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 50 mV, send 0.05 (or 50e-3) to select the 200 mV range.

NOTE

If automatic range selection is set to on, when you select a specific range, automatic is set to off. To set the range to automatic selection, use the source autorange command.

Example

`:SOURce:VOLTage:RANGe 3`	Send this command to source levels around 3 V. This example selects the 20 V range for the voltage source.

Also see

:SOURce[1]:<function>:RANGe:AUTO (on page 12-84)
Ranges (on page 4-39)

:SOURce[1]:<function>:RANGe:AUTO

This command determines if the range is selected manually or automatically for the selected source function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	1 (ON)

Usage

```
:SOURce[1]:CURRent:RANGe:AUTO <state>
:SOURce[1]:CURRent:RANGe:AUTO?
:SOURce[1]:VOLTage:RANGe:AUTO <state>
:SOURce[1]:VOLTage:RANGe:AUTO?
```

`<state>`	Disable automatic source range: `0` or `OFF` Enable automatic source range: `1` or `ON`

Details

This command indicates the state of the range for the selected source. When automatic source range is disabled, the source range is set manually.

When automatic source range is enabled, the instrument selects the range that is most appropriate for the value that is being sourced. The output level controls the range. If you read the range after the output level is set, the instrument returns the range that the instrument chose as appropriate for that source level.

If the source range is set to a specific value from the front panel or a remote command, the setting for automatic range is set to disabled.

Example

`SOUR:CURR:RANG:AUTO ON`	Enable the automatic source range.

Also see

:SOURce[1]:<function>:RANGe (on page 12-82)

:SOURce[1]:<function>:READ:BACK

This command determines if the instrument records the measured source value or the configured source value when making a measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	1 (ON)

Usage

```
:SOURce[1]:VOLTage:READ:BACK <state>
:SOURce[1]:VOLTage:READ:BACK?
:SOURce[1]:CURRent:READ:BACK <state>
:SOURce[1]:CURRent:READ:BACK?
```

`<state>`	Disable read back: `0` or `OFF` Enable read back: `1` or `ON`

Details

When source readback is off, the instrument records and displays the source value you set. When you use the actual source value (source readback on), the instrument measures the actual source value immediately before making the measurement of the device under test.

Using source readback results in more accurate measurements, but also a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the device-under-test measurement. When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the device-under-test measurement.

Example

`*RST`	Reset the instrument to default settings.
`TRAC:MAKE "MyBuffer", 100`	Make a buffer named "MyBuffer" that can hold 100 readings.
`SOUR:FUNC VOLT`	Set source function to voltage.
`SENS:FUNC "CURR"`	Set the measurement function to current.
`SOUR:VOLT:READ:BACK ON`	Set read back on.
`SOUR:VOLT 10`	Set the instrument to take 100 readings.
`COUNT 100`	Turn the output on.
`OUTP ON`	Take a measurement (100 readings).
`READ? "MyBuffer"`	Turn the output off.
`OUTP OFF`	Get the source values and measurements from the buffer.
`TRAC:DATA? 1, 100, "MyBuffer", SOUR, READ`	

Also see

:SOURce[1]:FUNCtion[:MODE] (on page 12-80)

:SOURce[1]:LIST:<function>

This command allows you to set up a list of custom values for a sweep.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:LIST:CURRent <list>
:SOURce[1]:LIST:CURRent?
:SOURce[1]:LIST:VOLTage <list>
:SOURce[1]:LIST:VOLTage?
```

`<list>`	Current: –1.05 A to 1.05 A Voltage: –1100 V to 1100 V

Details

This command defines a list of up to 100 source values for a source list. This list is used by the `:SOURce[1]:SWEep:<function>:LIST` command to define the source values for the sweep.

When you start the sweep, the instrument sequentially sources each current or voltage value in the list. A measurement is made at each source level.

If there is an existing list, it is replaced by the new list.

When you send this command, the instrument creates a source configuration list named `CurrCustomSweepList` if the function is set to current or `VoltCustomSweepList` if the function is set to voltage.

To add source values to an existing list, use the `:SOURce[1]:LIST:<function>:APPend` command.

Example

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SENS:CURR:RSEN OFF
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 1
SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5
SOUR:SWE:VOLT:LIST 1, 0.2
INIT
*WAI

TRAC:DATA? 1, 6, "defbuffer1", SOUR, READ
```

This example will source 1 V, 5 V, 1 V, 5 V, 1 V, 5 V and measure the resulting current at each voltage point. The time duration of each voltage point is 200 ms.

Also see

:SOURce[1]:LIST:<function>:APPend (on page 12-87)
:SOURce[1]:SWEep:<function>:LIST (on page 12-93)

:SOURce[1]:LIST:<function>:APPend

This command adds values to the source list for the selected source function.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:LIST:CURRent:APPend <list>
:SOURce[1]:LIST:VOLTage:APPend <list>
```

`<list>`	Current: –1.05 A to 1.05 A Voltage: –1100 V to 1100 V

Details

This adds up to 100 values to the list created with `:SOURce[1]:LIST:<function>`. The new values are added to the end of the existing values. You can have a total of 2500 values in a list, but you must append them in groups of 100.

If the list does not exist, this command creates one.

Example

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SENS:CURR:RSEN OFF
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 1
SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5
SOUR:LIST:VOLT:APP 1, 5, 1, 5, 1, 5
SOUR:SWE:VOLT:LIST 1, 0.2
INIT
*WAI

TRAC:DATA? 1, 12, "defbuffer1", SOUR, READ
```

This example will create a source configuration list (`VoltCustomSweepList`) and source 1 V, 5 V six times and measure the resulting current at each voltage point. The duration of each voltage point is 200 ms.

Also see

:SOURce[1]:LIST:<function> (on page 12-86)

:SOURce[1]:LIST:<function>:POINts?

This command queries the length of the source list for the selected source function.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:LIST:CURRent:POINts?
:SOURce[1]:LIST:VOLTage:POINts?
```

Details

This command returns the length of the specified source list. The response message indicates the number of source values in the list.

Example

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SENS:CURR:RSEN OFF
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 1
SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5
SOUR:SWE:VOLT:LIST 1, 0.2
INIT
*WAI

TRAC:DATA? 1, 6, "defbuffer1", SOUR, READ
SOUR:LIST:VOLT:POIN?
```

This example will source 1 V, 5 V, 1 V, 5 V, 1 V, 5 V and measure the resulting current at each voltage point. The time duration of each voltage point is 200 ms.
Check the number of points in the list.
Output:
6

Also see

:SOURce[1]:LIST:<function> (on page 12-86)
:SOURce[1]:LIST:<function>:APPend (on page 12-87)

:SOURce[1]:SWEep:<function>:LINear

This command sets up a linear sweep for a fixed number of measurement points.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, "<bufferName>"
```

`<function>`	Voltage sweep: `VOLTage` Current sweep: `CURRent`
`<start>`	The voltage or current source level at which the sweep starts: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
`<stop>`	The voltage or current at which the sweep stops: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
`<points>`	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
`<delay>`	The delay between measurement points; default is `-1`, which enables autodelay, or a specific delay value from 50 µs to 10,000 s, or 0 for no delay
`<count>`	The number of times to run the sweep; default is `1`: ■ Infinite loop: `0` ■ Finite loop: `1` to `268435455`
`<rangeType>`	The source range that is used for the sweep: ■ Most sensitive source range for each source level in the sweep: `AUTO` ■ Best fixed range: `BEST` (default) ■ Present source range for the entire sweep: `FIXed`
`<failAbort>`	Determines if the sweep is stopped immediately if a limit is exceeded; options are: ■ Abort the sweep if the source limit is exceeded: `ON` (default) ■ Complete the sweep if the source limit is exceeded: `OFF`

<dual>	Determines if the sweep runs from start to stop and then from stop to start: ■ Sweep from start to stop only: OFF (default) ■ Sweep from start to stop, then stop to start: ON
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it clears any existing trigger models, creates a source configuration list, and populates the trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the sweep delay, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

Example

`*RST`
`SOUR:FUNC VOLT`
`SOUR:VOLT:RANG 20`
`SENS:FUNC "CURR"`
`SENS:CURR:RANG 100e-6`
`SOUR:SWE:VOLT:LIN 0, 10, 20, 1e-3, 1, FIXED`
`INIT`
	Reset the instrument to its defaults.
	Set the source function to voltage.
	Set the source range to 20 V. Set the measure function to current with a range of 100 µA.
	Set up a linear sweep that sweeps from 0 to 10 V in 20 points with a source delay of 1 ms, a sweep count of 1, and a fixed source range.
	Start the sweep.

Also see

Sweep operation (on page 4-54)

:SOURce[1]:SWEep:<function>:LINear:STEP

This command sets up a linear source sweep configuration list and trigger model with a fixed number of steps.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>, <failAbort>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, "<bufferName>"
```

`<function>`	Voltage sweep: `VOLTage` Current sweep: `CURRent`
`<start>`	The voltage or current source level at which the sweep starts: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
`<stop>`	The voltage or current at which the sweep stops: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
`<steps>`	The step size at which the source level will change; step size must be greater than 0
`<delay>`	The delay between measurement points; default is `-1`, which enables autodelay, or a specific delay value from 50 µs to 10,000 s; or 0 for no delay
`<count>`	The number of times to run the sweep; default is `1`: ■ Infinite loop: `0` ■ Finite loop: `1` to `268435455`
`<rangeType>`	The source range that is used for the sweep: ■ Most sensitive source range for each source level in the sweep: `AUTO` ■ Best fixed range: `BEST` (default) ■ Present source range for the entire sweep: `FIXed`
`<failAbort>`	Determines if the sweep is stopped immediately if a limit is exceeded; options are: ■ Abort the sweep if the source limit is exceeded: `ON` (default) ■ Complete the sweep if the source limit is exceeded: `OFF`
`<dual>`	Determines if the sweep runs from start to stop and then from stop to start: ■ Sweep from start to stop only: `OFF` (default) ■ Sweep from start to stop, then stop to start: `ON`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

When the sweep is started, the instrument sources a specific voltage or current voltage to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it deletes the existing trigger model and creates a trigger model with a uniform series of ascending or descending voltage or current changes, called steps. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the stop level, which is calculated from the number of steps. A measurement is performed at each source step (including the start and stop levels). At this level, the instrument performs another measurement and then stops the sweep.

The instrument uses the step size parameter to determine the number of source level changes. The source level changes in equal steps from the start level to the stop level. To avoid a setting conflicts error, make sure the step size is greater than the start value and less than the stop value. To calculate the number of source-measure points in a sweep, use the following formula:

Points = [(Stop - Start) / Step] + 1

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```
*RST
SOUR:FUNC CURR
SOUR:CURR:RANGE 1
SENS:FUNC "VOLT"
SENS:VOLT:RANGE 20
SOUR:SWE:CURR:LIN:STEP -1.05, 1.05, .25, 10e-3, 1, FIXED
INIT
```

Reset the instrument to its defaults.
Set the source function to current.
Set the source range to 1 A. Set the measure function to voltage with a range of 20 V.
Set up a linear step sweep that sweeps from -1.05 A to 1.05 A in 0.25 A increments with a source delay of 1 ms, a sweep count of 1, and a fixed source range. The name of the configuration list that is created for this sweep is `CurrLinearSweep`.
Start the sweep.

Also see

Sweep operation (on page 4-54)

:SOURce[1]:SWEep:<function>:LIST

This command sets up a sweep based on a configuration list, which allows you to customize the sweep.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LIST <startIndex>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>, <failAbort>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>, <failAbort>,
   "<bufferName>"
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>, <failAbort>,
   "<bufferName>", "<configListName>"
```

`<function>`	The source function: ■ Current: `CURRent` ■ Voltage: `VOLTage`
`<startIndex>`	The index in the configuration list where the sweep starts; default is `1`
`<delay>`	The delay between measurement points; default is 0 for no delay or you can set a specific delay value from 50 µs to 10,000 s
`<count>`	The number of times to run the sweep; default is `1`: ■ Infinite loop: `0` ■ Finite loop: `1` to `268435455`
`<failAbort>`	Determines if the sweep is stopped immediately if a limit is exceeded; options are: ■ Abort the sweep if a limit is exceeded: `ON` ■ Complete the sweep even if a limit is exceeded: `OFF`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used
`<configListName>`	The name of the source configuration list that the sweep uses; this must be defined before sending this command

Details

This command allows you to set up a custom sweep, using a configuration list to specify the source levels.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

A configuration list must be created before you send this command. You can either use the `:SOURce[1]:LIST:<function>` to set up the configuration list, or you can create your own configuration list.

You cannot use a writable reading buffer to collect data from a sweep.

To run the sweep, initiate the trigger model.

Example 1

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SENS:CURR:RSEN OFF
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 1
SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5
SOUR:SWE:VOLT:LIST 1, 0.2
INIT
*WAI

TRAC:DATA? 1, 6, "defbuffer1", SOUR, READ
```

This example uses the `:SOURce[1]:LIST:<function>` command to set up the configuration list that is used by the sweep.
This example will source 1 V, 5 V, 1 V, 5 V, 1 V, 5 V and measure the resulting current at each voltage point. The time duration of each voltage point is 200 ms.

Example 2

```
SOUR:CONF:LIST:CRE "biasLevel"
SOUR:FUNC VOLT
SENS:FUNC "CURR"
SOUR:VOLT:LEV 5
SOUR:CONF:LIST:STORE "biasLevel"
SOUR:SWE:VOLT:LIST 1, .001, 1, 1, "defbuffer2", "biasLevel"
INIT
```

This example uses a user-defined configuration list.
Create a configuration list named `biasLevel`. Set the source function to 5 V and the measure function to current.
Store the configuration list.
Set up a voltage sweep that uses the configuration list, starting at index point 1 with a delay of 1 ms. The sweep is to abort if the source limit is exceeded, store data in `defbuffer2`, and use the configuration list `biasLevel`.

Also see

Configuration lists (on page 4-82)
:INITiate[:IMMediate] (on page 12-145)
:SOURce[1]:LIST:<function> (on page 12-86)
Sweep operation (on page 4-54)

:SOURce[1]:SWEep:<function>:LOG

This command sets up a logarithmic sweep for a set number of measurement points.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, "<bufferName>"
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, "<bufferName>", <asymptote>
```

`<function>`	The source function: ■ `CURRent` ■ `VOLTage`
`<start>`	The voltage or current source level at which the sweep starts: ■ Current: 1 pA to 1.05 A ■ Voltage: 1 pV to 1100 V
`<stop>`	The voltage or current at which the sweep stops: ■ Current: 1 pA to 1.05 A ■ Voltage: 1 pV to 1100 V
`<points>`	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
`<delay>`	The delay between measurement points; default is `-1`, which enables autodelay, or a specific delay value from 50 µs to 10,000 s, or 0 for no delay
`<count>`	The number of times to run the sweep; default is `1`: ■ Infinite loop: `0` ■ Finite loop: `1` to `268435455`
`<rangeType>`	The source range that is used for the sweep: ■ Most sensitive source range for each source level in the sweep: `AUTO` ■ Best fixed range: `BEST` (default) ■ Present source range for the entire sweep: `FIXed`
`<failAbort>`	Abort the sweep if the source limit is exceeded: `ON` (default) Complete the sweep if the source limit is exceeded: `OFF`

<dual>	Determines if the sweep runs from start to stop and then from stop to start: ▪ Sweep from start to stop only: OFF (default) ▪ Sweep from start to stop, then stop to start: ON
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<asymptote>	Default is 0; see **Details**

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it clears the existing trigger model and creates a new trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

The asymptote changes the inflection of the sweep curve and allows it to sweep through zero. You can use the asymptote parameter to customize the inflection and offset of the source value curve. Setting this parameter to zero provides a conventional logarithmic sweep. The asymptote value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep. The asymptote value must not be equal to or between the starting and ending values. It must be outside the range defined by the starting and ending values.

A configuration list must be created before you send this command. You can either use the :SOURce[1]:LIST:<function> to set up the configuration list, or you can create your own configuration list.

Example

```
*RST
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SENS:FUNC "CURR"
SENS:CURR:RANG 100e-6
SOUR:SWE:VOLT:LOG .1, 10, 20, 1e-3, 1, FIXED
INIT
```

Reset the instrument to its defaults.
Set the source function to voltage.
Set the source range to 20 V.
Set the measure function to current.
Set the current range to 100 µA.
Set up a log sweep that sweeps from 0.1 to 10 V in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range.
Start the sweep.

Also see

:INITiate[:IMMediate] (on page 12-145)
Sweep operation (on page 4-54)

STATus subsystem

The STATus subsystem controls the status registers of the instrument. For additional information on the status model, see Status model (on page 16-1).

:STATus:CLEar

This function clears event registers.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:CLEar
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It does not affect the Questionable Event Enable or Operation Event Enable registers.

Example

`:STATus:CLEar`	Clear the bits in the registers.

Also see

*CLS (on page 15-2)

:STATus:OPERation:CONDition?

This command reads the Operation Event Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation:CONDition?
```

Details

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page 16-15).

Example

`:STAT:OPER:COND?`	Returns the contents of the Operation Condition Register.

Also see

Operation Event Register (on page 16-7)

:STATus:OPERation:ENABle

This command sets or reads the contents of the Operation Event Enable Register of the status model.

Type	Affected by	Where saved	Default value
Command and query	STATus:PRESet	Not applicable	0

Usage

```
:STATus:OPERation:ENABle <n>
:STATus:OPERation:ENABle?
```

<n>	The status of the operation status register

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

When sending binary values, preface <n> with #b. When sending hexadecimal values, preface <n> with #h. No preface is needed when sending decimal values.

Example

`:STAT:OPER:ENAB #b0101000000000000`	Sets the 12 and 14 bits of the operation status enable register using a decimal value.
	You could also send the decimal value:
	`:STAT:OPER:ENAB 20480`
	Or the hexadecimal value:
	`:STAT:OPER:ENAB #h5000`

Also see

Operation Event Register (on page 16-7)

:STATus:OPERation[:EVENt]?

This command reads the Operation Event Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation[:EVENt]?
```

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example

`STAT:OPER?`	Returns the contents of the Operation Event Register of the status model.

Also see

Operation Event Register (on page 16-7)

:STATus:OPERation:MAP

This command allows you to map event numbers to bits in the Operation Event Registers.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation:MAP <bitNumber>, <setEvent>
:STATus:OPERation:MAP <bitNumber>, <setEvent>, <clearEvent>
:STATus:OPERation:MAP? <bitNumber>
```

`<bitNumber>`	The bit number that is mapped to an event (`0` to `14`)
`<setEvent>`	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
`<clearEvent>`	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

See Event numbers (on page 16-9) for information about event numbers.

The query requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers. When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`:STATus:OPERation:MAP 0, 4917, 4918`	When event 4917 occurs (a default buffer is 0% filled), bit 0 is set in the condition register and the event register of the Operation Event Register. When event 4918 occurs (a default buffer is 100% filled), bit 0 in the condition register is cleared.

Also see

Operation Event Register (on page 16-7)
Programmable status register sets (on page 16-4)

:STATus:PRESet

This command resets all bits in the status model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:PRESet
```

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the Service Request Enable Register (*SRE) to Standard Request Enable Register (*ESE).

Preset does not affect the event queue.

The Standard Event Status Register is not affected by this command.

Example

`STAT:PRES`	Resets the registers.

Also see

Status model (on page 16-1)

:STATus:QUEStionable:CONDition?

This command reads the Questionable Condition Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:QUEStionable:CONDition?
```

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page 16-15).

Example

`:STAT:QUES:COND?`	Reads the Questionable Condition Register.

Also see

Questionable Event Register (on page 16-6)
Understanding bit settings (on page 16-15)

:STATus:QUEStionable:ENABle

This command sets or reads the contents of the questionable event enable register of the status model.

Type	Affected by	Where saved	Default value
Command and query	STATus:PRESet	Not applicable	0

Usage

```
:STATus:QUEStionable:ENABle <n>
:STATus:QUEStionable:ENABle?
```

`<n>`	The value of the register (0 to 65535)

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register are set.

For detail on interpreting the value of a register, see Understanding bit settings (on page 16-15).

Example

`:STAT:QUES:ENAB 8` `:STAT:QUES:ENAB?`	Enable bit 4, Limit 3 Fail, when the limit test 3 failure value is exceeded. Check to see that the value was set.

Also see

Questionable Event Register (on page 16-6)

:STATus:QUEStionable:MAP

This command queries mapped event numbers or maps event numbers to bits in the event registers.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	0

Usage

```
:STATus:QUEStionable:MAP <bitNumber>, <setEvent>
:STATus:QUEStionable:MAP <bitNumber>, <setEvent>, <clearEvent>
:STATus:QUEStionable:MAP? <bitNumber>
```

`<bitNumber>`	The bit number that is mapped to an event (`0` to `14`)
`<setEvent>`	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
`<clearEvent>`	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to `0` when the event is detected.

If the event is set to zero (`0`), the bit is never set.

See Event numbers (on page 16-9) for information about event numbers.

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (`0`) indicates that the bits have not been set.

Example

`:STAT:QUES:MAP 0, 4917, 4918`	When event 4917 occurs (a default buffer is 0% filled), bit 0 is set in the condition register and the event register of the Questionable Event Register. When event 4918 occurs (a default buffer is 100% filled), bit 0 in the condition register is cleared.

Also see

Questionable Event Register (on page 16-6)

:STATus:QUEStionable[:EVENt]?

This command reads the Questionable Event Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:QUEStionable[:EVENt]?
```

Details

This query reads the contents of the questionable status event register. After sending this command and addressing the instrument to talk, a value is sent to the computer. This value indicates which bits in the appropriate register are set.

The Questionable Register can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set the Questionable Register to the sum of their decimal weights. For example, to set bits B12 and B13, set the Questionable Register to 12,288 (which is the sum of 4,096 + 8,192).

Example

`:STAT:QUES?`	Query the Questionable Register.

Also see

Questionable Event Register (on page 16-6)

SYSTem subsystem

This subsystem contains commands that affect the overall operation of the instrument, such as passwords, beepers, communications, event logs, and time.

:SYSTem:ACCess

This command contains the type of access users have to the instrument through different interfaces.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	FULL

Usage

```
:SYSTem:ACCess <permissions>
:SYSTem:ACCess?
```

`<permissions>`	The level of access that is allowed: ■ Full access for all users from all interfaces: `FULL` ■ Allows access by one remote interface at a time with login and logout required from other interfaces: `EXCLusive` ■ Allows access by one remote interface at a time with passwords required on all interfaces: `PROTected` ■ Allows access by one interface at a time (including the front panel) with passwords required on all interfaces: `LOCKout`

Details

When access is set to full, the instrument accepts commands from any interface with no login or password.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. You do not need a password with this access.

Protected access is similar to exclusive access, except that you must enter a password when logging in.

When the access is set to locked out, a password is required to change interfaces, including the front-panel interface.

Under any access type, if a script is running on one remote interface when a command comes in from another remote interface, the command is ignored and the message "FAILURE: A script is running, use ABORT to stop it" is generated.

Example

`:SYST:ACC LOCK`
`login admin`
`logout`
	Set the instrument access to locked out.
	Log into the interface using the default password.
	Log out of the interface.

Also see

:SYSTem:PASSword:NEW (on page 12-114)

:SYSTem:BEEPer[:IMMediate]

This command generates an audible tone.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:BEEPer[:IMMediate] <frequency>, <duration>
```

`<frequency>`	The frequency of the beep (20 Hz to 8000 Hz)
`<duration>`	The amount of time to play the tone (0.001 s to 100 s)

Details

You can use the beeper of the instrument to provide an audible signal at a specific frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

Using this function from a remote interface does not affect audible errors or key click settings that were made from the 2470 front panel.

Example

`:SYSTem:BEEPer 500, 1`	Beep at 500 Hz for 1 s.

Also see

None

:SYSTem:BREakdown:PROTection

This command allows you to enable the breakdown protection in situations where the current may exceed the programmed current or the limit current value due to a breakdown condition.

Type	Affected by	Where saved	Default value
Command and query	Recall setup Instrument reset Power cycle	Save setup	OFF

Usage

```
:SYSTem:BREakdown:PROTection <setting>
:SYSTem:BREakdown:PROTection?
```

`<setting>`	The breakdown protection setting: ■ Operate breakdown protection automatically (see **Details**): `AUTO` ■ Turn breakdown protection off: `OFF` ■ Turn breakdown protection on (not recommended): `ON`

Details

In some tests, the limited bandwidth of the SMU may cause current to exceed either the programmed current or the limit current value. To prevent this from occurring, you can turn on the breakdown protection function. This adds a 500 Ω resistor in series with the SMU force lead. This resistor limits, at a wide bandwidth, the breakdown current to a maximum value of V_{OUTPUT}/500.

When the breakdown protection is set for AUTO operation, the resistor is in place for the 200 V and 1000 V ranges and current ranges less than or equal to the 10 mA range. Above the 10 mA range or on lower voltage ranges, the breakdown protection resistor is automatically taken out of series with the SMU force lead.

An example of when breakdown protection is appropriate is when you are testing components to specify the DUT breakdown voltage. One method is to test the component at the specified breakdown voltage and determine if the leakage current has exceeded a threshold level. For example, if you are testing a 1000 V rated MOSFET, you can short the gate to the source and apply 1050 V from the drain to the source while measuring the actual drain current. Testing at 1050 V instead of 1000 V provides some assurance that the component both meets and exceeds the breakdown requirement by some user-specified margin. In this test, if the breakdown protection is off, you may find that at the exact moment of component breakdown, the current may exceed the limit current value. With the breakdown function on, the peak current is limited to V_{OUTPUT}/500.

A more comprehensive method of testing components to specify the DUT breakdown voltage is to measure the actual component breakdown voltage. To do this on a 1000 V rated MOSFET, you need to switch the sourcing method to current and the limit voltage to 1100 V (higher than the highest expected breakdown voltage). Sourcing the value of the breakdown current permits the SMU to measure the actual breakdown voltage of the component, which occurs precisely when the SMU reaches the programmed source current. When this operating point is reached, a voltage measurement records the actual breakdown voltage of the component, after which the small source current used to find this breakdown voltage can be reduced to zero while waiting for the next component to test. The entire test should be done quickly with the lowest possible breakdown current to limit device self-heating. When performing this test, if the breakdown protection is off, you may find that at the exact moment of component breakdown, the current may exceed the limit current value. With the breakdown function on, the peak current is limited to V_{OUTPUT}/500.

Example

```
SYST:BRE:PROT ON
```
Turn breakdown protection on.

Also see

None

:SYSTem:CLEar

This command clears the event log.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:CLEar
```

Details

This command removes all events from the event log, including entries in the front-panel event log.

Also see

:SYSTem:ERRor[:NEXT]? (on page 12-108)

:SYSTem:COMMunication:LAN:CONFigure

This command specifies the LAN configuration for the instrument.

Type	Affected by	Where saved	Default value
Command and query	Rear panel LAN reset	Nonvolatile memory	AUTO

Usage

```
:SYSTem:COMMunication:LAN:CONFigure "AUTO"
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPaddress>"
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPaddress>,<NETmask>"
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPaddress>,<NETmask>,<GATeway>"
:SYSTem:COMMunication:LAN:CONFigure?
```

AUTO	Use automatically configured LAN settings (default)
MANual	Use manually configured LAN settings
<IPaddress>	LAN IP address; must be a string specifying the IP address in dotted decimal notation; required if the mode is set to manual (default "0.0.0.0")
<NETmask>	The LAN subnet mask; must be a string in dotted decimal notation (default "255.255.255.0")
<GATeway>	The LAN default gateway; must be a string in dotted decimal notation (default "0.0.0.0")

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used. When specifying multiple parameters, do not use spaces after the commas.

The query form of the command returns the present settings in the order shown here.

Automatic:

```
AUTO,<IPaddress>,<NETmask>,<GATeway>
```

Manual:

```
MANual,<IPaddress>,<NETmask>,<GATeway>
```

Example

```
SYST:COMM:LAN:CONF "MANUAL,192.168.0.1,255.255.240.0,192.168.0.3"
SYST:COMM:LAN:CONF?
```

Set the IP address to be set manually, with the IP address set to 192.168.0.1, the subnet mask to 255.255.240.0, and the gateway address to 192.168.0.3.
Query to verify the settings. The response to the query should be:
`manual,192.168.0.1,255.255.240.0,192.168.0.3`

Also see

:SYSTem:COMMunication:LAN:MACaddress? (on page 12-107)

:SYSTem:COMMunication:LAN:MACaddress?

This command queries the LAN media access control (MAC) address.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:COMMunication:LAN:MACaddress?
```

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets.

Example

`:SYSTem:COMMunication:LAN:MACaddress?`	Returns the MAC address. For example, you might see: `08:00:11:00:00:57`

Also see

:SYSTem:COMMunication:LAN:CONFigure (on page 12-106)

:SYSTem:ERRor[:NEXT]?

This command returns the oldest unread error message from the event log and removes it from the log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor[:NEXT]?
```

Details

As error and status messages occur, they are placed in the event log. The event log is a first-in, first-out (FIFO) register that can hold up to 1000 messages.

This command returns the next entry from the event log.

This command does not affect the event log that is displayed on the front panel.

If there are no entries in the event log, the following message is returned:

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

This command returns only error messages from the event log. To return information and warning messages, see `:SYSTem:EVENtlog:NEXT?`.

Note that if you have used `:SYSTem:ERRor[:NEXT]?` to check events, `:SYSTem:EVENtlog:NEXT?` shows the next event item after the last error that was returned by `:SYSTem:ERRor[:NEXT]?` You will not see warnings or information event log items that occurred before you used `:SYSTem:ERRor[:NEXT]?`

Example

`SYST:ERR:NEXT?`	Returns information on the next error in the event log. For example, if you sent a command without a parameter, the return is: `-109,"Missing parameter;1;2017/05/06 12:57:04.484"`

Also see

:SYSTem:EVENtlog:NEXT? (on page 12-110)

:SYSTem:ERRor:CODE[:NEXT]?

This command reads the oldest error code.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor:CODE[:NEXT]?
```

Details

This command returns the numeric code of the next error in the event log. The error is cleared from the queue after being read.

This command returns only error messages from the event log. To return information and warning messages, see `:SYSTem:EVENtlog:NEXT?`

Example

`SYST:ERR:CODE?`	Returns the error code of the next error in the event log. For example, if error -222, Parameter data out of range error, occurred, the output is: `-222`

Also see

:SYSTem:EVENtlog:NEXT? (on page 12-110)

:SYSTem:ERRor:COUNt?

This command returns the number of errors in the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor:COUNt?
```

Details

This command does not return other types of events, such as information messages. To return other types of events, use `:SYSTem:EVENtlog:COUNt?`

This command does not clear the errors from the event log.

Example

`SYST:ERR:COUN?`	If there are five errors in the event log, the output is: 5

Also see

:SYSTem:EVENtlog:COUNt? (on page 12-109)

:SYSTem:EVENtlog:COUNt?

This command returns the number of unread events in the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:COUNt?
:SYSTem:EVENtlog:COUNt? <eventType>
:SYSTem:EVENtlog:COUNt? <eventType>, <eventType>
:SYSTem:EVENtlog:COUNt? <eventType>, <eventType>, <eventType>
```

`<eventType>`	Limits the list of event log entries to specific types; set to: ■ Returns the number of errors: `ERRor` ■ Returns the number of warnings: `WARNing` ■ Returns the number of informational messages: `INFormational` ■ Returns all events: `ALL`

Details

A count finds the number of unread events in the event log. You can specify the event types to return or return the count for all events.

This command reports the number of events that have occurred since the command was last sent or since the event log was last cleared.

Example

`:SYST:EVEN:COUN? ERR`	Displays the present number of errors in the instrument event log. If there are three errors in the event log, output is: 3

Also see

:SYSTem:CLEar (on page 12-106)
:SYSTem:EVENtlog:NEXT? (on page 12-110)
:SYSTem:EVENtlog:SAVE (on page 12-112)

:SYSTem:EVENtlog:NEXT?

This command returns the oldest unread event message from the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:NEXT?
:SYSTem:EVENtlog:NEXT? <eventType>
:SYSTem:EVENtlog:NEXT? <eventType>, <eventType>
:SYSTem:EVENtlog:NEXT? <eventType>, <eventType>, <eventType>
```

`<eventType>`	Limits the event log entries that are returned to specific types; set to: ■ Returns only the next error: `ERRor` ■ Returns only the next warning: `WARNing` ■ Returns only the next informational message: `INFormational` ■ Returns any event: `ALL`

Details

When an event occurs on the instrument, it is placed in the event log. The `:SYSTem:EVENtlog:NEXT?` command retrieves an unread event from the event log. Once an event is read, it can no longer be accessed remotely. However, it can be viewed on the front panel.

To read multiple commands, execute this command multiple times.

If there are no entries in the event log, the following is returned:

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

If the event type is not defined, an event of any type is returned.

Note that if you have used `:SYSTem:ERRor[:NEXT]?` to check events, `:SYSTem:EVENtlog:NEXT?` shows the next event item after the last error that was returned by `:SYSTem:ERRor[:NEXT]?` You will not see warnings or information event log items that occurred before you used `:SYSTem:ERRor[:NEXT]?`

If the event type is not defined, an event of any type is returned.

The information that is returned is in the order:

`<eventNumber>, <message>, <eventType>, <timeSeconds>, <timeNanoSeconds>`

`<eventNumber>`	The event number
`<message>`	A description of the event
`<eventType>`	The type of event: ■ Error only: 1 ■ Warning only: 2 ■ Information only: 4
`<timeSeconds>`	The seconds portion of the time when the event occurred
`<timeNanoSeconds>`	The fractional seconds portion of the time when the event occurred

Example

`SYST:EVEN:NEXT?`	Returns information on the next event in the event log. For example, if you sent a command without a parameter, the return is: `-109,"Missing parameter;1;2017/05/06 12:55:33.648"`

Also see

:SYSTem:CLEar (on page 12-106)
:SYSTem:EVENtlog:SAVE (on page 12-112)

:SYSTem:EVENtlog:POST

This command allows you to post your own text to the event log.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:POST "<message>"
:SYSTem:EVENtlog:POST "<message>", <eventType>
```

`<message>`	A string that contains the message that will be associated with this event
`<eventType>`	The type of event that is generated; set to: ■ The error type: `ERRor` ■ The warning type: `WARNing` ■ The informational type: `INFormational` (default)

Details

You can use this command to create your own event log entries and assign a severity level to them. This can be useful for debugging and status reporting.

From the front panel, you must set the Log Warnings and Log Information options on to have the custom warning and information events placed into the event log.

Example

`*CLS`
`SYST:EVEN:POST "my error", INF`
`SYST:EVEN:NEXT?`
	Clear the event log.
	Post an error named `my error`.
	Output:
	`1003,"User: my error;4,1400469179,431599191"`

Also see

Using the event log (on page 3-50)

:SYSTem:EVENtlog:SAVE

This command saves the event log to a file on a USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:SAVE "<filename>"
:SYSTem:EVENtlog:SAVE "<filename>", <eventType>
```

`<filename>`	A string that holds the name of the file to be saved
`<eventType>`	Limits the event log entries that are saved to specific types; set to: ■ `ERRor`: Saves only error entries ■ `WARNing`: Saves only warning entries ■ `INFormational`: Saves only informational entries ■ `ALL`: Saves all event log entries (default)

Details

This command saves all event log entries to a USB flash drive.

If you do not define an event type, the instrument saves all event log entries.

The extension `.csv` is automatically added to the file name.

Example

`SYST:EVEN:SAVE "/usb1/July_error_log", ERR`	Saves the error events in the event log to a file on the USB flash drive named `July_error_log.csv`.

Also see

:SYSTem:CLEar (on page 12-106)

:SYSTem:GPIB:ADDRess

This command contains the GPIB address.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	18

Usage

```
:SYSTem:GPIB:ADDRess <n>
:SYSTem:GPIB:ADDRess?
```

<n>	The GPIB address of the instrument (1 to 30)

Details

The address can be set to any address value from 1 to 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow sufficient time for the command to be processed before attempting to communicate with the instrument again.

*RST does not affect the GPIB address.

Example

`:SYSTem:GPIB:ADDRess 26` `:SYSTem:GPIB:ADDRess?`	Sets the GPIB address and reads the address. Output: `2.600000e+01`

Also see

GPIB setup (on page 2-9)

:SYSTem:LFRequency?

This query contains the power line frequency setting that is used for NPLC calculations.

Type	Affected by	Where saved	Default value
Query only	Power cycle	Not applicable	Not applicable

Usage

```
:SYSTem:LFRequency?
```

Details

The instrument automatically detects the power line frequency when the instrument is powered on. Power line frequency can be 50 Hz or 60 Hz.

Example

`:SYST:LFR?`	Check the line frequency.

Also see

None

:SYSTem:PASSword:NEW

This command stores the instrument password.

Type	Affected by	Where saved	Default value
Command only	Rear-panel LAN reset	Nonvolatile memory	admin

Usage

```
:SYSTem:PASSword:NEW "<password>"
```

`<password>`	A string that contains the instrument password (maximum 30 characters)

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

If you forget the password, you can reset the password to the default:

1. On the front panel, press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **Password Reset**.

You can also reset the password and the LAN settings from the rear panel by inserting a straightened paper clip into hole below LAN RESET.

Example

`SYST:PASS:NEW "N3wpa55w0rd"`	Change the password of the instrument to `N3wpa55w0rd`.

Also see

:SYSTem:ACCess (on page 12-103)

:SYSTem:POSetup

This command selects the defaults that are used when you power on the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	RST

Usage

```
:SYSTem:POSetup <name>
:SYSTem:POSetup?
```

<name>	Which setup to restore when you power on the instrument: ■ Power on to *RST defaults: RST ■ Stored setup 0: SAV0 ■ Stored setup 1: SAV1 ■ Stored setup 2: SAV2 ■ Stored setup 3: SAV3 ■ Stored setup 4: SAV4

Details

When you select RST, the instrument restores settings to their default values when the instrument is powered on.

When you select a SAV option, the settings in the selected saved setup are applied when the instrument is powered on. The settings are saved using the *SAV command.

Example

SYST:POS SAV1	Set the instrument to restore the settings that are saved in the stored setup 1 when the instrument is powered on.

Also see

*SAV (on page 12-9)

:SYSTem:TIME

This command sets the absolute time of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	See **Details**

Usage

```
:SYSTem:TIME <year>, <month>, <day>, <hour>, <minute>, <second>
:SYSTem:TIME <hour>, <minute>, <second>
:SYSTem:TIME?
:SYSTem:TIME? 1
```

`<year>`	Year; must be more than 1970
`<month>`	Month (1 to 12)
`<day>`	Day (1 to 31)
`<hour>`	Hour in 24-hour time format (0 to 23)
`<minute>`	Minute (0 to 59)
`<second>`	Second (0 to 59)

Details

When queried without a parameter, this command returns the present timestamp value in seconds since January 1, 1970 to the nearest second.

If you query with `1`, this command returns the present timestamp in the format:

`<weekday> <month> <day> <hour>:<minute>:<second> <year>`

Where `<weekday>` is the day of the week.

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

`syst:time 2018, 2, 15, 11, 30, 30` `syst:time? 1`	Set the system time to February 15, 2018 at 11:30:30 and confirm setting. Output: `Thu Feb 15 11:30:35 2018`

Also see

None

:SYSTem:VERSion?

Query the present SCPI version.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:VERSion?
```

Details

This query command returns the SCPI version.

Example

`SYSTem:VERSion?`	Query the version. An example of a return is: `1996.0`

Also see

None

TRACe subsystem

The TRACe subsystem contains commands that control the reading buffers.

:TRACe:ACTual?

This command contains the number of readings in the specified reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:ACTual?
:TRACe:ACTual? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This command returns the number of readings stored in the buffer.

Example

`TRACe:MAKE "testData", 200`
`COUN 10`
`MEASure:CURRent? "testData"`
	Creates 200 element reading buffer named `testData`.
	Set the measurement count to `10`.
	Set the measurement function to current. Make readings and store the readings in `testData`. Returns the tenth measurement reading after taking all ten readings.

`:TRACe:ACTual?`	Returns the number of readings in `defbuffer1`. Example output: `850`
`:TRACe:ACTual? "testData"`	Returns the number of readings in the buffer `testData`. Example output: `10`

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)

:TRACe:ACTual:END?

This command indicates the last index in a reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:ACTual:END?
:TRACe:ACTual:END? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

Use this command to find the ending index in a reading buffer.

Example

```
TRACe:MAKE "test1", 100
COUNt 6
MEASure:CURRent? "test1"
:TRACe:ACTual:STARt? "test1" ; END? "test1"
MEASure:CURRent? "test1"
:TRACe:ACTual:STARt? "test1" ; END? "test1"
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index and end index of `test1`.
Output: `1;6`
Make six more measurements and store them in buffer `test1`.
Get the start and end index of `test1`.
Output: `1;12`

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:ACTual:STARt? (on page 12-119)
:TRACe:MAKE (on page 12-126)

:TRACe:ACTual:STARt?

This command indicates the starting index in a reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:ACTual:STARt?
:TRACe:ACTual:STARt? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

Use this command to find the starting index in a reading buffer.

Example

```
TRACe:MAKE "test1", 100
COUNt 6
MEASure:CURRent? "test1"
:TRACe:ACTual:STARt? "test1" ; END? "test1"
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index and end index of `test1`.
Output: `1;6`

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:ACTual:END? (on page 12-118)
:TRACe:MAKE (on page 12-126)

:TRACe:CLEar

This command clears all readings and statistics from the specified buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:CLEar
:TRACe:CLEar "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Example

`TRACe:MAKE "testData", 200`
`MEASure:RESistance? "testData"`
`TRACe:ACTual? "testData"`
`TRACe:CLEar "testData"`
`TRACe:ACTual? "testData"`
	Create user-defined buffer named `testData`.
	Make a measurement and store it in `testData` and return the last reading measured.
	Verify that there is data in `testData` buffer.
	Output:
	`1`
	Clear `testData` buffer.
	Verify that `testData` is empty.
	Output:
	`0`

`TRACe:CLEar`
`TRACe:CLEar "defbuffer1"`
`TRACe:CLEar "defbuffer2"`
	Clear the default buffer. This command clears `defbuffer1`.
	Clear `defbuffer1`. Specify default buffer by name.
	Clear `defbuffer2`. Specify default buffer by name.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)

:TRACe:DATA?

This command returns specified data elements from a specified reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:DATA? <startIndex>, <endIndex>
:TRACe:DATA? <startIndex>, <endIndex>, "<bufferName>"
:TRACe:DATA? <startIndex>, <endIndex>, "<bufferName>", <bufferElements>
```

`<startIndex>`	Beginning index of the buffer to return; must be 1 or greater
`<endIndex>`	Ending index of the buffer to return
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used
`<bufferElements>`	A list of elements in the buffer to print; if nothing is specified, `READing` is used; see **Details** for the list of options for buffer elements; a maximum of 14 comma-delimited buffer elements may be specified

Details

The output of `:TRACe:DATA?` is affected by the data format selected by `:FORMat[:DATA]`. If you set `FORMat[:DATA]` to REAL or SREAL, you will have fewer options for buffer elements. The only buffer elements available are `READing`, `RELative`, `SOURce`, and `EXTRa`. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

NOTE

To change the number of digits returned in a remote command reading, use the `:FORMat:ASCii:PRECision` command.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include any or all of the buffer elements listed below in a single list. You can repeat elements if the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for <bufferElements> are described in the following table.

Option	Description
DATE	The date when the data point was measured
FORMatted	The measured value as it appears on the front panel
FRACtional	The fractional seconds for the data point when the data point was measured
READing	The measurement reading based on the [:SENSe[1]]:FUNCtion[:ON] setting; if no buffer elements are defined, this option is used
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
SOURce	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 12-85))
SOURFORMatted	The source value as it appears on the display
SOURSTATus	The status information associated with sourcing. The values returned indicate the status of the following conditions: ■ Overvoltage protection was active ■ Measured source value was read ■ Overtemperature condition existed ■ Source function level was limited ■ Four-wire sense was used ■ Output was on
SOURUNIT	The unit of value associated with the source value
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time for the data point
TSTamp	The timestamp for the data point
UNIT	The unit of measure associated with the measurement

The STATus buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the 2470, this will always be 0 (main)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
TRAC:MAKE "buf100", 100
TRIGger:LOAD "SimpleLoop", 5, 0, "buf100"
SOUR:VOLT 0.35
INIT
*WAI
TRAC:DATA? 1, 5, "buf100", READ, SOUR, REL
TRAC:DATA? 1, 5, "buf100", READ, REL
TRAC:DATA? 1, 5, "buf100", REL
TRAC:DATA? 1, 3, "buf100"
```

Create a buffer called `buf100` with a maximum size of 100.

Set the instrument to configure the trigger model to loop, make five readings with no delay, and store the readings in the `buf100` reading buffer.

Set the source level for voltage to `0.35`.

Initiate the trigger model and wait for the trigger model to complete. The trigger model will make five readings and store them in `buf100`.

Read the five data points, reading, programmed source, and relative time for each point.

Output:

```
-0.000000,0.350000,0.000000;
    -0.000000,0.350000,0.266978,
    -0.000000,0.350000,0.443087,
    -0.000000,0.350000,0.704459,
    -0.000000,0.350000,0.881419
```

Read five data points and include the reading and relative time for each data point.

Output:

```
-0.000000,0.000000,
    -0.000000,0.266978,
    -0.000000,0.443087,
    -0.000000,0.704459,
    -0.000000,0.881419
```

Read five data points and include relative time for each data point.

Output:

```
0.000000,0.266978;0.443087,
    0.704459,0.881419
```

Returns the first three reading values from `buf100` reading buffer

Output:

```
-0.000000,-0.000000,-0.000000
```

Also see

:FORMat[:DATA] (on page 12-36)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)

:TRACe:DELete

This command deletes a user-defined reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:DELete "<bufferName>"
```

`<bufferName>`	A string that contains the name of the user-defined reading buffer to delete

Details

You cannot delete the default reading buffers, `defbuffer1` and `defbuffer2`.

Example

```
TRAC:DEL "testData"
```

Delete the `testData` buffer.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)

:TRACe:FILL:MODE

This command determines if a reading buffer is filled continuously or is filled once and stops.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1: CONT defbuffer2: CONT User-defined buffers: ONCE

Usage

```
:TRACe:FILL:MODE <fillType>
:TRACe:FILL:MODE <fillType>, "<bufferName>"
:TRACe:FILL:MODE?
:TRACe:FILL:MODE? "<bufferName>"
```

`<fillType>`	Fill the buffer continuously: `CONTinuous` Fill the buffer, then stop: `ONCE`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

When a reading buffer is set to fill once, no data is overwritten in the buffer. When the buffer is filled, no more data is stored in that buffer and new readings are discarded.

When a reading buffer is set to fill continuously, the oldest data is overwritten by the newest data after the buffer fills.

When you change the fill mode of a buffer, any data in the buffer is cleared.

Example

`TRACe:MAKE "testData", 100`
 	
`TRACe:FILL:MODE? "testData"`
`TRACe:FILL:MODE CONT, "testData"`
`TRACe:FILL:MODE? "testData"`
 	
`TRACe:FILL:MODE?`
	Create a user-defined reading buffer named `testData` with a capacity of 100 measurements.
	Query the fill mode setting for `testData`.
	Output:
	`ONCE`
	Set `testData` fill mode to continuous.
	Query the fill mode setting for `testData`.
	Output:
	`CONT`
	Query the fill mode setting for `defbuffer1`.
	Output:
	`CONT`

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:CLEar (on page 12-120)

:TRACe:LOG:STATe

This command indicates if information events are logged when the specified reading buffer is at 0% or 100% filled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1: ON (1) defbuffer2: ON (1) User-created buffer: OFF (0)

Usage

```
:TRACe:LOG:STATe <logState>
:TRACe:LOG:STATe <logState>, "<bufferName>"
:TRACe:LOG:STATe?
:TRACe:LOG:STATe? "<bufferName>"
```

`<logState>`	Do not log information events: `OFF` or `0` Log information events: `ON` or `1`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

If this is set to on, when the reading buffer is cleared (0% filled) or full (100% filled), an event is logged in the event log. If this is set to off, reading buffer status is not reported in the event log.

Example

`TRACe:LOG:STATe?`	Query the log state of `defbuffer1`. Output: `1` Indicates that the log state is on.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
Using the event log (on page 3-50)

:TRACe:MAKE

This command creates a user-defined reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:MAKE "<bufferName>", <bufferSize>
:TRACe:MAKE "<bufferName>", <bufferSize>, <bufferStyle>
```

`<bufferName>`	A user-supplied string that indicates the name of the buffer
`<bufferSize>`	A number that indicates the maximum number of readings that can be stored in `<bufferName>`; minimum is `10`; set to `0` to maximize the buffer size (see **Details**)
`<bufferStyle>`	The type of reading buffer to create: ■ Store readings with reduced accuracy (6.5 digits) with no formatting information, 1 µs accurate timestamp: `COMPact` ■ Store readings with full accuracy with formatting: `STANdard` (default) ■ Store the same information as standard, plus additional information: `FULL` ■ Store external reading buffer data: `WRITable` ■ Store external reading buffer data with two reading values: `FULLWRITable`

Details

The buffer name for a user-defined reading buffer cannot be `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

If you create a reading buffer that has the same name as an existing user-defined buffer, the event message 1115, "Parameter error: TRACe:MAKE cannot take an existing reading buffer name," is generated.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you create becomes the active buffer.

If you select `0`, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The default fill mode of a user-defined buffer is once. You can change it to continuous later.

Once the buffer style is selected, it cannot be changed.

Once you store the first reading in a compact buffer, you cannot change certain measurement settings, including range, display digits, and units; you must clear the buffer first.

Not all remote commands are compatible with the compact, writable, and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

Writable reading buffers are used to bring external data into the instrument. You cannot assign them to collect data from the instrument.

You can change the buffer capacity for an existing buffer through the front panel or by using the `:TRACe:POINts` command.

Example 1

`TRACe:MAKE "capTrace", 200, WRITable`	Create a 200-element writable reading buffer named `capTrace`.

Example 2

`TRACe:MAKE "bufferVolts", 100`
`TRACe:POINts? "bufferVolts"`
 	
`TRACe:DELete "bufferVolts"`
`TRACe:MAKE "bufferVolts", 1000`
`TRACe:POINts?`
	Create a buffer named `bufferVolts` to store 100 readings.
	Query the size of `bufferVolts`.
	Output:
	`100`
	Delete the buffer named `bufferVolts`.
	Make a new buffer named `bufferVolts` to store 1000 readings.
	Query the size of `bufferVolts` again to verify it can store 1000 readings.
	Output:
	`1000`

Example 3

`TRACe:POINts 5000, "bufferVolts"` `TRACe:POINts?`	Resize an existing buffer named `bufferVolts` to store 5000 readings. Query the size of `bufferVolts` to verify it can store 5000 readings. Output: `5000`

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:FILL:MODE (on page 12-124)
:TRACe:POINts (on page 12-130)
:TRACe:WRITe:FORMat (on page 12-141)
:TRACe:WRITe:READing (on page 12-143)

:TRACe:MATH

This command allows you to run a mathematical expression on a measurement. The expression is applied when the measurement is placed in the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Instrument reset Power cycle	Not saved	NONE

Usage

```
:TRACe:MATH "<bufferName>", <units>, ADD
:TRACe:MATH "<bufferName>", <units>, AVERage
:TRACe:MATH "<bufferName>", <units>, DIVide
:TRACe:MATH "<bufferName>", <units>, EXPonent
:TRACe:MATH "<bufferName>", <units>, LOG10
:TRACe:MATH "<bufferName>", <units>, MULTiply
:TRACe:MATH "<bufferName>", <units>, NONE
:TRACe:MATH "<bufferName>", <units>, POLY, <constant0>, <constant1>, <constant2>,
   <constant3>, <constant4>, <constant5>
:TRACe:MATH "<bufferName>", <units>, POWer, <constant0>
:TRACe:MATH "<bufferName>", <units>, RATE
:TRACe:MATH "<bufferName>", <units>, RECiprocal
:TRACe:MATH "<bufferName>", <units>, SQRoot
:TRACe:MATH "<bufferName>", <units>, SUBtract
```

<bufferName>	String that contains the name of the reading buffer; must be set to the style FULL
<units>	The units to be applied to the value generated by the expression: ■ DC current: AMP ■ AC current: AMPAC ■ Celsius: CELSius ■ Custom unit 1: CUSTOM1 ■ Custom unit 2: CUSTOM2 ■ Custom unit 3: CUSTOM3 ■ DAC (voltage): DAC ■ Decibel-milliwatts: DBM ■ Decibels: DECibel ■ Digital I/O: DIO ■ Fahrenheit: FAHRenheit ■ Capacitance: FARad ■ Frequency: HERTz ■ Kelvin: KELVin ■ No unit: NONE ■ Resistance: OHM ■ Percent: PERCent ■ DC voltage ratio: RATio ■ Reciprocal: RECiprocal ■ Period: SECond ■ Totalizer: TOT ■ DC voltage: VOLT ■ AC voltage: VOLTAC ■ Power: WATT
<constant0>	The constant to be used for c0 in the expression
<constant1>	The constant to be used for c1 in the expression
<constant2>	The constant to be used for c2 in the expression
<constant3>	The constant to be used for c3 in the expression
<constant4>	The constant to be used for c4 in the expression
<constant5>	The constant to be used for c5 in the expression

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style FULL. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading
- a = previous reading
- t = timestamp of the reading
- c = constant

Expression	<expression>	Formula
No math applied	`NONE`	Not applicable
Add	`ADD`	$r + a$
Average	`AVERage`	$\frac{(r+a)}{2}$
Divide	`DIVide`	$\frac{r}{a}$
Exponent	`EXPonent`	10^r
Log10	`LOG10`	$\log_{10} r$
Multiply	`MULTiply`	$r * a$
Polynomial	`POLY`	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	`POWer`	r^c
Rate of change	`RATE`	$\frac{(r-r_{-1})}{(t - t_{-1})}$
Reciprocal	`RECiprocal`	$\frac{1}{r}$
Square Root	`SQRoot`	$\sqrt{r}$
Subtract	`SUBtract`	$r – a$

Example

```
*RST
TRAC:MAKE "expressions", 100, FULL
SENS:FUNC "VOLT"
TRACe:MATH "expressions", VOLT, ADD
COUN 10
READ? "expressions"
TRAC:DATA? 1, 10, "expressions", READ, EXTR
DISP:SCR READ
```

Instrument has terminals set to FRONT.
Reset the instrument.
Make a buffer named `expressions`, set to store 100 readings with a style of FULL.
Set the measure function to voltage.
Set up buffer math, using a unit of V, that adds the present and previous readings.
Make a reading and store it in the `expressions` buffer.
Read the data in buffer indexes 1 to 10, including the readings and the values generated by the expression.
Display the reading table on the front panel of the instrument.

Also see

:TRACe:UNIT (on page 12-139)

:TRACe:POINts

This command sets the number of readings a buffer can store.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:POINts <newSize>
:TRACe:POINts <newSize>, "<bufferName>"
:TRACe:POINts?
:TRACe:POINts? "<bufferName>"
```

`<newSize>`	The new size for the buffer; set to `0` to maximize the buffer size (see **Details**)
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This command allows you to change or view how many readings a buffer can store. Changing the size of a buffer will cause any existing data in the buffer to be lost.

If you select `0`, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The overall capacity of all buffers stored in the instrument can be up to 4,500,000 readings for standard reading buffers and 20,000,000 for compact reading buffers.

For more information about buffer capacity, see Setting reading buffer capacity (on page 6-9).

Example

`TRACe:MAKE "testData", 100`
`TRACe:POINts 300, "testData"`
`TRACe:POINts? "testData"`
 	
`TRACe:POINts?`
	Create a user-defined reading buffer named `testData` with a capacity of 100 measurements.
	Change the buffer capacity to 300.
	Query the capacity of `testData`.
	Output:
	`300`
	Query the capacity of the default buffer.
	Output:
	`10000`

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)

:TRACe:SAVE

This command saves data from the specified reading buffer to a USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:SAVE "<fileName>"
:TRACe:SAVE "<fileName>", "<bufferName>"
:TRACe:SAVE "<fileName>", "<bufferName>", <what>
:TRACe:SAVE "<fileName>", "<bufferName>", <what>, <start>, <end>
```

`<fileName>`	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used
`<what>`	Defines which information is saved in the file on the USB flash drive: ■ All information: `ALL` ■ Dates, times, and fractional seconds are saved; the default value: `FORMat` ■ Relative timestamps are saved: `RELative` ■ Seconds and fractional seconds are saved: `RAW` ■ Timestamps are saved: `STAMp` ■ Standard set of data: `STANdard` ■ Brief set of data (reading and relative timestamp only): `BRIEF` ■ Extra data: `EXTRa`
`<start>`	Defines the starting point in the buffer to start saving data
`<end>`	Defines the ending point in the buffer to stop saving data

Details

The file name must specify the full path (including `/usb1/`). If included, the file extension must be set to `.csv`. If no file extension is specified, `.csv` is added.

The 2470 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing CSV files on the flash drive.

Example

```
TRACe:MAKE "MyBuffer", 100
SENSe:COUNt 5
MEASure:CURRent:DC? "MyBuffer"
TRACe:DATA? 1, 5, "MyBuffer", READ, REL, SOUR
TRACe:SAVE "/usb1/myData.csv", "MyBuffer"
TRACe:SAVE "/usb1/myDataRel.csv", "MyBuffer", REL
```

Create a buffer called `MyBuffer` with a maximum size of 100.
Make five readings for each measurement request and return the data.
Make the measurements.
Read the reading, relative timestamp, and source value for each point from 1 to 5.
Output:

```
-0.000000,0.000000,
   0.000000,-0.000000,
   0.301759,0.000000,
   -0.000000,0.579068,
   0.000000,-0.000000,
   0.884302,0.000000,
   -0.000000,1.157444,
   0.000000
```

Save all reading and default time information from a buffer named `MyBuffer` to a file named `myData.csv` on the USB flash drive.
Save all readings and relative timestamps from `MyBuffer` to a file named `myDataRel.csv` on the USB flash drive.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:SAVE:APPend (on page 12-133)

:TRACe:SAVE:APPend

This command appends data from the reading buffer to a file on the USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:SAVE:APPend "<fileName>"
:TRACe:SAVE:APPend "<fileName>", "<bufferName>"
:TRACe:SAVE:APPend "<fileName>", "<bufferName>", <timeFormat>
:TRACe:SAVE:APPend "<fileName>", "<bufferName>", <timeFormat>, <start>, <end>
```

`<fileName>`	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used
`<timeFormat>`	Indicates how date and time information from the buffer is saved in the file on the USB flash drive; the values are: ■ All information: `ALL` ■ Dates, times, and fractional seconds are saved; the default value: `FORMat` ■ Relative timestamps are saved: `RELative` ■ Seconds and fractional seconds are saved: `RAW` ■ Timestamps are saved: `STAMp` ■ Standard set of data: `STANdard` ■ Brief set of data (reading and relative timestamp only): `BRIEF` ■ Extra data: `EXTRa`
`<start>`	Defines the starting point in the buffer to start saving data
`<end>`	Defines the ending point in the buffer to stop saving data

Details

If the file you specify does not exist on the USB flash drive, this command creates the file.

For options that save more than one item of time information, each item is comma-delimited. For example, the default format is date, time, and fractional seconds for each reading.

The file extension `.csv` is appended to the file name if necessary. Any file extension other than `.csv` generates an error.

The index column entry in the `.csv` file starts at 1 for each append operation.

Example

```
TRACe:MAKE "testData", 100
SENSe:COUNt 5
MEASure:CURRent:DC? "testData", READ, REL, SOUR
TRACe:SAVE "/usb1/myData5.csv", "testData"
TRACe:CLEAr
MEASure:CURRent:DC?
TRACe:SAVE:APPend "/usb1/myData5.csv", "defbuffer1"
MEASure:CURRent:DC? "testData"
TRACe:SAVE:APPend "/usb1/myData5.csv", "testData", RAW, 6, 10
```

Create a buffer called `testData`.

Make 5 readings and return the fifth point, which will contain the reading, relative timestamp, and source value. Store the buffer data in the `myData5.csv` file.

Clear `defbuffer1`.

Make 5 readings, store them in `defbuffer1`, and return the fifth reading.

Append all the readings stored in `defbuffer1` to the `myData5.csv` file.

Take 5 more readings, store them in `testData`, and return the fifth reading.

Append all the readings stored in positions 6 through 10 `testData` to the `myData5.csv` file using raw timestamps.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)

:TRACe:STATistics:AVERage?

This command returns the average of all readings in the buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:AVERage?
:TRACe:STATistics:AVERage? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This command returns the average reading calculated from all the readings in the specified reading buffer.

When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make.

Example

`TRACe:STAT:AVERage?`	Returns the average reading for the readings in the default buffer `defbuffer1`.
`TRACe:STAT:AVERage? "testData"`	Returns the average reading for the readings in the user-defined buffer `testData`.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:STATistics:CLEar (on page 12-135)
:TRACe:STATistics:MAXimum? (on page 12-136)
:TRACe:STATistics:MINimum? (on page 12-136)
:TRACe:STATistics:PK2Pk? (on page 12-137)
:TRACe:STATistics:STDDev? (on page 12-138)

:TRACe:STATistics:CLEar

This command clears the statistical information associated with the specified buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:CLEar
:TRACe:STATistics:CLEar "<bufferName>"
```

`<bufferName>`	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; if no buffer is defined, clears the statistics from `defbuffer1`

Details

This command clears the statistics without clearing the readings.

Example

`TRACe:STATistics:CLEar`	Clear all statistics in `defbuffer1`.
`TRACe:STATistics:CLEar "testData"`	Clears all statistics in a user-defined buffer named `testData`.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:STATistics:AVERage? (on page 12-134)
:TRACe:STATistics:MAXimum? (on page 12-136)
:TRACe:STATistics:MINimum? (on page 12-136)
:TRACe:STATistics:PK2Pk? (on page 12-137)
:TRACe:STATistics:STDDev? (on page 12-138)

:TRACe:STATistics:MAXimum?

This command returns the maximum reading value in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:MAXimum?
:TRACe:STATistics:MAXimum? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Example

`TRACe:STAT:MAXimum?`	Returns the maximum reading value in the default buffer, `defbuffer1`.
`TRACe:STAT:MAXimum? "testData"`	Returns the maximum reading value in the user-defined buffer `testData`.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:STATistics:AVERage? (on page 12-134)
:TRACe:STATistics:CLEar (on page 12-135)
:TRACe:STATistics:MINimum? (on page 12-136)
:TRACe:STATistics:PK2Pk? (on page 12-137)
:TRACe:STATistics:STDDev? (on page 12-138)

:TRACe:STATistics:MINimum?

This command returns the minimum reading value in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:MINimum?
:TRACe:STATistics:MINimum? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Example

`TRACe:STAT:MINimum?`	Returns the minimum reading value in the default buffer `defbuffer1`.
`TRACe:STAT:MINimum? "testData"`	Returns the minimum reading value in the user-defined buffer `testData`.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:STATistics:AVERage? (on page 12-134)
:TRACe:STATistics:CLEar (on page 12-135)
:TRACe:STATistics:MAXimum? (on page 12-136)
:TRACe:STATistics:PK2Pk? (on page 12-137)
:TRACe:STATistics:STDDev? (on page 12-138)

:TRACe:STATistics:PK2Pk?

This command returns the peak-to-peak value of all readings in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:PK2Pk?
:TRACe:STATistics:PK2Pk? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Example

`TRACe:STAT:PK2Pk?`	Returns the peak-to-peak reading value in the default buffer `defbuffer1`.
`TRACe:STAT:PK2Pk? "testData"`	Returns the peak-to-peak reading value in the user-defined buffer `testData`.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:STATistics:AVERage? (on page 12-134)
:TRACe:STATistics:CLEar (on page 12-135)
:TRACe:STATistics:MAXimum? (on page 12-136)
:TRACe:STATistics:MINimum? (on page 12-136)
:TRACe:STATistics:STDDev? (on page 12-138)

:TRACe:STATistics:STDDev?

This command returns the standard deviation of all readings in the buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:STDDev?
:TRACe:STATistics:STDDev? "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Example

`TRACe:STAT:STDDev?`	Returns the standard deviation of the readings in the default buffer `defbuffer1`.
`TRACe:STAT:STDDev? "testData"`	Returns the standard deviation of the readings in the user-defined buffer `testData`.

Also see

Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
:TRACe:MAKE (on page 12-126)
:TRACe:STATistics:CLEar (on page 12-135)
:TRACe:STATistics:MAXimum? (on page 12-136)
:TRACe:STATistics:MINimum? (on page 12-136)
:TRACe:STATistics:PK2Pk? (on page 12-137)

:TRACe:TRIGger

This command makes readings using the active measure function and stores them in a reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:TRIGger
:TRACe:TRIGger "<bufferName>"
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

A measure function must be selected before sending this command.

This command makes the number of measurements that is set by the count command.

Example

`TRACe:MAKE "MyBuffer", 100`
`COUN 5`
`TRACe:TRIG "MyBuffer"`
`TRACe:DATA? 1, 5, "MyBuffer", rel`
	Create a buffer called `MyBuffer` with a maximum size of 100.
	Make readings and store them in `MyBuffer`.
	Recall the relative time when the data points were measured for the first five readings in the buffer.
	Example output:
	`0.000000,0.408402,0.816757,1.208823,1.617529`

Also see

[:SENSe[1]]:COUNt (on page 12-67)
[:SENSe[1]]:FUNCtion[:ON] (on page 12-68)
:TRACe:DATA? (on page 12-121)
:TRACe:MAKE (on page 12-126)

:TRACe:UNIT

This command allows you to create up to three custom units of measure for use in buffers.

Type	Affected by	Where saved	Default value
Command only	Power cycle	Not saved	CUSTOM1: X CUSTOM2: Y CUSTOM3: Z

Usage

```
:TRACe:UNIT CUSTOM<n>, "<unitOfMeasure>"
```

`<n>`	The number of the custom unit, `1`, `2`, or `3`
`<unitOfMeasure>`	A string that defines the custom unit; up to three characters

Details

You can use custom units of measures in buffer math and writable buffers.

If you specify more than two characters, the additional characters are ignored. Some characters are converted to other symbols:

- `u` is displayed as µ.
- `dC` is displayed as `°C`.
- `dF` is displayed as `°F`.
- `RA` is displayed as `V/V`.

This unit is reset when power is cycled. It is not affected by reset.

Example

```
*RST
TRAC:MAKE "expressions", 100, FULL
SENS:FUNC "VOLT"
TRAC:UNIT CUSTOM1, "fb"
TRAC:MATH "expressions", CUSTOM1, ADD
COUN 10
READ? "expressions"
TRAC:DATA? 1, 10, "expressions", READ, EXTR
DISP:SCR READ
```

Instrument has terminals set to FRONT.
Reset the instrument.
Make a buffer named `expressions`, set to store 100 readings with a style of FULL.
Set the measure function to voltage.
Set the custom 1 buffer unit to `fb`.
Set up buffer math, using the custom 1 unit of measure, that adds the present and previous readings.
Set the instrument to make 10 measurements.
Make a reading and store it in the `expressions` buffer.
Read the data in buffer indexes 1 to 10, including the readings and the values generated by the expression.
Display the reading table on the front panel of the instrument.

Also see

:TRACe:MATH (on page 12-128)
:TRACe:WRITe:FORMat (on page 12-141)

:TRACe:WRITe:FORMat

This command sets the units and number of digits of the readings that are written into the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:WRITe:FORMat "<bufferName>", <units>, <displayDigits>
:TRACe:WRITe:FORMat "<bufferName>", <units>, <displayDigits>, <extraUnits>
:TRACe:WRITe:FORMat "<bufferName>", <units>, <displayDigits>, <extraUnits>,
   <extraDigits>
```

<bufferName>	A user-supplied string that indicates the name of the buffer
<units>	The units for the first measurement in the buffer index: ■ AMP ■ AMP_AC ■ AMPAC ■ CELSius ■ CUSTOM1 (user-defined unit) ■ CUSTOM2 (user-defined unit) ■ CUSTOM3 (user-defined unit) ■ DAC ■ DBM ■ DECibel ■ DIO ■ FAHRenheit ■ FARad ■ HERTz ■ KELVin ■ NONE ■ OHM ■ PERCent ■ RATio ■ RECiprocal ■ SECond ■ TOT ■ VOLT ■ VOLT_AC ■ VOLTAC ■ WATT ■ X
<displayDigits>	The number of digits to use for the first value in the buffer index: 3 to 8
<extraUnits>	The units for the second measurement in the buffer index; the selections are the same as <units>; if this parameter is not specified, the value for <units> is used; extra units are only valid for buffer style FULLWRITable
<extraDigits>	The number of digits to use for the second measurement; the selections are the same as <displayDigits>; if this parameter is not specified, the value for <displayDigits> is used; extra digits are only valid for buffer style FULLWRITable

Details

This command is valid when the buffer style is writable or full writable. When the buffer style is set to full writable, you can include an extra value.

The format defines the units and the number of digits that are reported for the data. This command affects how the data is shown in the reading buffer and what is shown on the front-panel Home, Histogram, Reading Table, and Graph screens.

Example 1

```
:TRAC:MAKE "write2me", 1000, WRITable
:TRAC:WRIT:FORM "write2me", WATT, 4
:TRAC:WRIT:READ "write2me", 1
:TRAC:WRIT:READ "write2me", 2
:TRAC:WRIT:READ "write2me", 3
:TRAC:WRIT:READ "write2me", 4
:TRAC:WRIT:READ "write2me", 5
:TRAC:WRIT:READ "write2me", 6
:TRAC:DATA? 1, 6, "write2me", read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is writable.
Set the data format to show units of watts with 4-½ digit resolution.
Write six pieces of data into the buffer.
Read the buffer.
Output:

```
1.000000E+00,Watt DC,2.000000E+00,Watt DC,3.000000E+00,Watt DC,4.000000E+00,Watt
    DC,5.000000E+00,Watt DC,6.000000E+00,Watt DC
```

Example 2

```
:TRAC:MAKE "write2me", 1000, FULLWRIT
:TRAC:WRIT:FORM "write2me", WATT, 4, WATT, 4
:TRAC:WRIT:READ "write2me", 1, 7
:TRAC:WRIT:READ "write2me", 2, 8
:TRAC:WRIT:READ "write2me", 3, 9
:TRAC:WRIT:READ "write2me", 4, 10
:TRAC:WRIT:READ "write2me", 5, 11
:TRAC:WRIT:READ "write2me", 6, 12
:TRAC:DATA? 1, 6, "write2me", read, unit, read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is full writable.
Set the data format to show units of watts with 4½ digit resolution for the first value and the second value in the buffer index.
Write 12 pieces of data into the buffer.
Read the buffer.
Output:

```
1.000000E+00,Watt DC,7.000000E+00,Watt DC,2.000000E+00,Watt DC,8.000000E+00,Watt
    DC,3.000000E+00,Watt DC,9.000000E+00,Watt DC,4.000000E+00,Watt
    DC,1.000000E+01,Watt DC,5.000000E+00,Watt DC,1.100000E+01,Watt
    DC,6.000000E+00,Watt DC,1.200000E+01,Watt DC
```

Also see

Reading buffers (on page 6-1)
:TRACe:MAKE (on page 12-126)
:TRACe:WRITe:READing (on page 12-143)
Writable reading buffers (on page 6-31)

:TRACe:WRITe:READing

This command allows you to write readings into the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

For buffers that are set to the writable buffer style:

```
:TRACe:WRITe:READing "<bufferName>", <readingValue>
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <seconds>
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <seconds>, <fractionalSeconds>
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <seconds>, <fractionalSeconds>,
   <status>
```

For buffers that are set to the full writable buffer style:

```
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <extraValue>
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <extraValue>, <seconds>
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <extraValue>, <seconds>,
   <fractionalSeconds>
:TRACe:WRITe:READing "<bufferName>", <readingValue>, <extraValue>, <seconds>,
   <fractionalSeconds>, <status>
```

`<bufferName>`	A user-supplied string that indicates the name of the buffer
`<readingValue>`	The first value that is recorded in the buffer index
`<extraValue>`	A second value that is recorded in the buffer index (only valid for buffer style `FULLWRITable`)
`<seconds>`	An integer that represents the seconds
`<fractionalSeconds>`	The portion of time that represents the fractional seconds
`<status>`	Information about the reading; see **Details**

Details

This command writes the data you specify into a reading buffer. The reading buffer must be set to the writable or full writable style, which is set when you make the buffer.

Data must be added in chronological order. If the time is not specified for a reading, it is set to one integer second after the last reading. As you write the data, the front-panel home screen updates and displays the reading you entered.

The `<status>` parameter provides additional information about the reading. The options are shown in the following table.

Buffer status bits for sense measurements

Bit (hex)	Decimal	Description
`0x0001`	1	Measure status questionable
`0x0006`	6	A/D converter from which reading originated; for the 2470, this will always be 0 (main)
`0x0008`	8	Measure terminal; front is 1, rear is 0
`0x0010`	16	Measure status limit 2 low
`0x0020`	32	Measure status limit 2 high
`0x0040`	64	Measure status limit 1 low
`0x0080`	128	Measure status limit 1 high
`0x0100`	256	First reading in a group
`0x0200`	512	Relative offset
`0x0400`	1024	Scan

Example 1

```
:TRAC:MAKE "write2me", 1000, WRITable
:TRAC:WRIT:FORM "write2me", WATT, 4
:TRAC:WRIT:READ "write2me", 1
:TRAC:WRIT:READ "write2me", 2
:TRAC:WRIT:READ "write2me", 3
:TRAC:WRIT:READ "write2me", 4
:TRAC:WRIT:READ "write2me", 5
:TRAC:WRIT:READ "write2me", 6
:TRAC:DATA? 1, 6, "write2me", read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is writable.
Set the data format to show a unit of watts with 4½ digit resolution.
Write 6 pieces of data into the buffer.
Read the buffer.
Output:

```
1.000000E+00,Watt DC,2.000000E+00,Watt DC,3.000000E+00,Watt DC,4.000000E+00,Watt
    DC,5.000000E+00,Watt DC,6.000000E+00,Watt DC
```

Example 2

```
:TRAC:MAKE "write2me", 1000, FULLWRIT
:TRAC:WRIT:FORM "write2me", WATT, 4, WATT, 4
:TRAC:WRIT:READ "write2me", 1, 7
:TRAC:WRIT:READ "write2me", 2, 8
:TRAC:WRIT:READ "write2me", 3, 9
:TRAC:WRIT:READ "write2me", 4, 10
:TRAC:WRIT:READ "write2me", 5, 11
:TRAC:WRIT:READ "write2me", 6, 12
:TRAC:DATA? 1, 6, "write2me", read, unit, read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is full writable.
Set the data format to show units of watts with 4½ digit resolution for the first value and the second value in the buffer index.
Write 12 pieces of data into the buffer.
Read the buffer.
Output:

```
1.000000E+00,Watt DC,7.000000E+00,Watt DC,2.000000E+00,Watt DC,8.000000E+00,Watt
    DC,3.000000E+00,Watt DC,9.000000E+00,Watt DC,4.000000E+00,Watt
    DC,1.000000E+01,Watt DC,5.000000E+00,Watt DC,1.100000E+01,Watt
    DC,6.000000E+00,Watt DC,1.200000E+01,Watt DC
```

Also see

Reading buffers (on page 6-1)
:TRACe:DATA? (on page 12-121)
:TRACe:MAKE (on page 12-126)
:TRACe:WRITe:FORMat (on page 12-141)
Writable reading buffers (on page 6-31)

TRIGger subsystem

The commands in this subsystem configure and control the trigger operations, including the trigger model.

:ABORt

This command stops all trigger model commands on the instrument.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:ABORt
```

Details

When this command is received, the instrument stops the trigger model.

Also see

Aborting the trigger model (on page 8-50)
Trigger model (on page 8-25)

:INITiate[:IMMediate]

This command starts the trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:INITiate[:IMMediate]
```

Example

```
INIT
*WAI
```

Starts the trigger model and then waits until the commands are complete to accept new commands.

Also see

:ABORt (on page 12-145)
Trigger model (on page 8-25)

:TRIGger:BLENder<n>:CLEar

This command clears the blender event detector and resets the overrun indicator of blender <n>.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLENder<n>:CLEar
```

<n>	The blender number (up to two)

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

```
:TRIG:BLEN2:CLE
```

Clears the event detector for blender 2.

Also see

None

:TRIGger:BLENder<n>:MODE

This command selects whether the blender performs OR operations or AND operations.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Trigger blender clear	Save settings	AND

Usage

```
:TRIGger:BLENder<n>:MODE <operation>
:TRIGger:BLENder<n>:MODE?
```

<n>	The blender number (up to two)
<operation>	The type of operation: ■ OR ■ AND

Details

This command selects whether the blender waits for any one event (OR) or waits for all selected events (AND) before signaling an output event.

Example 1

`:DIG:LINE3:MODE TRIG, IN`
`:DIG:LINE5:MODE TRIG, IN`
`:TRIG:BLEN1:MODE OR`
`:TRIG:BLEN1:STIM1 DIG3`
`:TRIG:BLEN1:STIM2 DIG5`
	Set digital I/O lines 3 and 5 as trigger in lines. Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

:TRIGger:BLENder<n>:STIMulus<m> (on page 12-148)

:TRIGger:BLENder<n>:OVERrun?

This command indicates whether or not an event was ignored because of the event detector state.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

`:TRIGger:BLENder<n>:OVERrun?`

`<n>`	The blender number (up to two)

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

`:TRIG:BLEN1:OVER?`	If an event was ignored, the output is `1`. If an event was not ignored, the output is `0`.

Also see

:TRIGger:BLENder<n>:CLEar (on page 12-146)

:TRIGger:BLENder<n>:STIMulus<m>

This command specifies the events that trigger the blender.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Trigger blender clear	Save settings	NONE

Usage

```
:TRIGger:BLENder<n>:STIMulus<m> <event>
:TRIGger:BLENder<n>:STIMulus<m>?
```

<n>	The blender number (up to two)
<m>	The stimulus input number (1 to 4)
<event>	See **Details**

Details

There are four stimulus inputs that can each select a different event.

Use none to disable the blender input.

The <event> parameter may be any of the trigger events shown in the following table.

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
A command interface trigger: ■ **Any remote interface:** *TRG ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command device_trigger	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Front-panel TRIGGER key press	DISPlay
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
No trigger event	NONE
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
Source limit condition occurs	SLIMit
Trigger timer <n> (1 to 4) expired	TIMer<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>

Example

`:DIG:LINE3:MODE TRIG, IN`
`:DIG:LINE5:MODE TRIG, IN`
`:TRIG:BLEN1:MODE OR`
`:TRIG:BLEN1:STIM1 DIG3`
`:TRIG:BLEN1:STIM2 DIG5`
	Set digital I/O lines 3 and 5 as trigger in lines. Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

:TRIGger:BLENder<n>:MODE (on page 12-146)

:TRIGger:BLOCk:BRANch:ALWays

This command defines a trigger model block that always goes to a specific block.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

`:TRIGger:BLOCk:BRANch:ALWays <blockNumber>, <branchToBlock>`

`<blockNumber>`	The sequence of the block in the trigger model
`<branchToBlock>`	The block number of the trigger-model block to execute when the trigger model reaches this block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by `<branchToBlock>`.

Example

`TRIG:BLOC:BRAN:ALW 9, 20`	When the trigger model reaches block 9, it always branches to block 20.

Also see

None

:TRIGger:BLOCk:BRANch:COUNter

This command defines a trigger model block that branches to a specified block a specified number of times.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:COUNter <blockNumber>, <targetCount>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<targetCount>`	The number of times to repeat
`<branchToBlock>`	The block number of the trigger model block to execute when the counter is less than to the `<targetCount>` value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value. At that point, the trigger model continues to the next building block in the sequence.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting. Use `:TRIGger:BLOCk:BRANch:COUNter:COUNt?` to query the counter.

Example

```
TRIG:LOAD "EMPTY"
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and make five readings.
Delay a second.
Loop three more times back to block 2.
At end of execution, 15 readings are stored in `defbuffer1`.

Also see

:TRIGger:BLOCk:BRANch:COUNter:COUNt? (on page 12-151)

:TRIGger:BLOCk:BRANch:COUNter:COUNt?

This command returns the count value of the trigger model counter block.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:COUNter:COUNt? <blockNumber>
```

`<blockNumber>`	The sequence of the block in the trigger model

Details

This command returns the counter value. When the counter is active, this returns the present count. If the trigger model has started or is running but has not yet reached the counter block, this value is 0.

Example

```
*RST
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:DEL:CONS 3, 0.1
TRIG:BLOC:BRAN:COUN 4, 10, 2
INIT

TRIG:BLOCK:BRAN:COUN:COUN? 4
*WAI
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and make five readings.
Delay 0.1 s.
Loop ten more times back to block 2.
Send the count command to check the count that has been completed for block 4.
At end of execution, 10 readings are stored in `defbuffer1`.

Also see

:TRIGger:BLOCk:BRANch:COUNter (on page 12-150)

:TRIGger:BLOCk:BRANch:COUNter:RESet

This command creates a block in the trigger model that resets a branch counter to 0.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:COUNter:RESet <blockNumber>, <counter>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<counter>`	The block number of the counter that is to be reset

Details

When the trigger model reaches the Counter Reset block, it resets the count of the specified Branch on Counter block to zero.

Example

```
TRIG:LOAD "EMPTY"
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
TRIG:BLOC:BRAN:COUN:RES 6, 3
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and make five readings.
Delay a second.
Loop three more times back to block 2.
Reset block 3 to 0.

Also see

:TRIGger:BLOCk:BRANch:COUNter (on page 12-150)
:TRIGger:BLOCk:BRANch:COUNter:COUNt? (on page 12-151)

:TRIGger:BLOCk:BRANch:DELTa

This command defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:DELTa <blockNumber>, <targetDifference>, <branchToBlock>
:TRIGger:BLOCk:BRANch:DELTa <blockNumber>, <targetDifference>, <branchToBlock>,
   <measureDigitizeBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<targetDifference>`	The value against which the block compares the difference between the measurements
`<branchToBlock>`	The block number of the trigger model block to execute when the difference between the measurements is less than or equal to the `<targetDifference>`
`<measureDigitizeBlock>`	The block number of the measure/digitize block that makes the measurements to be compared; if this is `0` or undefined, the trigger model uses the previous measure/digitize block

Details

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

Example

`TRIG:BLOC:BRAN:DELT 5, 0.5, 7, 4`	Configure trigger block 5 to compare the differences between the measurements made in block 4. If the difference between them is less the 0.5, branch to block 7.

Also see

Delta block (on page 8-40)

:TRIGger:BLOCk:BRANch:EVENt

This command branches to a specified block when a specified trigger event occurs.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:EVENt <blockNumber>, <event>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<event>`	The event that must occur before the trigger model branches the specified block
`<branchToBlock>`	The block number of the trigger model block to execute when the specified event occurs

Details

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The following table shows the constants for the events.

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	`BLENder<n>`
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	`DIGio<n>`
Front-panel TRIGGER key press	`DISPlay`
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	`LAN<n>`
No trigger event	`NONE`
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	`NOTify<n>`
Source limit condition occurs	`SLIMit`
Trigger timer <n> (1 to 4) expired	`TIMer<n>`
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	`TSPLink<n>`

Example

```
:TRIG:BLOC:BRAN:EVEN 6, DISP, 2
```

When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the next block in the trigger model).

Also see

On event block (on page 8-37)

:TRIGger:BLOCk:BRANch:LIMit:CONStant

This command defines a trigger model block that goes to a specified block if a measurement meets preset criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:LIMit:CONStant <blockNumber>, <limitType>, <limitA>, <limitB>,
   <branchToBlock>
:TRIGger:BLOCk:BRANch:LIMit:CONStant <blockNumber>, <limitType>, <limitA>, <limitB>,
   <branchToBlock>, <measureDigitizeBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<limitType>`	The type of limit (`ABOVe`, `BELow`, `INside`, or `OUTside`)
`<limitA>`	The limit that the measurement is tested against; if *`limitType`* is set to: ■ `ABOVe`: This value is ignored ■ `BELow`: The measurement must be below this value ■ `INside`: The low limit that the measurement is compared against ■ `OUTside`: The low limit that the measurement is compared against
`<limitB>`	The upper limit that the measurement is tested against; if *`limitType`* is set to: ■ `ABOVe`: The measurement must be above this value ■ `BELow`: This value is ignored ■ `INside`: The high limit that the measurement is compared against ■ `OUTside`: The high limit that the measurement is compared against
`<branchToBlock>`	The block number of the trigger model block to execute when the measurement meets the defined criteria
`<measureDigitizeBlock>`	The block number of the measure/digitize block that makes the measurements to be compared; if this is `0` or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected
- Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected
- Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
- Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value

The measurement block must be a measure/digitize block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measure/digitize block is used.

If the limit A is more than the limit B, the values are automatically swapped so that the lesser value is used as the lower limit.

Example

```
TRIGger:BLOCk:BRANch:LIMit:CONStant 5, OUTside, 0.15, 0.65, 8
```

Configure trigger block 5 to check for measurements in the last measure/digitize block. If the measurements are outside of the 0.15 and 0.65 limits, branch to block 8.

Also see

Constant Limit block (on page 8-38)

:TRIGger:BLOCk:BRANch:LIMit:DYNamic

This command defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:LIMit:DYNamic <blockNumber>, <limitType>, <limitNumber>,
   <branchToBlock>
:TRIGger:BLOCk:BRANch:LIMit:DYNamic <blockNumber>, <limitType>, <limitNumber>,
   <branchToBlock>, <measureDigitizeBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<limitType>`	The type of limit (`ABOVe`, `BELow`, `INside`, or `OUTside`)
`<limitNumber>`	The limit number (`1` or `2`)
`<branchToBlock>`	The block number of the trigger model block to execute when the limits are met
`<measureDigitizeBlock>`	The block number of the measure/digitize block that makes the measurements to be compared; if this is `0` or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-dynamic-limits block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

Example

`CALC2:LIM1:STAT ON`
`CALC2:LIM1:LOW -5.17`
`CALC2:LIM1:UPP -4.23`
`TRIG:BLOC:BRAN:LIM:DYN 9, IN, 1, 12, 7`
	Set the limits on with a low limit of -5.17 and a high limit of -4.23. Set trigger block 9 to test if the limit is inside those limits based on the measurement reading at block 7. If the measurement is within the limits, go to block 12.

Also see

Dynamic Limit block (on page 8-39)
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 12-21)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 12-23)

:TRIGger:BLOCk:BRANch:ONCE

This command causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:ONCE <blockNumber>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<branchToBlock>`	The block number of the trigger model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time trigger model execution encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when trigger model execution reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model execution encounters this block.

Example

```
:TRIG:BLOC:BRAN:ONCE 2, 4
```

The first time the trigger model reaches block 2, the trigger model goes to block 4 instead of proceeding to the default sequence of block 3.

Also see

Once block (on page 8-41)
:TRIGger:BLOCk:BRANch:ONCE:EXCLuded (on page 12-158)

:TRIGger:BLOCk:BRANch:ONCE:EXCLuded

This command causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:ONCE:EXCLuded <blockNumber>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<branchToBlock>`	The block number of the trigger model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

Example

```
:TRIG:BLOC:BRAN:ONCE:EXCL 2, 4
```

When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

Once excluded block (on page 8-41)
:TRIGger:BLOCk:BRANch:ONCE (on page 12-158)

:TRIGger:BLOCk:BUFFer:CLEar

This command defines a trigger model block that clears the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BUFFer:CLEar <blockNumber>
:TRIGger:BLOCk:BUFFer:CLEar <blockNumber>, "<bufferName>"
```

`<blockNumber>`	The sequence of the block in the trigger model
`<bufferName>`	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used

Details

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

Example

```
TRIG:LOAD "EMPTY"
TRIG:BLOC:BUFF:CLE 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and make 5 readings.
Delay 1 s.
Loop three more times back to block 2.
At end of execution, 15 readings are stored in `defbuffer1`.

Also see

Buffer clear block (on page 8-27)
:TRACe:MAKE (on page 12-126)

:TRIGger:BLOCk:CONFig:NEXT

This command recalls the settings at the next index of a source or measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:CONFig:NEXT <blockNumber>, "<configurationList>"
:TRIGger:BLOCk:CONFig:NEXT <blockNumber>, "<configurationList>",
   "<configurationList2>"
```

`<blockNumber>`	The sequence of the block in the trigger model
`<configurationList>`	A string that defines the source or measure configuration list to recall
`<configurationList2>`	The second source or measure configuration list from which to recall settings; the type must be opposite of `<configurationList>`

Details

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger model block. If the configuration list has recalled an index with the recall block, the next index in the list is recalled instead of the first. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the next block is reached after that recall, index 2 is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Next block.

If you use a second configuration list, it must be the opposite type of configuration list. For example, if the first configuration list is a measure list, the second configuration list must be a source list.

Example

`TRIG:BLOC:CONF:NEXT 12, "SETTINGS_LIST"`	Set trigger block 12 to restore the settings from the next index that is stored in the configuration list `SETTINGS_LIST`.

Example 2

```
:SENS:CONF:LIST:CRE "measTrigList"
:SENS:CONF:LIST:STORE "measTrigList"
:TRIG:LOAD "Empty"
:TRIG:BLOCK:CONF:RECALL 1, "measTrigList"
:TRIG:BLOC:BUFF:CLE 2, "defbuffer1"
:TRIG:BLOC:CONF:NEXT 3, "measTrigList"
:TRIG:BLOC:LIST?
```

Create a configuration list named `measTrigList`.
Clear the trigger model.
Recall index 1 of a configuration list named `measTrigList`.
Clear reading buffer named `defbuffer1`.
Recall the second index of a configuration list named `measTrigList`.
Output:

```
1) CONFIG_RECALL          CONFIG_LIST: measTrigList  INDEX: 1
2) BUFFER_CLEAR           BUFFER: defbuffer1
3) CONFIG_NEXT            CONFIG_LIST: measTrigList
```

Example 3

```
:TRIG:BLOC:CONF:NEXT 7, "measTrigList", "sourTrigList")
```

Configure trigger block 7 to load the next index in both the configuration list named `measTrigList` and the configuration list named `sourTrigList`.

Also see

Configuration lists (on page 4-82)

:TRIGger:BLOCk:CONFig:PREVious

This command defines a trigger model block that recalls the settings stored at the previous index in a source or measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:CONFig:PREVious <blockNumber>, "<configurationList>"
:TRIGger:BLOCk:CONFig:NEXT <blockNumber>, "<configurationList>",
   "<configurationList2>"
```

`<blockNumber>`	The sequence of the block in the trigger model
`<configurationList>`	A string that defines the source or measure configuration list to recall
`<configurationList2>`	The second source or measure configuration list from which to recall settings; the type must be opposite of `<configurationList>`

Details

The Config List Prev block defines a trigger model block that recalls the settings stored at the previous index in a source or measure configuration list if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are each recalled from the previous index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger model block. If the configuration list has recalled an index with the recall block, the previous index in the list is called instead of the last. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the previous block is reached after that recall, the last index is recalled. However, if the recall block recalled index 3, the previous block would recall index 2. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Prev block.

Example

`TRIG:BLOC:CONF:PREV 14, "SETTINGS_LIST"`	Set trigger block 14 to restore the settings from the previous index that is stored in the configuration list `SETTINGS_LIST`.

Also see

Configuration lists (on page 4-82)

:TRIGger:BLOCk:CONFig:RECall

This command recalls the system settings that are stored in a source or measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:CONFig:RECall <blockNumber>, "<configurationList>"
:TRIGger:BLOCk:CONFig:RECall <blockNumber>, "<configurationList>", <index>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<configurationList>`	A string that defines the source or measure configuration list to recall
`<index>`	The index in the configuration list to recall; default is `1`

Details

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration lists must be defined before you can use this block. If one of the indices for the configuration list changes, verify that the trigger model count is still accurate.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Recall block.

Example

`SOUR:CONF:LIST:CRE "biasLevel"`
`SOUR:FUNC VOLT`
`SENS:FUNC "CURR"`
`SOUR:VOLT:LEV 5`
`SOUR:CONF:LIST:STORE "biasLevel"`
`TRIG:BLOCK:CONF:RECALL 1, "biasLevel", 1`
	Create a configuration list named `biasLevel`. Set the source function to 5 V and the measure function to current.
	Store the source settings at index 1 in the configuration list named `biasLevel`.
	Recall index 1 of the configuration list named `biasLevel` as block 1 of the trigger model.

Also see

Configuration lists (on page 4-82)
[:SENSe[1]]:CONFiguration:LIST:STORe (on page 12-66)
:SOURce[1]:CONFiguration:LIST:STORe (on page 12-73)

:TRIGger:BLOCk:DELay:CONStant

This command adds a constant delay to the execution of a trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:DELay:CONStant <blockNumber>, <time>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<time>`	The amount of time to delay (167 ns to 10 ks; 0 for no delay)

Details

When trigger model execution reaches a delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

Example

`SOUR:CONF:LIST:CRE "ampLevel"`	Create configuration lists named `ampLevel` and `biasLevel`.
`SOUR:CONF:LIST:CRE "biasLevel"`	
`SOUR:FUNC VOLT`	Set the source function to 5 V and the measurement function to current.
`SENS:FUNC "CURR"`	
`SOUR:VOLT:LEV 5`	Store the settings in the `ampLevel` configuration list at index 1.
`SOUR:CONF:LIST:STORE "ampLevel"`	
`SOUR:VOLT:LEV 0`	Set the voltage level to 0 V.
`SOUR:CONF:LIST:STORE "biasLevel"`	Store the setting in the `biasLevel` configuration list at index 1.
`TRIG:BLOC:SOUR:STATE 1, ON`	Set block 1 to turn the source output on.
`TRIG:BLOCK:CONF:RECALL 2, "ampLevel", 1`	Set block 2 to recall the `ampLevel` settings from index 1.
`TRIG:BLOC:DEL:CONS 3, 0.1`	Set block 3 to provide a constant delay of 0.1 s.
`TRIG:BLOCK:MDIG 4`	Set block 4 to make a measurement.
`TRIG:BLOCK:CONF:RECALL 5, "biasLevel", 1`	Set block 5 to recall the `biasLevel` settings at index 1.
`TRIG:BLOCK:DEL:CONS 6, 0.2`	Set block 6 to provide a constant delay of 0.2 s.
`TRIG:BLOCK:BRAN:COUN 7, 19, 2`	Set block 7 to repeat block 2 nineteen times.
`INIT`	

Also see

None

:TRIGger:BLOCk:DELay:DYNamic

This command adds a user delay to the execution of the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:DELay:DYNamic <blockNumber>, <userDelay>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<userDelay>`	The number of the user delay: ■ `SOURce<n>`, where `<n>` is the number of the user delay (1 to 5) set by `:SOURce[1]:<function>:DELay:USER<n>` ■ `MEASure<n>`, where `<n>` is the number of the user delay (1 to 5) set by `[:SENSe[1]]:<function>:DELay:USER<n>`

Details

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Each source function can also have up to five unique user delay times (S1 to S5). The delay time is set by the user-delay command, which is only available over a remote interface.

Example

`:SOUR:VOLT:DEL:USER1 5`
`:TRIG:BLOC:SOUR:STAT 1, ON`
`:TRIG:BLOC:DEL:DYN 2, SOUR1`
`:TRIG:BLOC:MDIG 3`
`:TRIG:BLOC:SOUR:STAT 4, OFF`
`:TRIG:BLOC:BRAN:COUN 5, 10, 1`
`:INIT`
	Set user delay 1 for the voltage source to 5 s.
	Set trigger block 1 to turn the source output on.
	Set trigger block 2 to a dynamic delay that calls source user delay 1.
	Set trigger block 3 to make a measurement.
	Set trigger block 4 to turn the source output off.
	Set trigger block 5 to branch to block 1 ten times.
	Start the trigger model.

Also see

[:SENSe[1]]:<function>:DELay:USER<n> (on page 12-48)
:SOURce[1]:<function>:DELay:USER<n> (on page 12-76)

:TRIGger:BLOCk:DIGital:IO

This command defines a trigger model block that sets the lines on the digital I/O port high or low.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:DIGital:IO <blockNumber>, <bitPattern>
:TRIGger:BLOCk:DIGital:IO <blockNumber>, <bitPattern>, <bitMask>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<bitPattern>`	Sets the value that specifies the output line bit pattern (0 to 63)
`<bitMask>`	Specifies the bit mask; if omitted, all lines are driven (0 to 63)

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern that is specified as an integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63) or omit this parameter. If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

`:DIGital:LINE3:MODE DIG,OUT`
`:DIGital:LINE4:MODE DIG,OUT`
`:DIGital:LINE5:MODE DIG,OUT`
`:DIGital:LINE6:MODE DIG,OUT`
`:TRIG:BLOC:DIG:IO 4, 20, 60`
	The first four lines of code configures digital I./O lines 3 through 6 as digital outputs.
	Trigger block `4` is then configured with a bit pattern of `20` (digital I/O lines 3 and 5 high). The optional bit mask is specified as `60` (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

:DIGital:LINE<n>:MODE (on page 12-24)
Digital I/O bit weighting (on page 8-23)
Digital I/O port configuration (on page 8-14)

:TRIGger:BLOCk:LIST?

This command returns the settings for all trigger model blocks.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLOCk:LIST?
```

Details

This returns the settings for the trigger model.

Example

```
:TRIG:BLOC:LIST?
```

Returns the settings for the trigger model. Example output is:

```
 1) BUFFER_CLEAR          BUFFER: defbuffer1
 2) DELAY_CONSTANT        DELAY: 0.001000000
 3) MEASURE_DIGITIZE      BUFFER: defbuffer1 INITIAL MODE: MEAS INITIAL COUNT: 1
 4) BRANCH_COUNTER        VALUE: 11  BRANCH_BLOCK: 2
```

Also see

None

:TRIGger:BLOCk:LOG:EVENt

This command allows you to log an event in the event log when the trigger model is running.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:LOG:EVENt <blockNumber>, <eventNumber>, "<message>"
```

`<blockNumber>`	The sequence of the block in the trigger model
`<eventNumber>`	The event number: ▪ `INFO<n>` ▪ `WARNing<n>` ▪ `ERRor<n>` Where <n> is 1 to 4; you can define up to four of each type You can also set `ABORt`, which aborts the trigger model immediately and posts a warning event log message
`<message>`	A string up to 31 characters

Details

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

Example

```
TRIGger:BLOCk:LOG:EVENt 9, INFO2, "Trigger model complete"
```

Set trigger-model block 9 to log an event when the trigger model completes.

Also see

None

:TRIGger:BLOCk:MDIGitize

This command defines a trigger block that makes or digitizes a measurement.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:MDIGitize <blockNumber>
:TRIGger:BLOCk:MDIGitize <blockNumber>, "<bufferName>"
:TRIGger:BLOCk:MDIGitize <blockNumber>, "<bufferName>", <count>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<bufferName>`	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used
`<count>`	Specifies the number of readings to make before moving to the next block in the trigger model; set to: ■ A specific value (default is `1` if nothing set) ■ Infinite (run continuously until stopped): `INF` ■ Stop infinite to stop the count: `0` ■ Use most recent count value (default): `AUTO`

Details

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

Digitized measurements are not a feature on the 2470. However, you can use this command to communicate with other Keithley instruments that do offer the digitized measurements feature and to share code with other Keithley instruments.

Firmware versions of the 2470 before version 1.7.0 had a separate measure block. If you have code that is using that block, it works in this version of the 2470 firmware.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1. For example, `:TRIGger:BLOCk:MEASure 1, "defbuffer1"` changes to `:TRIGger:BLOCk:MDIGitize 1, "defbuffer1", 1`.

Example 1

`TRIG:LOAD "EMPTY"`
`TRIG:BLOC:BUFF:CLEAR 1, "defbuffer2"`
`TRIG:BLOC:MDIG 2, "defbuffer2"`
`TRIG:BLOC:BRAN:COUN 3, 5, 2`
`TRIG:BLOC:DEL:CONS 4, 1`
`TRIG:BLOC:BRAN:COUN 5, 3, 2`
`INIT`
`*WAI`
`TRAC:ACT? "defbuffer2"`
	Reset trigger model settings.
	Clear `defbuffer2` at the beginning of the trigger model.
	Set the measurements to be stored in `defbuffer2`.
	Loop and make five readings.
	Delay 1 s.
	Loop three more times back to block 2.
	At end of execution, 15 readings are stored in `defbuffer2`.
	Output:
	15

Example 2

```
*RST
SENS:CONF:LIST:CRE "countactive"
COUN 2
SENSe:CONF:LIST:STOR "countactive"
COUN 10
SENSe:CONF:LIST:STOR "countactive"
COUN 3
SENSe:CONF:LIST:STOR "countactive"

TRIG:BLOC:CONF:NEXT 1, "countactive"
TRIG:BLOC:MDIG 2, "defbuffer1", AUTO
TRIG:BLOC:DEL:CONS 3, 1
TRIG:BLOC:BRAN:COUN 4, 3, 1

INIT
*WAI
TRAC:ACT? "defbuffer1"
```

Reset the instrument.
Set up a configuration list named `countactive`.
Set the measure count to 2. (If you are digitizing, replace `COUN` with `DIG:COUN`.)
Store the count in index 1.
Set the measure count to 10.
Store the count in index 2.
Set the measure count to 3.
Store the count in index 3.
Set up trigger-model block 1 to call the next index from the `countactive` configuration list.
Set block 2 to measure or digitize and store the readings in `defbuffer1` and to use the active count.
Set block 3 to add a delay of 1 s.
Set block 4 to iterate through the trigger model three times, returning to block 1.
Start the trigger model.
Output the number of readings. There should be 15 readings.

Also see

Measure/Digitize block (on page 8-27)
:TRACe:MAKE (on page 12-126)

:TRIGger:BLOCk:NOP

This command creates a placeholder that performs no action in the trigger model; available only using remote commands.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:NOP <blockNumber>
```

`<blockNumber>`	The sequence of the block in the trigger model

Details

If you remove a trigger model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

`TRIG:BLOC:NOP 5`	Set block number 5 to be a no operation block.

Also see

None

:TRIGger:BLOCk:NOTify

This command defines a trigger model block that generates a trigger event and immediately continues to the next block.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:NOTify <blockNumber>, <notifyID>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<notifyID>`	The identification number of the notification; 1 to 8

Details

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

When you call this event, you use the format NOTIFY followed by the notify identification number. For example, if you assign <notifyID> as 4, you would refer to it as NOTIFY4 in the command that references it.

Example

```
:TRIG:BLOC:NOT 5, 2
:TRIG:BLOC:BRAN:EVEN 6, NOTIFY2, 2
```

Define trigger-model block 5 to be the notify 2 event. Assign the notify 2 event to be the trigger for stimulus for the branch event for block 6.

Also see

Notify block (on page 8-31)

:TRIGger:BLOCk:SOURce:STATe

This command defines a trigger block that turns the output source on or off.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:SOURce:STATe <blockNumber>, <state>
```

<blockNumber>	The sequence of the block in the trigger model
<state>	Disable the source: OFF Enable the source: ON

Details

The source output block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as SOURCE_OUTPUT.

Example

```
TRIG:BLOC:SOUR:STAT 1, 1
TRIG:BLOC:DEL:CONS 2, 0.01
TRIG:BLOC:MDIG 3
TRIG:BLOC:BRAN:COUN 4, 100, 2
TRIG:BLOC:SOUR:STAT 5, 0
```

This example turns the output on.
Delay 10 ms.
Make a measurement.
Loop and take 100 readings.
The output is turned off after the 100 readings are made.

Also see

Wait block (on page 8-29)

:TRIGger:BLOCk:WAIT

This command defines a trigger model block that waits for an event before allowing the trigger model to continue.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:WAIT <blockNumber>, <event>
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <clear>
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <clear>, <logic>, <event>
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <clear>, <logic>, <event>, <event>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<event>`	The event that must occur before the trigger block allows trigger execution to continue; see **Details** for event names
`<clear>`	To clear previously detected trigger events when entering the wait block: `ENTer` To immediately act on any previously detected triggers and not clear them (default): `NEVer`
`<logic>`	If each event must occur before the trigger model continues: `AND` If at least one of the events must occur before the trigger model continues: `OR`

Details

You can use the wait block to synchronize measurements with other instruments and devices.

You can set the instrument to wait for the following events:

- Front-panel TRIGGER key press
- Notify (only available when using remote commands)
- Command interface trigger
- Digital input/output signals, such as DIGIO and TSP-Link
- LAN
- Blender
- Timer
- Source limit condition

The event can occur before trigger model execution reaches the wait block. If the event occurs after trigger model execution starts but before the trigger model execution reaches the wait block, the trigger model records the event. By default, when trigger model execution reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when trigger model execution is at the start block and when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action; you cannot combine AND and OR logic in a single block.

You cannot leave the first event as no trigger. If the first event is not defined, the trigger model errors when you attempt to initiate it.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a notify block before the wait block. Set the notify block to use the same timer as the wait block.

The following usage has been deprecated; replace it with the usage above that includes the <clear> parameter.

```
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <logic>, <event>
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <logic>, <event>, <event>
```

The following table shows the constants for the events.

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
A command interface trigger: ■ **Any remote interface:** *TRG ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command device_trigger	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Front-panel TRIGGER key press	DISPlay
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
No trigger event	NONE
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
Source limit condition occurs	SLIMit
Trigger timer <n> (1 to 4) expired	TIMer<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>

Example 1

`:TRIGger:BLOCk:WAIT 9, DISP`	Set trigger model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing and to act on a recorded TRIGGER key event that gets detected either before or after reaching block 9.

Example 2

`:TRIGger:BLOCk:WAIT 9, DISP, ENTer`	Set trigger model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing and to act only on a recorded TRIGGER key event that gets detected when block 9 is reached.

Also see

Wait block (on page 8-29)

:TRIGger:CONTinuous

This command determines the trigger mode setting after bootup.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Nonvolatile memory Save settings	AUTO

Usage

```
TRIGger:CONTinuous <setting>
TRIGger:CONTinuous?
```

`<setting>`	Do not start continuous measurements after bootup: `OFF` Start continuous measurements after bootup: `AUTO` Place the instrument into local control and start continuous measurements after bootup: `RESTart`

Details

Conditions must be valid before continuous measurements can start.

When the restart parameter is selected, the instrument is placed in local mode, aborts any running scripts, and aborts any trigger models that are running. If the command is in a script, it is the last command that runs before the script is aborted. The restart parameter is not stored in nonvolatile memory, so it does not affect start up behavior.

The off and automatic parameters are stored in nonvolatile memory, so they affect start up behavior.

Example

`TRIG:CONT OFF`	When the instrument starts up, the Measurement Method is set to idle.

Also see

None

:TRIGger:DIGital<n>:IN:CLEar

This command clears the trigger event on a digital input line.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:DIGital<n>:IN:CLEar
```

<n>	Digital I/O trigger line (1 to 6)

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to 0).

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

`:TRIG:DIG2:IN:CLE`	Clears the trigger event detector on I/O line 2.

Also see

:DIGital:LINE<n>:MODE (on page 12-24)
Digital I/O port configuration (on page 8-14)
:TRIGger:DIGital<n>:IN:OVERrun? (on page 12-176)

:TRIGger:DIGital<n>:IN:EDGE

This command sets the edge used by the trigger event detector on the given trigger line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	FALL

Usage

```
:TRIGger:DIGital<n>:IN:EDGE <detectedEdge>
:TRIGger:DIGital<n>:IN:EDGE?
```

<n>	Digital I/O trigger line (1 to 6)
<detectedEdge>	The trigger edge value: ■ Detect falling-edge triggers as inputs: FALLing ■ Detect rising-edge triggers as inputs: RISing ■ Detect either falling or rising-edge triggers as inputs: EITHer See **Details** for descriptions of values

Details

This command sets the logic on which the trigger event detector and the output trigger generator operate on the specified trigger line.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Trigger mode values

Value	Description
FALLing	Detects falling-edge triggers as input when the line is configured as an input or open drain.
RISing	Detects rising-edge triggers as input when the line is configured as an open drain.
EITHer	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain.

Example

```
:DIG:LINE4:MODE TRIG,IN
:TRIG:DIG4:IN:EDGE RIS
```

Sets the input trigger mode for the digital I/O line 4 to detect rising-edge triggers as input.

Also see

Digital I/O port configuration (on page 8-14)
:DIGital:LINE<n>:MODE (on page 12-24)
:DIGital:WRITe <n> (on page 12-27)
:TRIGger:DIGital<n>:IN:CLEar (on page 12-175)

:TRIGger:DIGital<n>:IN:OVERrun?

This command returns the event detector overrun status.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:DIGital<n>:IN:OVERrun?
```

<n>	Digital I/O trigger line (1 to 6)

Details

This command returns the event detector overrun status as 0 (false) or 1 (true).

If this is 1, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
TRIG:DIG1:IN:OVER?
```

Returns 0 if no overruns have occurred or 1 if one or more overruns have occurred for I/O line 1.

Also see

Digital I/O port configuration (on page 8-14)
:DIGital:LINE<n>:MODE (on page 12-24)

:TRIGger:DIGital<n>:OUT:LOGic

This command sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NEG

Usage

```
:TRIGger:DIGital<n>:OUT:LOGic <logicType>
:TRIGger:DIGital<n>:OUT:LOGic?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<logicType>`	The output logic of the trigger generator: ■ Assert a TTL-high pulse for output: `POSitive` ■ Assert a TTL-low pulse for output: `NEGative`

Details

This command sets the trigger event generator to assert a TTL pulse for output logic. Positive is a high pulse; negative is a low pulse.

Example

```
:DIG:LINE4:MODE TRIG, OUT
:TRIG:DIG4:OUT:LOG NEG
```

Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

:DIGital:LINE<n>:MODE (on page 12-24)
Digital I/O port configuration (on page 8-14)

:TRIGger:DIGital<n>:OUT:PULSewidth

This command describes the length of time that the trigger line is asserted for output triggers.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	10e-6 (10 µs)

Usage

```
:TRIGger:DIGital<n>:OUT:PULSewidth <width>
:TRIGger:DIGital<n>:OUT:PULSewidth?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<width>`	Pulse length (0 to 100 ks)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely.

Example

`DIG:LINE1:MODE TRIG, OUT` `TRIG:DIG1:OUT:PULS 2`	Set digital line 1 to trigger out. Set the pulse to 2 s.

Also see

:DIGital:LINE<n>:MODE (on page 12-24)
:DIGital:WRITe <n> (on page 12-27)
Digital I/O port configuration (on page 8-14)

:TRIGger:DIGital<n>:OUT:STIMulus

This command selects the event that causes a trigger to be asserted on the digital output line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:DIGital<n>:OUT:STIMulus <event>
:TRIGger:DIGital<n>:OUT:STIMulus?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<event>`	The event to use as a stimulus; see **Details**

Details

The digital trigger pulsewidth command determines how long the trigger is asserted.

The trigger stimulus for a digital I/O line can be set to one of the trigger events that are described in the following table.

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	`BLENder<n>`
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	`DIGio<n>`
Front-panel TRIGGER key press	`DISPlay`
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	`LAN<n>`
No trigger event	`NONE`
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	`NOTify<n>`
Source limit condition occurs	`SLIMit`
Trigger timer <n> (1 to 4) expired	`TIMer<n>`
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	`TSPLink<n>`

Example

`:TRIG:DIG2:OUT:STIMulus TIM3`	Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.

Also see

Digital I/O port configuration (on page 8-14)
:DIGital:LINE<n>:STATe (on page 12-26)
:TRIGger:DIGital<n>:OUT:LOGic (on page 12-177)

:TRIGger:LAN<n>:IN:CLEar

This command clears the event detector for a LAN trigger.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LAN<n>:IN:CLEar
```

`<n>`	The LAN event number (1 to 8) to clear

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

`:TRIG:LAN5:IN:CLE`	Clears the event detector with LAN packet 5.

Also see

:TRIGger:LAN<n>:IN:OVERrun? (on page 12-180)

:TRIGger:LAN<n>:IN:EDGE

This command sets the trigger operation and detection mode of the specified LAN event.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	EITH

Usage

```
:TRIGger:LAN<n>:IN:EDGE <mode>
:TRIGger:LAN<n>:IN:EDGE?
```

`<n>`	The LAN event number (1 to 8)
`<mode>`	The trigger mode; see the **Details** for more information

Details

This command controls how the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values		
Mode	**Trigger packets detected as input**	**LAN trigger packet generated for output with a…**
`EITHer`	Rising or falling edge (positive or negative state)	negative state
`FALLing`	Falling edge (negative state)	negative state
`RISing`	Rising edge (positive state)	positive state

Example

`:TRIG:LAN2:IN:EDGE FALL`	Set the LAN trigger mode for event 2 to falling edge.

Also see

Digital I/O (on page 8-12)
TSP-Link System Expansion Interface (on page 9-1)

:TRIGger:LAN<n>:IN:OVERrun?

This command indicates the overrun status of the LAN event detector.

Type	Affected by	Where saved	Default value
Query only	LAN trigger clear	Not applicable	Not applicable

Usage

```
:TRIGger:LAN<n>:IN:OVERrun?
```

`<n>`	The LAN event number (1 to 8)

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

The trigger overrun state for the specified LAN packet is returned as `1` (true) or `0` (false).

Example

`TRIG:LAN5:IN:OVER?`	Checks the overrun status of a trigger on LAN5 and outputs the value, such as: `0`

Also see

:TRIGger:LAN<n>:IN:CLEar (on page 12-179)

:TRIGger:LAN<n>:OUT:CONNect:STATe

This command prepares the event generator for outgoing trigger events.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LAN<n>:OUT:CONNect:STATe <state>
:TRIGger:LAN<n>:OUT:CONNect:STATe?
```

`<n>`	The LAN event number (1 to 8)
`<state>`	Do not send event messages: `OFF` or `0` Prepare to send event messages: `ON` or `1`

Details

When this is set to `ON`, the instrument prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event is changed.

When this is set to `OFF`, for TCP connections, this closes the TCP connection.

Example

`:TRIGger:LAN1:OUT:PROTocol MULT` `:TRIGger:LAN1:OUT:CONNect:STATe ON`	Set the protocol to multicast and prepare the event generator to send event messages.

Also see

:TRIGger:LAN<n>:OUT:IP:ADDRess (on page 12-181)
:TRIGger:LAN<n>:OUT:PROTocol (on page 12-183)

:TRIGger:LAN<n>:OUT:IP:ADDRess

This command specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	"0.0.0.0"

Usage

```
:TRIGger:LAN<n>:OUT:IP:ADDRess "<address>"
:TRIGger:LAN<n>:OUT:IP:ADDRess?
```

`<n>`	The LAN event number (1 to 8)
`<address>`	A string that represents the LAN address in dotted decimal notation

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

`TRIG:LAN1:OUT:IP:ADDR "192.0.32.10"`	Use IP address `192.0.32.10` to connect the LAN trigger.

Also see

:TRIGger:LAN<n>:OUT:CONNect:STATe (on page 12-181)

:TRIGger:LAN<n>:OUT:LOGic

This command sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NEG

Usage

```
:TRIGger:LAN<n>:OUT:LOGic <logicType>
:TRIGger:LAN<n>:OUT:LOGic?
```

`<n>`	The LAN event number (1 to 8)
`<logicType>`	The type of logic: ■ `POSitive` ■ `NEGative`

Example

`TRIG:LAN1:OUT:LOG POS`	Set the logic to positive.

Also see

None

:TRIGger:LAN<n>:OUT:PROTocol

This command sets the LAN protocol to use for sending trigger messages.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	TCP

Usage

```
:TRIGger:LAN<n>:OUT:PROTocol <protocol>
:TRIGger:LAN<n>:OUT:PROTocol?
```

`<n>`	The LAN event number (1 to 8)
`<protocol>`	The protocol to use for messages from the trigger: ■ `TCP` ■ `UDP` ■ `MULTicast`

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored, and event messages are sent to the multicast address 224.0.23.159.

Example

`:TRIG:LAN1:OUT:PROT TCP` `:TRIG:LAN1:OUT:CONN:STAT`	Set the LAN protocol for trigger messages to be TCP and re-connect the LAN trigger event generator.

Also see

:TRIGger:LAN<n>:OUT:CONNect:STATe (on page 12-181)
:TRIGger:LAN<n>:OUT:IP:ADDRess (on page 12-181)

:TRIGger:LAN<n>:OUT:STIMulus

This command specifies events that cause this trigger to assert.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:LAN<n>:OUT:STIMulus <LANevent>
:TRIGger:LAN<n>:OUT:STIMulus?
```

`<n>`	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)
`<LANevent>`	The LAN event that causes this trigger to assert

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored, and the action overrun is set.

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	`BLENder<n>`
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	`DIGio<n>`
Front-panel TRIGGER key press	`DISPlay`
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	`LAN<n>`
No trigger event	`NONE`
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	`NOTify<n>`
Source limit condition occurs	`SLIMit`
Trigger timer <n> (1 to 4) expired	`TIMer<n>`
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	`TSPLink<n>`

Example

`TRIG:LAN1:OUT:STIM TIM1`	Set the timer 1 trigger event as the source for the LAN packet 1 trigger stimulus.

Also see

:TRIGger:LAN<n>:OUT:CONNect:STATe (on page 12-181)

:TRIGger:LOAD "ConfigList"

This command loads a trigger-model template configuration that uses source and measure configuration lists.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "ConfigList", "<measureConfigList>", "<sourceConfigList>"
:TRIGger:LOAD "ConfigList", "<measureConfigList>", "<sourceConfigList>", <delay>
:TRIGger:LOAD "ConfigList", "<measureConfigList>", "<sourceConfigList>", <delay>,
   "<bufferName>"
```

`<measureConfigList>`	A string that contains the name of the measurement configuration list to use
`<sourceConfigList>`	A string that contains the name of the source configuration list to use
`<delay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This trigger-model template incorporates a source configuration list and measure configuration list. You must set up the configuration lists before loading the trigger model. If the configuration lists change, you must resend this command.

You can also set a delay and change the reading buffer.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

This command replaces the `TRIGger:LOAD:CONFiguration:LIST` command, which is deprecated.

Example

```
*RST
:SOURce:CONF:LIST:CRE "SOURCE_LIST"
:SENS:CONF:LIST:CRE "MEASURE_LIST"
:SOUR:VOLT 1
:SOURce:CONF:LIST:STORE "SOURCE_LIST"
:SENS:CURR:RANG 1e-3
:SENSe:CONF:LIST:STOR "MEASURE_LIST"
:SOUR:VOLT 5
:SOURce:CONF:LIST:STORE "SOURCE_LIST"
:SENS:CURR:RANG 10e-3
:SENSe:CONF:LIST:STOR "MEASURE_LIST"
:SOUR:VOLT 10
:SOURce:CONF:LIST:STORE "SOURCE_LIST"
:SENS:CURR:RANG 100e-3
:SENSe:CONF:LIST:STOR "MEASURE_LIST"
:TRIG:LOAD "ConfigList", "MEASURE_LIST", "SOURCE_LIST"
INIT
```

Set up a source configuration list named `SOURCE_LIST` and a measurement configuration list named `MEASURE_LIST`.
Create the source list with three indexes that set the source voltage levels to 1, 5, and 10.
Create the measure list with three indexes that set the measure current ranges to 1 mA, 10 mA, and 100 mA.
Load the configuration list trigger model, using these two configuration lists. Start the trigger model.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)

:TRIGger:LOAD "DurationLoop"

This command loads a trigger-model template configuration that makes continuous measurements for a specified amount of time.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "DurationLoop", <duration>
:TRIGger:LOAD "DurationLoop", <duration>, <delay>
:TRIGger:LOAD "DurationLoop", <duration>, <delay>, "<readingBuffer>"
```

`<duration>`	The amount of time for which to make measurements (167 ns to 100 ks)
`<delay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<readingBuffer>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

When you load this trigger-model template, you can specify amount of time to make a measurement and the length of the delay before the measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

This command replaces the `TRIGger:LOAD:LOOP:DURation` command, which is deprecated.

Example

`*RST`
`SOUR:FUNC VOLT`
`SOUR:VOLT 5`
`SENS:FUNC "CURR"`
`TRIG:LOAD "DurationLoop", 10, 0.01`
`INIT`
	Reset the instrument. Set the instrument to source voltage at 5 V. Set to measure current.
	Load the DurationLoop trigger model to take measurements for 10 s with a 10 ms delay before each measurement.
	Start the trigger model.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)

:TRIGger:LOAD "Empty"

This command resets the trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LOAD "Empty"
```

Details

When you load this trigger-model template, any blocks that have been defined in the trigger model are cleared so the trigger model has no blocks defined.

Example

```
TRIG:LOAD "Empty"
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
TRAC:ACT? "defbuffer1"
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of execution of the trigger model.
Loop and take 5 readings.
Delay 1 s.
Loop three more times back to block 2.
At the end of execution, 15 readings are stored in `defbuffer1`.
Output:
15

Also see

None

:TRIGger:LOAD "GradeBinning"

This command loads a trigger-model template configuration that sets up a grading operation.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>
```

```
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>, <limit4Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>, <limit4Low>, <limit4Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>, <limit4Low>, <limit4Pattern>, "<bufferName>"
```

`<components>`	The number of components to measure (1 to 268,435,455)
`<startInLine>`	The input line that starts the test; `5` for digital line 5, `6` for digital line 6; default is `5`
`<startDelay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<endDelay>`	The delay time after the measurement (167 ns to 10 ks); default is `0` for no delay
`<limitxHigh>`	*x* is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
`<limitxLow>`	*x* is 1, 2, 3, or 4; the lower limit that the measurement is compared against
`<limit1Pattern>`	The bit pattern that is sent when the measurement fails limit 1; range 1 to 15; default is 1
`<limit2Pattern>`	The bit pattern that is sent when the measurement fails limit 2; range 1 to 15; default is 2
`<limit3Pattern>`	The bit pattern that is sent when the measurement fails limit 3; range 1 to 15; default is 4
`<limit4Pattern>`	The bit pattern that is sent when the measurement fails limit 4; range 1 to 15; default is 8
`<allPattern>`	The bit pattern that is sent when all limits have passed; 1 to 15; default is 15
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This trigger-model template allows you to grade components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the pass pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)

:TRIGger:LOAD "LogicTrigger"

This command loads a trigger-model template configuration that sets up a digital trigger through the digital I/O.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>, <clear>
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>, <clear>, <delay>
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>, <clear>, <delay>,
   "<bufferName>"
```

`<digInLine>`	The digital input line (1 to 6); also, the event that the trigger model will wait on in block 1
`<digOutLine>`	The digital output line (1 to 6)
`<count>`	The number of measurements the instrument will make
`<clear>`	To clear previously detected trigger events when entering the wait block: `ENTer` To immediately act on any previously detected triggers and not clear them (default): `NEVer`
`<delay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<bufferName>`	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; default is `defbuffer1`

Details

This trigger model waits for a digital input event to occur, makes a measurement, and issues a notify event. The notify event asserts a digital output line.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

Example

```
:TRIGger:LOAD "LogicTrigger", 1, 2, 10, ENTer, 0.001, "defbuffer1"
```

Set up the template to use the digital in line 1 and wait until the wait block is entered to detect the pulse from digital in line 1 and to trigger measurements.
Pulse digital out line 2 when the measurement is complete.
Make 10 measurements, with a delay of 1 ms before each measurement.
Store the measurements in `defbuffer1`.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)
:TRIGger:DIGital<n>:OUT:LOGic (on page 12-177)

:TRIGger:LOAD "LoopUntilEvent"

This command loads a trigger-model template configuration that makes continuous measurements until the specified event occurs.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <delay>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <delay>, "<bufferName>"
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <clear>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <clear>, <delay>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <clear>, <delay>,
   "<bufferName>"
```

`<eventConstant>`	The event that ends infinite triggering or readings set to occur before the trigger; see **Details**
`<position>`	The number of readings to make in relation to the size of the reading buffer; enter as a percentage (0% to 100%)
`<clear>`	To clear previously detected trigger events when entering the wait block (default): `ENTer` To immediately act on any previously detected triggers and not clear them: `NEVer`
`<delay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

The event constant is the event that ends infinite triggering or ends readings set to occur before the trigger and start post-trigger readings. The trigger model makes readings until it detects the event constant. After the event, it makes a finite number of readings, based on the setting of the trigger position.

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled. For example, if this is set to 75 for a reading buffer that holds 10,000 readings, the trigger model makes 2500 readings after it detects the source event. There are 7500 pre-trigger readings and 2500 post-trigger readings.

The instrument makes two sets of readings. The first set is made until the trigger event occurs. The second set is made after the trigger event occurs, up to the number of readings calculated by the position parameter.

You cannot have the event constant set at none when you run this trigger-model template.

You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

Trigger events	
Event description	**Event constant**
Front-panel TRIGGER key press	`DISPlay`
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	`NOTify<n>`
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	`DIGio<n>`
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	`TSPLink<n>`
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	`LAN<n>`
Trigger event blender <n> (up to two), which combines trigger events	`BLENder<n>`
Trigger timer <n> (1 to 4) expired	`TIMer<n>`
Source limit condition occurs	`SLIMit`

Example

`*RST`
`SENS:FUNC "CURR"`
`TRIG:LOAD "LoopUntilEvent", DISP, 25`
`INIT`
	Reset the instrument.
	Set the instrument to measure DC current.
	Set the LoopUntilEvent trigger model to make measurements until the front-panel TRIGGER key is pressed after starting the trigger model, then make measurements that constitute 75% of the reading buffer.
	Start the trigger model.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)

:TRIGger:LOAD "SimpleLoop"

This command loads a trigger-model template configuration.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "SimpleLoop", <count>
:TRIGger:LOAD "SimpleLoop", <count>, <delay>
:TRIGger:LOAD "SimpleLoop", <count>, <delay>, "<bufferName>"
```

`<count>`	The number of measurements the instrument will make
`<delay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen.

You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

This command replaces the `TRIGger:LOAD:LOOP:SIMPle` command, which is deprecated.

Example

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SOUR:FUNC VOLT
SOUR:DEL 0.1
SOUR:VOLT 5
SOUR:VOLT:ILIM 0.01
TRIG:LOAD "SimpleLoop", 10
OUTP ON
INIT
*WAI
TRAC:DATA? 1, 10, "defbuffer1", SOUR, READ, REL
```

Reset the instrument and set it to measure current with automatic range setting.
Source 5 V with a source delay of 0.1 s.
Set a current limit of 0.01 A.
Set a simple trigger loop with a count of 10.
Turn the output on.
Start the trigger model.
Postpone execution of subsequent commands until all previous commands are finished.
Read data and store the source, reading, and relative time.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)

:TRIGger:LOAD "SortBinning"

This command loads a trigger-model template configuration that sets up a sorting operation.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>
```

```
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>, <limit4Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>, <limit4Low>, <limit4Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
   <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
   <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
   <limit4High>, <limit4Low>, <limit4Pattern>, "<bufferName>"
```

`<components>`	The number of components to measure
`<startInLine>`	The input line that starts the test; `5` for digital line 5, `6` for digital line 6; default is `5`
`<startDelay>`	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
`<endDelay>`	The delay time after the measurement (167 ns to 10 ks); default is `0` for no delay
`<limitxHigh>`	*x* is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
`<limitxLow>`	*x* is 1, 2, 3, or 4; the lower limit that the measurement is compared against
`<limit1Pattern>`	The bit pattern that is sent when the measurement passes limit 1; range 1 to 15; default is 1
`<limit2Pattern>`	The bit pattern that is sent when the measurement passes limit 2; range 1 to 15; default is 2
`<limit3Pattern>`	The bit pattern that is sent when the measurement passes limit 3; range 1 to 15; default is 4
`<limit4Pattern>`	The bit pattern that is sent when the measurement passes limit 4; range 1 to 15; default is 8
`<allPattern>`	The bit pattern that is sent when all limits have failed; 1 to 15; default is 15
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This trigger-model template allows you to sort components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the all fail pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCk:LIST?` command to view the trigger-model blocks in a list format.

Also see

:TRIGger:BLOCk:LIST? (on page 12-166)

:TRIGger:PAUSe

This command pauses a running trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:PAUSe
```

Details

This command pauses the trigger model.

To continue the trigger model, send the resume command.

Example

```
INIT
TRIG:PAUS
TRIG:RES
*WAI
```

Start a trigger model, then pause and resume it.

Also see

:INITiate[:IMMediate] (on page 12-145)
:TRIGger:RESume (on page 12-196)

:TRIGger:RESume

This command continues a paused trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:RESume
```

Details

This command continues running the trigger model operation if the trigger model was paused.

Example

```
INIT
TRIG:PAUS
TRIG:RES
*WAI
```

Start a trigger model, then pause and resume it.

Also see

:INITiate[:IMMediate] (on page 12-145)
:TRIGger:PAUSe (on page 12-195)

:TRIGger:STATe?

This command returns the present state of the trigger model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:STATe?
```

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model last executed.

The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped

Example

`:TRIG:STAT?`	An example output if the trigger model is inactive and ended at block 9 is: `IDLE;IDLE;9`

Also see

None

:TRIGger:TIMer<n>:CLEar

This command clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:TIMer<n>:CLEar
```

`<n>`	Trigger timer number (1 to 4)

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

`:TRIG:TIM1:CLE`	Clears trigger timer 1.

Also see

:TRIGger:TIMer<n>:COUNt (on page 12-197)
:TRIGger:TIMer<n>:STARt:OVERrun? (on page 12-200)

:TRIGger:TIMer<n>:COUNt

This command sets the number of events to generate each time the timer generates a trigger event or is enabled as a timer or alarm.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1

Usage

```
:TRIGger:TIMer<n>:COUNt <count>
:TRIGger:TIMer<n>:COUNt?
```

`<n>`	Trigger timer number (1 to 4)
`<count>`	The number of times to repeat the trigger (0 to 1,048,575)

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example 1

`TRIG:TIM2:COUN 4`	Set the number of events to generate for trigger timer 2 to four.

Example 2

```
*RST
TRIG:TIM4:DEL 0.5
TRIG:TIM4:STAR:STIM NOT8
TRIG:TIM4:STAR:GEN OFF
TRIG:TIM4:COUN 20
TRIG:TIM4:STAT ON

TRIG:LOAD "Empty"
TRIG:BLOC:BUFF:CLEAR 1, "defbuffer1"
TRIG:BLOC:NOT 2, 8
TRIG:BLOC:WAIT 3, TIM4
TRIG:BLOC:MDIG 4
TRIG:BLOC:BRAN:COUN 5, 20, 3
INIT
*WAI
TRAC:ACT? "defbuffer1"
```

Set trigger timer 4 to have a 0.5 s delay.
Set the stimulus for trigger timer 4 to be the notify 8 event.
Set the trigger timer 4 stimulus to off.
Set the timer event to occur when the timer delay elapses.
Set the trigger timer 4 count to 20.
Enable trigger timer 4.

Clear the trigger model.
Set trigger-model block 1 to clear the buffer.
Set trigger-model block 2 to generate the notify 8 event.
Set trigger-model block 3 to wait for trigger timer 4 to occur.
Set trigger-model block 4 to make a reading and store it in default buffer 1.
Set trigger-model block 5 to repeat the trigger model 20 times, starting at block 3.
Start the trigger model.
Output the number of entries in default buffer 1.

Output:
`20`

Also see

:TRIGger:TIMer<n>:CLEar (on page 12-197)
:TRIGger:TIMer<n>:DELay (on page 12-199)

:TRIGger:TIMer<n>:DELay

This command sets and reads the timer delay.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1e-5 (10 µs)

Usage

```
:TRIGger:TIMer<n>:DELay <interval>
:TRIGger:TIMer<n>:DELay?
```

<n>	Trigger timer number (1 to 4)
<interval>	Delay interval (8 µs to 100 ks)

Details

A delay is the period after the timer is triggered and before the timer generates a trigger event. Each time the timer is triggered, it uses this delay period.

Reading this command returns the delay interval that will be used the next time the timer is triggered.

Example

`TRIG:TIM2:DEL 50E-6`	Set trigger timer 2 to delay for 50 µs.

Also see

None

:TRIGger:TIMer<n>:STARt:FRACtional

This command configures an alarm or a time in the future when the timer will start.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:TRIGger:TIMer<n>:STARt:FRACtional <time>
:TRIGger:TIMer<n>:STARt:FRACtional?
```

<n>	Trigger timer number (1 to 4)
<time>	The time in fractional seconds (0 to <1 s)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time in the past or if it is in the future.

Example

`TRIG:TIM1:STAR:SEC 60`
`TRIG:TIM1:START:FRAC 0.5`
`TRIG:TIM1:STAT ON`
	Set the timer for 60.5 s.
	Enable the trigger timer for timer 1.

Also see

:TRIGger:TIMer<n>:STARt:SEConds (on page 12-201)
:TRIGger:TIMer<n>:STATe (on page 12-203)

:TRIGger:TIMer<n>:STARt:GENerate

This command specifies when timer events are generated.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (OFF)

Usage

```
:TRIGger:TIMer<n>:STARt:GENerate <state>
:TRIGger:TIMer<n>:STARt:GENerate?
```

`<n>`	Trigger timer number (1 to 4)
`<state>`	Generate a timer event when the timer delay elapses: `OFF` or `0` Generate a timer event when the timer starts and when the delay elapses: `ON` or `1`

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event `TIMER`*`N`*.

Example

`TRIG:TIM3:STAR:GEN ON`	Set trigger timer 3 to generate an event when the timer starts and when the timer delay elapses.

Also see

None

:TRIGger:TIMer<n>:STARt:OVERrun?

This command indicates if an event was ignored because of the event detector state.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:TIMer<n>:STARt:OVERrun?
```

`<n>`	Trigger timer number (1 to 4)

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

This returns `0` if there is no overrun or `1` if there is an overrun.

Example

`TRIG:TIM1:STAR:OVER?`	Checks the overrun status on trigger timer 1.

Also see

None

:TRIGger:TIMer<n>:STARt:SEConds

This command configures an alarm or a time in the future when the timer will start.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (0 s)

Usage

```
:TRIGger:TIMer<n>:STARt:SEConds <time>
:TRIGger:TIMer<n>:STARt:SEConds?
```

`<n>`	Trigger timer number (1 to 4)
`<time>`	The time: 0 to 2,147,483,647 s

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

`TRIG:TIM1:STAR:SEC 60`
`TRIG:TIM1:STAR:FRAC 0.5`
`TRIG:TIM1:STAT ON`
	Set the timer for 60.5 s.
	Enable the trigger timer for timer 1.

Also see

:TRIGger:TIMer<n>:STATe (on page 12-203)

:TRIGger:TIMer<n>:STARt:STIMulus

This command describes the event that starts the trigger timer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:TIMer<n>:STARt:STIMulus <event>
:TRIGger:TIMer<n>:STARt:STIMulus?
```

<n>	Trigger timer number (1 to 4)
<event>	The event that starts the trigger timer; see **Details**

Details

Set the stimulus to any trigger event to start the timer when that event occurs.

Set the stimulus to none to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

Trigger events	
Event description	**Event constant**
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
A command interface trigger: ■ **Any remote interface:** *TRG ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command device_trigger	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Front-panel TRIGGER key press	DISPlay
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
No trigger event	NONE
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
Source limit condition occurs	SLIMit
Trigger timer <n> (1 to 4) expired	TIMer<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>

Example

`*RST`	Reset the instrument to default settings. Set digital I/O line 1 for use as a trigger input.
`DIG:LINE1:MODE TRIG,IN`	Set digital I/O line 2 for use as a trigger output.
`DIG:LINE2:MODE TRIG,OUT`	Set timer 1 to delay 35 ms.
`TRIG:TIM1:DEL 35e-3`	Set timer 1 to start delaying once the digital I/O 1 event is detected.
`TRIG:TIM1:STAR:STIM DIG1`	Set digital I/O line 2 to output a pulse once the timer 1 event is detected.
`TRIG:DIG2:OUT:STIM TIM1`	

Also see

None

:TRIGger:TIMer<n>:STATe

This command enables the trigger timer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF (0)

Usage

```
:TRIGger:TIMer<n>:STATe <state>
:TRIGger:TIMer<n>:STATe?
```

`<n>`	Trigger timer number (1 to 4)
`<state>`	Disable the trigger timer: `OFF` or `0` Enable the trigger timer: `ON` or `1`

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

Example 1

`DIG:LINE3:MODE TRIG,OUT`
`TRIG:DIG3:OUT:STIM TIM2`
`SYSTem:TIME?`
`TRIG:TIM2:START:SECONDS <current time> + 60`
`TRIG:TIM2:STAT ON`
	To configure timer 2 for an alarm to fire 1 minute from now and output a pulse on digital I/O line 3, query to get the current time. Add 60 s to that value and use that to configure the start seconds. Enable the timer.

Example 2

`*RST`
`DIG:LINE5:MODE TRIG,OUT`
`TRIG:DIG5:OUT:STIM TIM3`
`TRIG:TIM3:DEL 3e-3`
`TRIG:TIM3:COUNT 5`
`TRIG:TIM3:STAT ON`
	Configure timer 3 to generate five pulses on digital I/O line 5 that are 3 ms apart.

Example 3

`*RST`
`DIG:LINE3:MODE TRIG,IN`
`DIG:LINE5:MODE TRIG,OUT`
`TRIG:DIG5:OUT:STIM TIM3`
`TRIG:TIM3:DEL 3e-3`
`TRIG:TIM3:COUNT 5`
`TRIG:TIM3:START:STIM DIG3`
`TRIG:TIM3:STAT ON`
	Configure timer 3 to generate 5 pulses on digital I/O line 5 that are 3 ms apart when a digital input is detected on digital line 3.

Also see

None

Section 13

Introduction to TSP commands

In this section:

Introduction to TSP operation 13-1
Fundamentals of scripting for TSP 13-4
Fundamentals of programming for TSP 13-12
Test Script Builder (TSB) 13-31
Memory considerations for the run-time environment 13-41
About TSP Commands 13-42

Introduction to TSP operation

Instruments that are enabled for Test Script Processor (TSP®) operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you would when using any other instrument.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without an external controller. You can load a series of TSP commands into the instrument using a remote computer or the front-panel port with a USB flash drive. You can store these commands as a script that can be run later by sending a single command message to the instrument.

You do not have to choose between using conventional control or script control. You can combine these forms of instrument control in the way that works best for your test application.

Controlling the instrument by sending individual command messages

The simplest method of controlling an instrument through the communication interface is to send it a message that contains remote commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

TSP commands can be function-based or attribute-based. Function-based commands are commands that control actions or activities. Attribute-based commands define characteristics of an instrument feature or operation.

Constants and enumerated types are commands that represent fixed values.

Functions

Function-based commands control actions or activities. A function-based command performs an immediate action on the instrument.

Each function consists of a function name followed by a set of parentheses ( ). You should only include information in the parentheses if the function takes a parameter. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Example 1

`beeper.beep(0.5, 2400)`
`delay(0.250)`
`beeper.beep(0.5, 2400)`
	Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

You can use the results of a function-based command directly or assign variables to the results for later access. The following code defines `x` and prints it.

`x = math.abs(-100)` `print(x)`	Output: `100`

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, a characteristic of TSP-enabled instruments is the model number (`localnode.model`).

Attributes can be read-only, read-write, or write-only. They can be used as a parameter of a function or assigned to another variable.

To set the characteristics, attribute-based commands define a value. For many attributes, the value is in the form of a number, enumerated type, or a predefined constant.

Example 1: Set an attribute using a number

`testData = buffer.make(500)` `testData.capacity = 600`	Use a function to create a buffer named `testData` with a capacity of 500, then use the *`bufferVar`*`.capacity` attribute to change the capacity to 600.

Example 2: Set an attribute using an enumerated type

`display.lightstate = display.STATE_LCD_75`	This attribute controls the brightness of the front-panel display and buttons. Setting this attribute to `display.STATE_LCD_75` sets the brightness of the display and buttons to 75% of full brightness.

Example 3: Set an attribute using a constant

`format.data = format.REAL64`	Using the constant `REAL64` sets the print format to double precision floating point format.

Reading an attribute

To read an attribute, you can use the attribute as the parameter of a function or assign it to another variable.

Example 1: Read an attribute using a function

`print(display.lightstate)`	Reads the status of the light state by passing the attribute to the print function. If the display light state is set to 50%, the output is: `display.STATE_LCD_50`

Example 2: Read an attribute using a variable

`light = display.lightstate` `print(light)`	This reads the light state by assigning the attribute to a variable named `light`. If the display light state is set to 25%, the output is: `display.STATE_LCD_25`

Queries

Test Script Processor (TSP®) enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` and `printnumber()` commands generate output in the form of response messages. Each `print()` command creates one response message.

Example

`x = 10` `print(x)`	Example of an output response message: `10` Note that your output may be different if you set your ASCII precision setting to a different value.

USB flash drive path

You can use the file commands to open and close directories and files, write data, or to read a file on an installed USB flash drive.

The root folder of the USB flash drive has the absolute path:

`"/usb1/"`

Information on scripting and programming

If you need information about using scripts with your TSP-enabled instrument, see Fundamentals of scripting for TSP (on page 13-4).

If you need information about using the Lua programming language with the instrument, see Fundamentals of programming for TSP (on page 13-12).

Fundamentals of scripting for TSP

NOTE

Though it can improve your process to use scripts, you do not have to create scripts to use the instrument. Most of the examples in the documentation can be run by sending individual command messages. The next few sections of the documentation describe scripting and programming features of the instrument. You only need to review this information if you are using scripting and programming.

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument more efficiently.

Scripts offer several advantages compared to sending individual commands from the host controller (computer):

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs more quickly and efficiently when it processes scripts than it does when it processes individual commands.
- You can incorporate features such as looping and branching into scripts.
- Scripts allow the controller to perform other tasks while the instrument is running a script, enabling some parallel operation.
- Scripts eliminate repeated data transfer times from the controller.

In the instrument, the Test Script Processor (TSP®) scripting engine processes and runs scripts.

This section describes how to create, load, modify, and run scripts.

What is a script?

A script is a collection of instrument control commands and programming statements. Scripts that you create are referred to as **user scripts**.

Your scripts can be interactive. Interactive scripts display messages on the front panel of the instrument that prompt the operator to enter parameters.

Run-time and nonvolatile memory storage of scripts

Scripts are loaded into the run-time environment of the instrument. From there, they can be stored in nonvolatile memory in the instrument.

The run-time environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to store a value while the instrument is turned on. When you create a script, the instrument creates a global variable with the same name so that you can reference the script more conveniently. After scripts are loaded into the run-time environment, you can run and manage them from the front panel of the instrument or from a computer. Information in the run-time environment is lost when the instrument is turned off.

Nonvolatile memory is where information is stored even when the instrument is turned off. Save scripts to nonvolatile memory to save them even if the power is cycled. The scripts that are in nonvolatile memory are loaded into the run-time environment when the instrument is turned on.

Scripts are placed in the run-time environment at the following times:

- When they are run.
- When they are loaded over a remote command interface.
- When the instrument is turned on (if they are stored in nonvolatile memory).

For detail on the amount of available memory, see Memory considerations for the run-time environment (on page 13-41).

NOTE

If you make changes to a script in the run-time environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See Saving a script to nonvolatile memory (on page 13-8).

What can be included in scripts?

Scripts can include combinations of Test Script Processor (TSP®) commands and Lua code. TSP commands instruct the instrument to do one thing and are described in the command reference (see TSP commands (on page 14-8)). Lua is a scripting language that is described in Fundamentals of programming for TSP (on page 13-12).

Working with scripts

This section describes the basics of working with scripts.

You can create and manage scripts from the front panel or over a remote interface. Scripts can be saved in the instrument, on a computer, or on a USB flash drive.

Tools for managing scripts

You can use any of the following tools to manage scripts:

- The front-panel menu options and USB flash drive. For information, refer to Saving setups (on page 3-45).
- Messages sent to the instrument. For information, see Load a script by sending commands over a remote interface (on page 13-7).
- Keithley Instruments Test Script Builder (TSB) software, which is available at tek.com/keithley. For more information, see Creating a new TSP project (on page 13-36).
- Your own development tool or program.
- The front-panel interface options in the Scripts menu. For information, refer to the following sections.

Script rules

You can have as many scripts as needed in the instrument. The only limitation is the amount of memory available to the run-time environment.

When a script is loaded into the run-time environment, a global variable with the same name as the script is created to reference the script.

Important points regarding scripts:

- Each script must have a unique name.
- Script names must not contain spaces.
- Script names must be less than 256 characters.
- If you load a new script with the same name as an existing script, an error event message is generated. You must delete the existing script before you create a new script with the same name.
- If you revise a script and save it to the instrument with a new name, the previously loaded script remains in the instrument with the original name.
- You can save scripts to nonvolatile memory in the instrument. Saving a script to nonvolatile memory allows the instrument to be turned off without losing the script. See Saving a script to nonvolatile memory (on page 13-8).

Loading a script into the instrument

You can load scripts from the front-panel display by copying them from a USB flash drive. You can also load them over a remote interface using `loadscript` commands.

If a script on a USB flash drive is named `autoinstall.tsp`, the script is automatically copied to the list of internal scripts when the drive is inserted into the instrument.

Loading a script using a USB flash drive

After loading a script onto a USB flash drive, you can copy the script using options on the front-panel display.

To load a script using a USB flash drive:

1. Insert the USB flash drive into the USB port on the front panel.
2. Press the **MENU** key.
3. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
4. In the USB Scripts list, select the script you want to copy from the USB flash drive.
5. Select **<**. The file is transferred to the instrument, and the corresponding file name is displayed in the Internal Scripts box.

Load a script by sending commands over a remote interface

To load a script over the remote interface, you can use the `loadscript` and `endscript` commands.

Normally, when the instrument receives a command, it runs the command immediately. When the instrument receives the `loadscript` command, the instrument starts collecting subsequent messages instead of running them immediately.

The `endscript` command tells the instrument to stop collecting messages. It then compiles the collection of messages into a script. The script is stored as a function. This script is loaded into the run-time environment — you need to save it to store it in the instrument.

To load a script:

Send the `loadscript` command with a script name. This tells the instrument to start collecting messages for the function named `testInfo`:

```
loadscript testInfo
```

Send the commands for the script; this example displays text on the USER swipe screen when the script is run:

```
display.settext(display.TEXT1, "Batch 233")
display.settext(display.TEXT2, "Test Information")
display.changescreen(display.SCREEN_USER_SWIPE)
```

Send the command that tells the instrument that the script is complete:

```
endscript
```

Run the script by sending the script name followed by `()`:

```
testInfo()
```

The USER swipe screen on the front panel is displayed and shows the text "Batch 233 Test Information" when you run this script.

To save the script to nonvolatile memory, send the command:

```
testInfo.save()
```

To load a script by sending commands:

1. Send the command `loadscript` *`scriptName`*, where *`scriptName`* is the name of the script. The name must be a legal Lua variable name.
2. Send the commands that need to be included in the script.
3. Send the command `endscript`.
4. You can now run the script. Send the script name followed by `()`. For more information, see Running scripts using a remote interface (on page 13-8).

Running scripts using the front-panel interface

To run a script from the Scripts menu:

1. Press the **MENU** key.
2. Under Scripts, select **Run**. The RUN SCRIPTS window is displayed.
3. From the Available Scripts list, select the script you want to run.
4. Select **Run Selected**.

To run a script from the home screen:

1. Press the **HOME** key.
2. Select the Script Indicator. The available scripts are displayed.
3. Select the script. A confirmation prompt is displayed.
4. Select **Yes** to run the script.

Running scripts using a remote interface

You can run any script using *`scriptVar`*`.run()`. Replace *`scriptVar`* with the name of a script that is in nonvolatile or run-time memory.

Saving a script to nonvolatile memory

You can save scripts to nonvolatile memory. To keep a script through a power cycle, you must save the script to nonvolatile memory.

To save a script to nonvolatile memory:

1. Create and load a script (see Working with scripts (on page 13-5)).
2. Send the command *`scriptVar`*`.save()`, where *`scriptVar`* is the name of the script.

Example: Save a user script to nonvolatile memory

`test1.save()`	Assume a script named `test1` has been loaded. `test1` is saved into nonvolatile memory.

Saving a script to a USB flash drive

You can save scripts to a USB flash drive.

To save a script to an external USB flash drive:

1. Load a script.
2. Insert a USB flash drive into the USB port on the front panel.
3. Send the command *`scriptVar`*`.save("/usb1/`*`filename`*`.tsp")`, where *`scriptVar`* is the variable referencing the script and *`filename`* is the name of the file.

Rename a script

To rename a script in the runtime environment:

1. Load the script into the runtime environment with a different name.
2. Delete the previous version of the script.

To rename a script in nonvolatile memory:

Send the commands:

```
scriptVar = script.load(file)
scriptVar.save()
```

Where:

- *`scriptVar`* is the name of variable that references the script
- *`file`* is the path and file name of the script file to load

For example, to load a script named `test8` from the USB flash drive and save it to nonvolatile memory, send the commands:

```
test8 = script.load("/usb1/test8.tsp")
test8.save()
```

NOTE

If the new name is the same as a name that is already used for a script, an event message is displayed, and the script is not saved.

Retrieve a user script from the instrument

You can review user scripts that are in the nonvolatile memory of the instrument and retrieve them.

To see a list of scripts from the front-panel interface:

1. Press the **MENU** key.
2. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.

The scripts are listed in the Internal Scripts list. To see the contents of the script, you can copy them to a USB flash drive. You can read the scripts with a text editor. See Saving a script to a USB flash drive (on page 13-9).

To retrieve the content of a script, use `scriptVar.source`, where `scriptVar` is the name of the script you want to retrieve. For example, to retrieve a script named `contactTest`, you would send:

```
print(contactTest.source)
```

The command is returned as a single string. The `loadscript` and `endscript` keywords are not included.

Deleting a user script using a remote interface

Deleting a user script deletes the script from the instrument.

To delete a script from the instrument:

Send the command:

```
script.delete("name")
```

Where: *name* is the user-defined name of the script.

Example: Delete a user script

`script.delete("test8")`	Delete a user script named `test8` from the instrument.

Power up script

The power up script runs automatically when the instrument is powered on. To create a power up script, save a new script and name it `autoexec`. The `autoexec` script is automatically saved to nonvolatile memory. See Saving a script to nonvolatile memory (on page 13-8).

NOTE

If an `autoexec` script already exists, you must delete it by sending the `script.delete("autoexec")` command. Performing a system reset does not delete the `autoexec` script.

To set up the power up script from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. Select **Copy to Power Up**. A dialog box confirms that the script was copied.
4. Select **OK**.

To save the power up script using remote commands:

Send the command:

```
autoexec.save()
```

To delete the existing `autoexec` script, send the command:

```
script.delete("autoexec")
```

Commands that cannot be used in scripts

You cannot use the following commands as variables in scripts:

NOTE

There are some functions that resemble some of the strings below but are actually defined TSP functions. For example, printbuffer() (on page 14-110) is a function you can use in scripts. If you are uncertain, check the TSP command reference (on page 14-1) to verify that the string is part of a defined function.

- `abort`
- `bit`
- `createconfigscript`
- `endflash`
- `endscript`
- `flash`
- `fs`
- `io`
- `loadscript`
- `loadandrunscript`
- `login`
- `logout`
- `node`
- `opc`
- `prevflash`
- `printbuffer`
- `printnumber`
- `scpi`
- `table`
- `waitcomplete`

Common commands that cannot be used in scripts are shown in the following table with equivalent commands that can be used.

Unavailable commands with TSP equivalents

Common commands	TSP equivalent commands
*CLS	eventlog.clear() status.clear()
*ESE	status.standard.enable
*ESE?	print(status.standard.enable)
*ESR?	print(status.standard.event)
*IDN?	print(localnode.model) print(localnode.serialno) print(localnode.version)
*LANG	No equivalent
*LANG?	No equivalent
*OPC	opc()
*OPC?	waitcomplete() print([[1]])
*RST	reset()
*SRE	status.request_enable
*SRE?	print(status.request_enable)
*STB?	print(status.condition)
*TRG	No equivalent
*TST?	print([[0]])
*WAI	waitcomplete()

Fundamentals of programming for TSP

To conduct a test, a computer (controller) is programmed to send sequences of commands to an instrument. The controller orchestrates the actions of the instrumentation. The controller is typically programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

To take advantage of the advanced features of the instrument, you can add programming commands to your scripts. Programming commands control script execution and provide tools such as variables, functions, branching, and loop control.

The Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands.

What is Lua?

Lua is a programming language that can be used with TSP-enabled instruments. Lua is an efficient language with simple syntax that is easy to learn.

Lua is also a scripting language, which means that scripts are compiled and run when they are sent to the instrument. You do not compile them before sending them to the instrument.

Lua basics

This section contains the basics about the Lua programming language to allow you to start adding Lua programming commands to your scripts quickly.

For more information about Lua, see the Lua website (lua.org). Another source of useful information is the Lua users group (lua-users.org), created for and by users of Lua programming language.

Comments

Comments start anywhere outside a string with a double hyphen (`--`). If the text immediately after a double hyphen (`--`) is anything other than double left brackets (`[[`), the comment is a short comment, which continues only until the end of the line. If double left brackets (`[[`) follow the double hyphen (`--[[`), it is a long comment, which continues until the corresponding double right brackets (`]]`) close the comment. Long comments may continue for several lines and may contain nested `[[ . . . ]]` pairs. The table below shows how to use code comments.

Using code comments

Type of comment	Comment delimiters	Usage	Example
Short comment	`--`	Use when the comment text fits on a single line.	`-- Turn off the front-panel display.` `display.lightstate = display.STATE_LCD_OFF`
Long comment	`--[[` `]]`	Use when the comment text is longer than one line.	`--[[Display a menu with three menu items. If the second menu item is selected, the selection will be given the value Test2.]]`

Function and variable name restrictions

You cannot use Lua reserved words and top-level command names for function or variable names.

Variable names must contain at least three characters.

The following table lists some of the Lua reserved words. If you attempt to assign these, the event code -285, "TSP Syntax error at line x: unexpected symbol near '*word*' " is displayed, where *word* is the Lua reserved word.

Lua reserved words		
`and`	`for`	`or`
`break`	`function`	`repeat`
`do`	`if`	`return`
`else`	`in`	`then`
`elseif`	`local`	`true`
`end`	`nil`	`until`
`false`	`not`	`while`

Values and variable types

In Lua, you use variables to store values in the run-time environment for later use.

Lua is a dynamically-typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been assigned a value.

Variable types

Variables can be one of the following types.

Variable types and values

Variable type returned	Value	Notes
`"nil"`	not declared	The type of the value `nil`, whose main property is to be different from any other value; usually it represents the absence of a useful value.
`"boolean"`	true or false	Boolean is the type of the values `false` and `true`. In Lua, both `nil` and `false` make a condition `false`; any other value makes it `true`.
`"number"`	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers. Hexadecimal and binary values are also handled as the number type in TSP.
`"string"`	sequence of words or characters	
`"function"`	a block of code	Functions perform a task or compute and return values.

Variable types and values

Variable type returned	Value	Notes
`"table"`	an array	New tables are created with { } braces. For example, {1, 2, 3.00e0}.
`"userdata"`	variables	Allows arbitrary program data to be stored in Lua variables.
`"thread"`	line of execution	

To determine the type of a variable, you can call the `type()` function, as shown in the examples below.

NOTE

The output you get from these examples may vary depending on the data format that is set.

Example: Nil

```
x = nil
print(x, type(x))
```

```
nil      nil
```

Example: Boolean

```
y = false
print(y, type(y))
```

```
false      boolean
```

Example: Hex constant

You can enter hexadecimal values, but to return a hexadecimal value, you must create a function, as shown in this example. Note that hexadecimal values are handled as a number type.

```
hex = function (i) return "0x"..string.format("%X", i) end
print(hex(0x54|0x55))
print(hex(0x54&0x66))
```

Set the format to return hexadecimal values, then `OR` two hexadecimal values and `AND` two hexadecimal values.
Output:

```
0x55
0x44
```

Example: Binary constant

Binary values are returned as floating-point decimal values. Note that binary values are handled as a number type.

```
x = 0b0000000011111111
y = 0B1111111100000000
print(x, type(x))
print(y, type(y))
```

```
255 number
65280 number
```

Example: String and number

```
x = "123"
print(x, type(x))

x = x + 7
print(x, type(x))
```

```
123      string


Adding a number to x forces its type to number.
130      number
```

Example: Function

```
function add_two(first_value,
   second_value)
   return first_value + second_value
end
print(add_two(3, 4), type(add_two))
```

```
7     function
```

Example: Table

```
atable = {1, 2, 3, 4}
print(atable, type(atable))
print(atable[1])
print(atable[4])
```

Defines a table with four numeric elements.
Note that the "table" value (shown here as a096cd30) will vary.

```
table: a096cd30      table
1
4
```

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the run-time environment.

Operators

You can compare and manipulate Lua variables and constants using operators.

Arithmetic operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
-	negation (for example, c = –a)
^	exponentiation

Relational operators

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~= !=	not equal
==	equal

Bitwise operators

Operator	Description
&	AND
|	OR
^^	exclusive OR
<<	bitwise shift left
>>	bitwise shift right
!	logical NOT

Logical and bitwise operators

The logical operators in Lua are `and`, `or`, and `not`. All logical operators consider both `false` and `nil` as false and anything else as true.

The operator `not` always returns `false` or `true`.

The conjunction operator `and` returns its first argument if the first argument is `false` or `nil`; otherwise, `and` returns its second argument. The disjunction operator `or` returns its first argument if this value is different from `nil` and `false`; otherwise, `or` returns its second argument. Both `and` and `or` use shortcut evaluation, that is, the second operand is evaluated only if necessary.

NOTE

The example output you get may vary depending on the data format settings of the instrument.

Example 1

```
print(10 or eventlog.next())
print(nil or "a")
print(nil and 10)
print(false and eventlog.next())
print(false and nil)
print(false or nil)
print(10 and 20)
```

Output:

```
10
a
nil
false
false
nil
20
```

Example 2

```
hex = function (i) return "0x"..string.format("%X", i) end
print(hex(0x54 | 0x55))
print(hex(0x54 & 0x66))
```

Set the format to return hexadecimal values, then OR two hexadecimal values and AND two hexadecimal values.
Output:
```
0x55
0x44
```

Example 3

```
hex = function (i) return "0x"..string.format("%X", i) end
a, b = 0b01010100, 0b01100110
print(hex(a), "&", hex(b), "=", hex(a & b))
```

Set the format to return hexadecimal values, define binary values for a and b, then AND a and b.
Output:
```
0x54 & 0x66 = 0x44
```

String concatenation

String operators

Operator	Description
..	Concatenates two strings. If either argument is a number, it is coerced to a string (in a reasonable format) before concatenation.

Example: Concatenation

```
print(2 .. 3)
print("Hello " .. "World")
```

Output:
```
23
Hello World
```

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority):

- ^ (exponentiation)
- not, - (unary), ! (logical NOT)
- *, /, <<, >>
- +, –, &, |, ^^
- .. (concatenation)
- <, >, <=, >=, ~=, !=, ==
- and
- or

You can use parentheses to change the precedences in an expression. The concatenation ("..") and exponentiation ("^") operators are right associative. All other binary operators are left associative. The examples below show equivalent expressions.

Equivalent expressions

`reading + offset < testValue/2+0.5`	=	`(reading + offset) < ((testValue/2)+0.5)`
`3+reading^2*4`	=	`3+((reading^2)*4)`
`Rdg < maxRdg and lastRdg <= expectedRdg`	=	`(Rdg < maxRdg) and (lastRdg <= expectedRdg)`
`-reading^2`	=	`-(reading^2)`
`reading^testAdjustment^2`	=	`reading^(testAdjustment^2)`

Functions

With Lua, you can group commands and statements using the `function` keyword. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple values.

You can use functions to form expressions that calculate and return a value. Functions can also act as statements that execute specific tasks.

Functions are first-class values in Lua. That means that functions can be stored in variables, passed as arguments to other functions, and returned as results. They can also be stored in tables.

Note that when a function is defined, it is stored in the run-time environment. Like all data that is stored in the run-time environment, the function persists until it is removed from the run-time environment, is overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or in Lua code in either of the following forms:

```
function myFunction(parameterX) functionBody end
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function.
- *parameterX*: Parameter names. To use multiple parameters, separate the names with commas.
- *functionBody*: The code that is executed when the function is called.

To execute a function, substitute appropriate values for *parameterX* and insert them into a message formatted as:

```
myFunction(valueForParameterX, valueForParameterY)
```

Where *valueForParameterX* and *valueForParameterY* represent the values to be passed to the function call for the given parameters.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example 1

```
function add_two(first_value, second_value)
   return first_value + second_value
end
print(add_two(3, 4))
```

Creates a variable named `add_two` that has a variable type of function.
Output:
7

Example 2

```
add_three = function(first_value,
   second_value, third_value)
   return first_value + second_value +
      third_value
end
print(add_three(3, 4, 5))
```

Creates a variable named `add_three` that has a variable type of function.
Output:
12

Example 3

```
function sum_diff_ratio(first_value,
   second_value)
   psum = first_value + second_value
   pdif = first_value - second_value
   prat = first_value / second_value
   return psum, pdif, prat
end
sum, diff, ratio = sum_diff_ratio(2, 3)
print(sum)
print(diff)
print(ratio)
```

Returns multiple parameters (sum, difference, and ratio of the two numbers passed to it).
Output:
5
-1
0.66666666666667

Create functions using scripts

You can use scripts to define functions. Scripts that define a function are like any other script: They do not cause any action to be performed on the instrument until they are executed. The global variable of the function does not exist until the script that created the function is executed.

A script can consist of one or more functions. Once a script has been run, the computer can call functions that are in the script directly.

For detail on creating functions, see Fundamentals of scripting for TSP (on page 13-4).

Conditional branching

Lua uses the `if`, `else`, `elseif`, `then`, and `end` keywords to do conditional branching.

Note that in Lua, `nil` and `false` are `false` and everything else is `true`. Zero (0) is `true` in Lua.

The syntax of a conditional block is as follows:

```
if expression then
   block
elseif expression then
   block
else
   block
end
```

Where:

- *expression* is Lua code that evaluates to either `true` or `false`
- *block* consists of one or more Lua statements

Example: If

```
if 0 then
   print("Zero is true!")
else
   print("Zero is false.")
end
```

Output:
```
Zero is true!
```

Example: Comparison

```
x = 1
y = 2
if x and y then
   print("Both x and y are true")
end
```

Output:
```
Both x and y are true
```

Example: If and else

```
x = 2
if not x then
   print("This is from the if block")
else
   print("This is from the else block")
end
```

Output:
```
This is from the else block
```

Example: Else and elseif

```
x = 1
y = 2
if x and y then
   print("'if' expression 2 was not false.")
end

if x or y then
   print("'if' expression 3 was not false.")
end

if not x then
   print("'if' expression 4 was not false.")
else
   print("'if' expression 4 was false.")
end

if x == 10 then
   print("x = 10")
elseif y > 2 then
   print("y > 2")
else
   print("x is not equal to 10, and y is not greater than 2.")
end
```

Output:

```
'if' expression 2 was not false.
'if' expression 3 was not false.
'if' expression 4 was false.
x is not equal to 10, and y is not greater than 2.
```

Loop control

If you need to repeat code execution, you can use the Lua `while`, `repeat`, and `for` control structures. To exit a loop, you can use the `break` keyword.

While loops

To use conditional expressions to determine whether to execute or end a loop, you use `while` loops. These loops are similar to Conditional branching (on page 13-20) statements.

```
while expression do
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: While

```
list = {
   "One", "Two", "Three", "Four", "Five", "Six"}
print("Count list elements on numeric index:")
element = 1
while list[element] do
   print(element, list[element])
   element = element + 1
end
```

This loop exits when `list[element] = nil`.

Output:

```
Count list elements on
   numeric index:
1   One
2   Two
3   Three
4   Four
5   Five
6   Six
```

Repeat until loops

To repeat a command, you use the `repeat` ... `until` statement. The body of a `repeat` statement always executes at least once. It stops repeating when the conditions of the `until` clause are met.

```
repeat
   block
until expression
```

Where:

- *`block`* consists of one or more Lua statements
- *`expression`* is Lua code that evaluates to either `true` or `false`

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Repeat until

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
print("Count elements in list using repeat:")
element = 1
repeat
   print(element, list[element])
   element = element + 1
until not list[element]
```

Output:

```
Count elements in list
   using repeat:
1  One
2  Two
3  Three
4  Four
5  Five
6  Six
```

For loops

There are two variations of `for` statements supported in Lua: Numeric and generic.

NOTE

In a `for` loop, the loop expressions are evaluated once, before the loop starts.

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Numeric for

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
---------- For loop -----------
print("Counting from one to three:")
for element = 1, 3 do
   print(element, list[element])
end
print("Counting from one to four, in steps of two:")
for element = 1, 4, 2 do
   print(element, list[element])
end
```

The numeric `for` loop repeats a block of code while a control variable runs through an arithmetic progression.

Output:

```
Counting from one to three:
1   One
2   Two
3   Three
Counting from one to four, in steps of two:
1   One
3   Three
```

Example: Generic for

```
days = {"Sunday",
   "Monday",     "Tuesday",
   "Wednesday",  "Thursday",
   "Friday",     "Saturday"}

for i, v in ipairs(days) do
   print(days[i], i, v)
end
```

The generic `for` statement works by using functions called iterators. On each iteration, the iterator function is called to produce a new value, stopping when this new value is nil.
Output:

```
Sunday     1    Sunday
Monday     2    Monday
Tuesday    3    Tuesday
Wednesday  4    Wednesday
Thursday   5    Thursday
Friday     6    Friday
Saturday   7    Saturday
```

Break

The `break` statement can be used to terminate the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A break ends the innermost enclosing loop.

Return and break statements can only be written as the last statement of a block. If it is necessary to return or break in the middle of a block, an explicit inner block can be used.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Break with while statement

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a `break` value (`breakValue`) so that the `break` statement is used to exit the `while` loop before the value of k reaches `0`.
Output:

```
Going to break and k = 3
```

Example: Break with while statement enclosed by comment delimiters

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
-- local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a break value (`breakValue`), but the break value line is preceded by comment delimiters so that the break value is not assigned, and the code reaches the value `0` to exit the `while` loop.

Output:

```
Break value not found
```

Example: Break with infinite loop

```
a, b = 0, 1
while true do
   print(a, b)
   a, b = b, a + b
   if a > 500 then
      break
   end
end
```

This example uses a `break` statement that causes the while loop to exit if the value of a becomes greater than 500.

Output:

```
0     1
1     1
1     2
2     3
3     5
5     8
8     13
13    21
21    34
34    55
55    89
89    144
144   233
233   377
377   610
```

Tables and arrays

Lua makes extensive use of the data type table, which is a flexible array-like data type. Table indices start with 1. Tables can be indexed not only with numbers, but with any value except `nil`. Tables can be heterogeneous, which means that they can contain values of all types except `nil`.

Tables are the sole data structuring mechanism in Lua. They may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, and so on. To represent records, Lua uses the field `name` as an index. The language supports this representation by providing `a.name` as an easier way to express `a["name"]`.

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Loop array

```
atable = {1, 2, 3, 4}
i = 1
while atable[i] do
   print(atable[i])
   i = i + 1
end
```

Defines a table with four numeric elements. Loops through the array and prints each element.
The Boolean value of `atable[index]` evaluates to `true` if there is an element at that index. If there is no element at that index, `nil` is returned (`nil` is considered to be `false`).

Output:

```
1
2
3
4
```

Standard libraries

In addition to the standard programming constructs described in this document, Lua includes standard libraries that contain useful functions for string manipulation, mathematics, and related functions. Test Script Processor (TSP®) scripting engine instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP scripting engine. These libraries are automatically loaded when the TSP scripting engine starts and do not need to be managed by the programmer.

The following topics provide information on some of the basic Lua standard libraries. For additional information, see the Lua website (lua.org).

NOTE

When referring to the Lua website, please be aware that the TSP scripting engine uses Lua 5.0.2.

Base library functions

Base library functions

Function	Description
`collectgarbage()` `collectgarbage(`*`limit`*`)`	Sets the garbage-collection threshold to the given limit (in kilobytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, Lua immediately runs the garbage collector. If there is no limit parameter, it defaults to zero (0), which forces a garbage-collection cycle. See Lua memory management (on page 13-28) for more information.
`gcinfo()`	Returns the number of kilobytes of dynamic memory that the Test Script Processor (TSP®) scripting engine is using and returns the present garbage collector threshold (also in kilobytes). See Lua memory management (on page 13-28) for more information.
`tonumber(`*`x`*`)` `tonumber(`*`x, base`*`)`	Returns *x* converted to a number. If *x* is already a number, or a convertible string, the number is returned; otherwise, it returns `nil`. An optional argument specifies the base to use when interpreting the numeral. The base may be any integer from 2 to 36, inclusive. In bases above 10, the letter A (in either upper or lower case) represents 10, B represents 11, and so forth, with Z representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(`*`x`*`)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(`*`v`*`)`	Returns (as a string) the type of its only argument. The possible results of this function are `"nil"` (a string, not the value `nil`), `"number"`, `"string"`, `"boolean"`, `"table"`, `"function"`, `"thread"`, and `"userdata"`.

Lua memory management

Lua automatically manages memory, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua occasionally runs a garbage collector to collect all objects that are no longer accessible from Lua. All objects in Lua are subject to automatic management, including tables, variables, functions, threads, and strings.

Lua uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory Lua is using; the other is a threshold. When the number of bytes crosses the threshold, Lua runs the garbage collector, which reclaims the memory of all inaccessible objects. The byte counter is adjusted, and the threshold is reset to twice the new value of the byte counter.

String library functions

This library provides generic functions for string manipulation, such as finding and extracting substrings. When indexing a string in Lua, the first character is at position 1 (not 0, as in ANSI C). Indices may be negative and are interpreted as indexing backward from the end of the string. Thus, the last character is at position –1, and so on.

String library functions

Function	Description

`string.byte(s)`
`string.byte(s, i)`
`string.byte(s, i, j)`
	Returns the internal numeric codes of the characters `s[i], s[i+1], ···, s[j]`. The default value for `i` is `1`; the default value for `j` is `i`.

`string.char(···)`	Receives zero or more integers separated by commas. Returns a string with length equal to the number of arguments, in which each character has the internal numeric code equal to its corresponding argument.

`string.format(`
`  formatstring, ...)`
	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the modifiers `*`, `l`, `L`, `n`, `p`, and `h` are not supported and there is an extra option, `q`. The `q` option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written.
	For example, the call:
	`string.format('%q', 'a string with "quotes" and \n new line')`
	will produce the string:
	`"a string with \"quotes\" and \ new line"`
	The options `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `X`, and `x` all expect a number as argument. `q` and `s` expect a string. This function does not accept string values containing embedded zeros, except as arguments to the `q` option.

`string.len(s)`	Receives a string and returns its length. The empty string `""` has length `0`. Embedded zeros are counted, so `"a\000bc\000"` has length `5`.
`string.lower(s)`	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
`string.rep(s, n)`	Returns a string that is the concatenation of *n* copies of the string `s`.
`string.sub(s, i)` `string.sub(s, i, j)`	Returns the substring of `s` that starts at `i` and continues until `j`; `i` and `j` can be negative. If `j` is absent, it is assumed to be equal to `-1` (which is the same as the string length). In particular, the call `string.sub(s, 1, j)` returns a prefix of `s` with length `j`, and `string.sub(s, -i)` returns a suffix of `s` with length `i`.
`string.upper(s)`	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions

Function	Description
`math.abs(x)`	Returns the absolute value of *x*.
`math.acos(x)`	Returns the arc cosine of *x*.
`math.asin(x)`	Returns the arc sine of *x*.
`math.atan(x)`	Returns the arc tangent of *x*.
`math.atan2(y, x)`	Returns the arc tangent of *y/x* but uses the signs of both parameters to find the quadrant of the result (it also handles correctly the case of *x* being zero).
`math.ceil(x)`	Returns the smallest integer larger than or equal to *x*.
`math.cos(x)`	Returns the cosine of *x*.
`math.deg(x)`	Returns the angle *x* (given in radians) in degrees.
`math.exp(x)`	Returns the value e^x.
`math.floor(x)`	Returns the largest integer smaller than or equal to *x*.
`math.frexp(x)`	Returns `m` and e such that $x = m2^e$, where e is an integer and the absolute value of `m` is in the range `[0.5, 1]` (or zero when *x* is zero).
`math.ldexp(m, e)`	Returns $m2^e$ (e should be an integer).
`math.log(x)`	Returns the natural logarithm of *x*.
`math.log10(x)`	Returns the base-10 logarithm of *x*.
`math.max(x, ...)`	Returns the maximum value among its arguments.
`math.min(x, ...)`	Returns the minimum value among its arguments.
`math.pi`	The value of π (3.141592654).
`math.pow(x, y)`	Returns x^y (you can also use the expression `x^y` to compute this value).
`math.rad(x)`	Returns the angle *x* (given in degrees) in radians.

`math.random()`
`math.random(m)`
`math.random(m, n)`
	This function is an interface to the simple pseudorandom generator function `rand` provided by ANSI C.
	When called without arguments, returns a uniform pseudorandom real number in the range `[0,1]`. When called with an integer number *m*, `math.random()` returns a uniform pseudorandom integer in the range `[1, m]`. When called with two integer numbers *m* and *n*, `math.random()` returns a uniform pseudorandom integer in the range `[m, n]`.

`math.randomseed(x)`	Sets *x* as the seed for the pseudorandom generator: equal seeds produce equal sequences of numbers.
`math.sin(x)`	Returns the sine of *x*.
`math.sqrt(x)`	Returns the square root of *x*. (You can also use the expression `x^0.5` to compute this value.)
`math.tan(x)`	Returns the tangent of *x*.

Test Script Builder (TSB)

Keithley Instruments Test Script Builder (TSB) is a software tool you can use to develop scripts for TSP-enabled instruments.

You must use the TSP command set to use Test Script Builder. Refer to Determining the command set you will use (on page 2-36) for information about the command sets and changing them.

Installing the TSB software

The installation files for the Test Script Builder software are available at tek.com/keithley.

To install the Test Script Builder (TSB) software:

1. Close all programs.
2. Download the installer to your computer and double-click the `.exe` file to start the installation.
3. Follow the on-screen instructions.

Installing the TSB add-in

When you install the Test Script Builder Software Suite, all available updates for TSB Add-in software are also installed. This includes any additional tools for the Test Script Builder (TSB) and model-specific examples and help files (see Installing the TSB software (on page 13-31)). If you have an existing version of TSB that does not have model-specific examples for an instrument you are using, you can download a separate add-in from the Keithley Instruments support website (tek.com/support).

Before installing the TSB Add-in software, you must install the TSB software.

To install the TSB Add-in software:

1. Close all programs.
2. Download the Add-in to your computer and double-click it to start installation.
3. Follow the on-screen instructions.

Using Test Script Builder (TSB)

Keithley Instruments Test Script Builder (TSB) is a software tool that simplifies building test scripts. You can use TSB to perform the following operations:

- Send remote commands and Lua statements
- Receive responses (data) from commands and scripts
- Upgrade instrument firmware
- Create, manage, and run user scripts
- Debug scripts
- Import factory scripts to view or edit and convert to user scripts

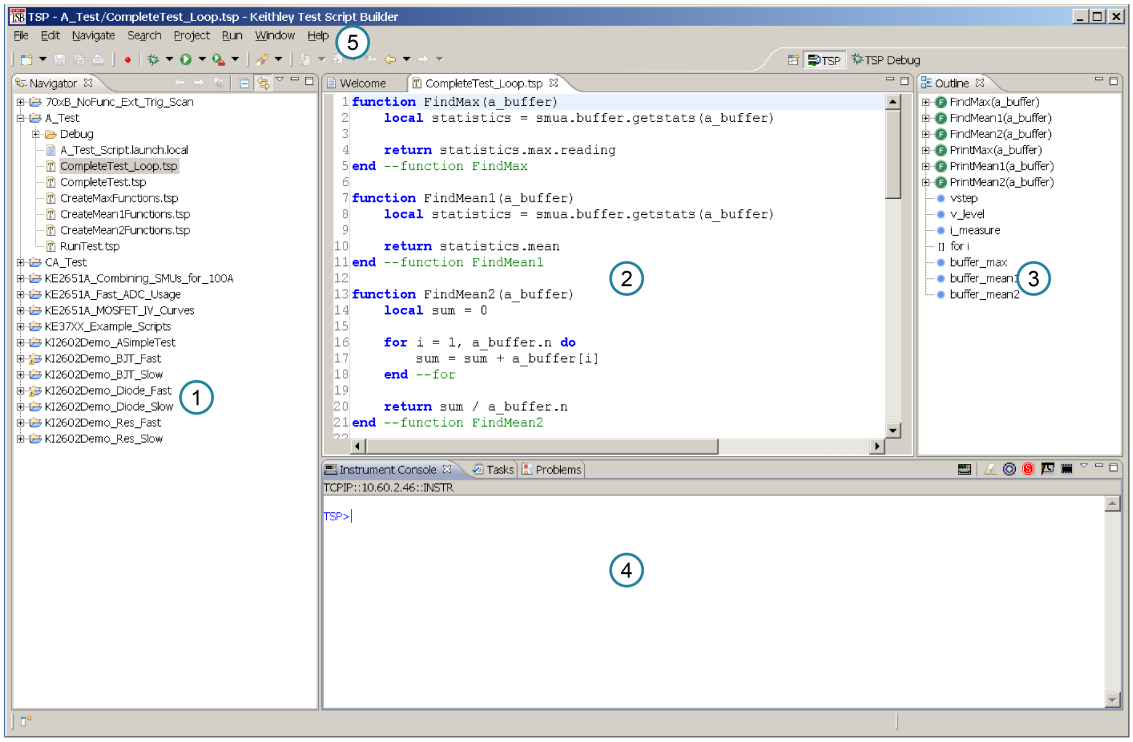
The Keithley Instruments Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands. For more information about using the Lua scripting language with Keithley TSP-enabled instruments, refer to the Fundamentals of programming for TSP (on page 13-12).

Keithley has created a collection of remote commands specifically for use with Keithley TSP-enabled instruments; for detailed information about those commands, refer to the "Command reference" section of the documentation for your specific instrument. You can build scripts from a combination of these commands and Lua programming statements. Scripts that you create are referred to as "user scripts." Also, some TSP-enabled instruments include built-in factory scripts.

The following figure shows an example of the Test Script Builder. As shown, the workspace is divided into these areas:

- Project navigator
- Script editor
- Outline view
- Programming interaction
- Help files

Figure 141: Example of the Test Script Builder workspace



Item	Description
1	Project navigator
2	Script editor; right-click to run the script that is displayed
3	Outline view
4	Programming interaction
5	Help; includes detailed information on using Test Script Builder

Project navigator

The project navigator consists of project folders and the script files (.tsp) created for each project. Each project folder can have one or more script files.

To view the script files in a project folder, select the plus (+) symbol next to the project folder. To hide the folder contents, select the minus (–) symbol next to the project folder.

You can download a TSP project to the instrument and run it, or you can run it from the TSB interface.

Script editor

The script editor is where you write, modify, and debug scripts.

To open and display a script file, double-click the file name in the project navigator. You can have multiple script files open in the script editor at the same time. Each open script file is displayed on a separate tab.

To display another script file that is already open, select the tab that contains the script in the script editor area.

Outline view

The outline view allows you to navigate through the structure of the active script in the script editor. Double-clicking a variable name or icon causes the first instance of the variable in the active script to be highlighted.

This view shows:

- Names of local and global variables
- Functions referenced by the active script in the script editor
- Parameters
- Loop control variables
- Table variables
- Simple assignments to table fields

Programming interaction

This part of the workspace is where you interact with the scripts that you are building in Test Script Builder (TSB). The actual contents of the programming interaction area of the workspace can vary.

You can send commands from the Instrument Console command line, retrieve data, view variables and errors, and view and set breakpoints when using the debug feature.

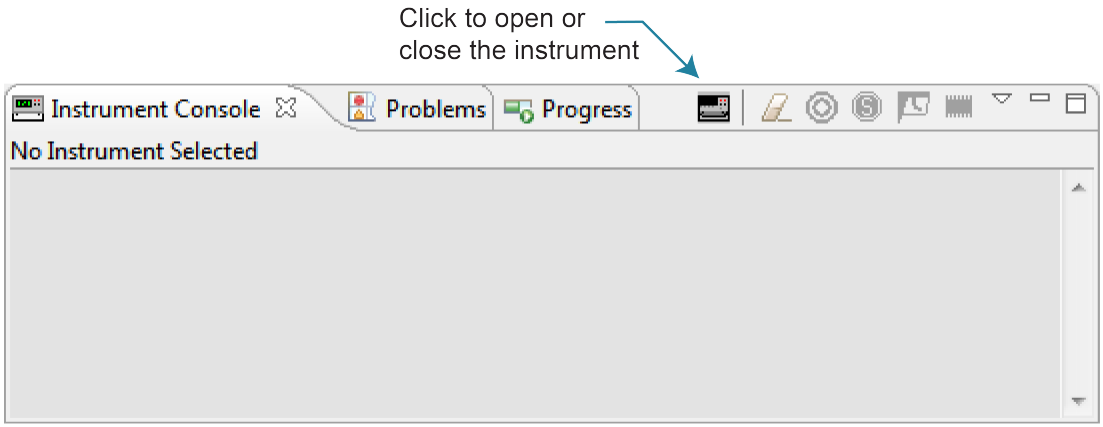
Connecting an instrument in TSB

You must use the TSP command set with the Test Script Builder software. For information on changing the command set, refer to Determining the command set you will use (on page 2-36).

To connect the Test Script Builder software to an instrument:

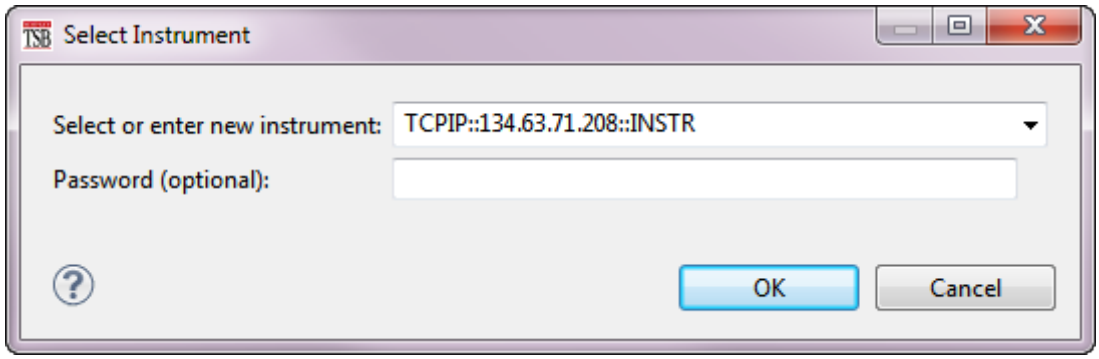
1. Select the **Open Instrument** icon in the script editor toolbar.

Figure 142: Opening an instrument connection in TSB



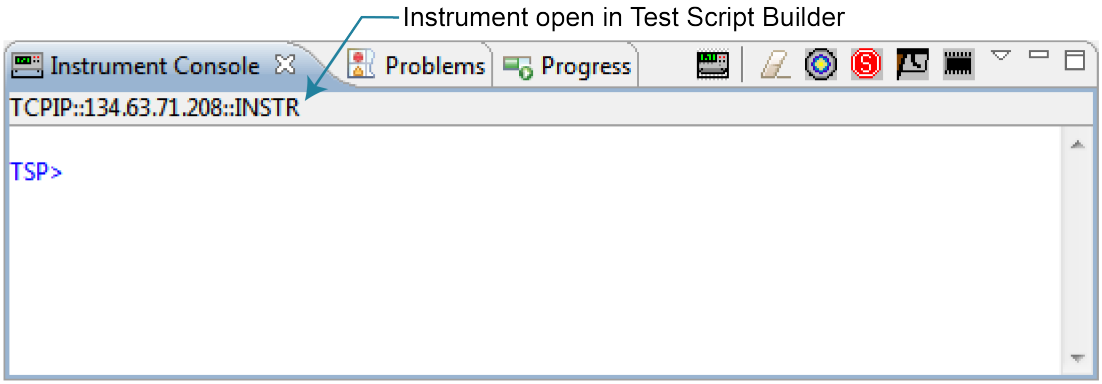
2. The Select Instrument dialog box opens. Select an existing instrument from the list or type the VISA resource ID of the instrument in the **Select or enter new instrument** box.
3. If needed, enter a password.

Figure 143: Select Instrument dialog box



4. Select **OK**. You briefly see the Opening Resource dialog box, and then the instrument is visible in the Instrument Console.

Figure 144: Instrument connected in TSB

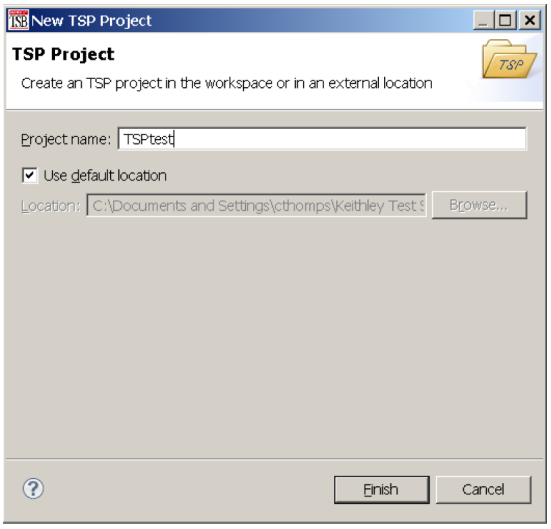


Creating a new TSP project

To create a new Test Script Processor (TSP®) project:

1. On the **File** menu in the TSP perspective, select **New > TSP Project**. The New TSP Project dialog box opens.

Figure 145: New TSP Project dialog box



2. Type a name for your project in the **Project name** box.
3. Select the location to create the new project.
4. Select **Finish**. The new project appears in the list of projects in the project navigator, and a file named `main.tsp` is created in the project. You can rename the `.tsp` file.
5. If you do not want to build your project automatically when it is saved or run, from the **Project** menu, clear **Build Automatically**.

NOTE

If you make changes to your project and do not build it before you run it, the Problems tab may not appear when problems are encountered.

Adding a new TSP file to a project

To add a new TSP file to a project:

1. Select the **File** menu and select **New > TSP File**. The New TSP File dialog box opens.
2. Select the project folder where you want to save the file.
3. Enter a name in the **File name** box.
4. Select **Finish**.

Running a script

You can run a script in the Test Script Builder (TSB) software using any of the following methods:

- Run a script that is open in the script editor area
- Run scripts that are listed in the Navigator area that are not currently open in the script editor window
- Run a collection of scripts by creating a run configuration (see Creating a run configuration (on page 13-38))

NOTE

When you use any of the run controls to run a script, the area that has focus in the workspace is important. For example, if the Navigator area is active (the tab is shaded) when you select the **Run** icon, the script file that is highlighted in the Navigator area is run instead of the active script in the script editor area.

The following list describes the most commonly used controls to run scripts in TSB:

- Right-click in the script editor area and select **Run Editor Contents** to run the active script as it currently appears in the script editor
- Right-click in the script editor area and select **Run As > 1 TSP File** to run the last saved version of the active script in the script editor as a `.tsp` file
- Select an action from the **Run** menu at the top of the TSB software interface

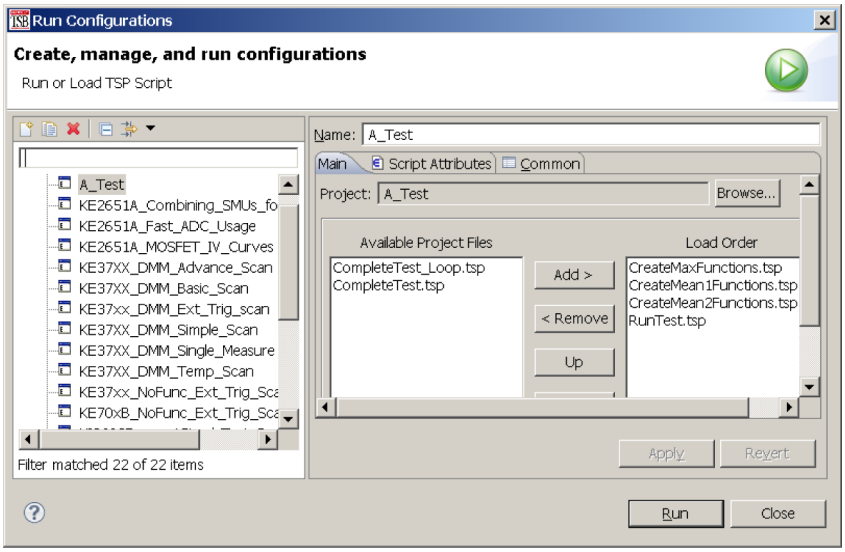
Creating a run configuration

A run configuration allows you to download multiple script files to an instrument and execute them as a single script.

To create a run configuration:

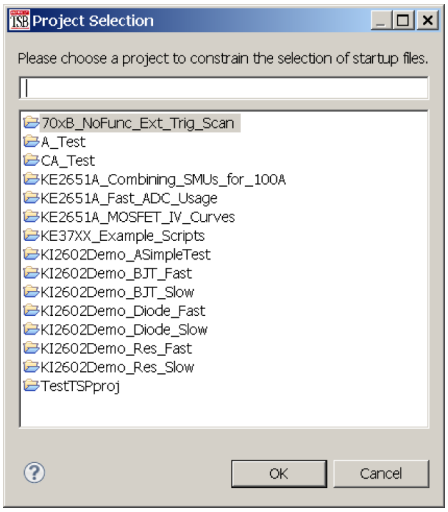
1. On the **Run** menu, select **Run Configurations**. The Run Configurations dialog box opens.
2. The left pane of the dialog box lists existing run and debug configurations. Select the script where the Run Configuration will be saved.
3. Select the **New launch configuration** icon at the top left of the dialog box. By default, a new configuration is created with the name `New_configuration`.

Figure 146: Run Configurations dialog box



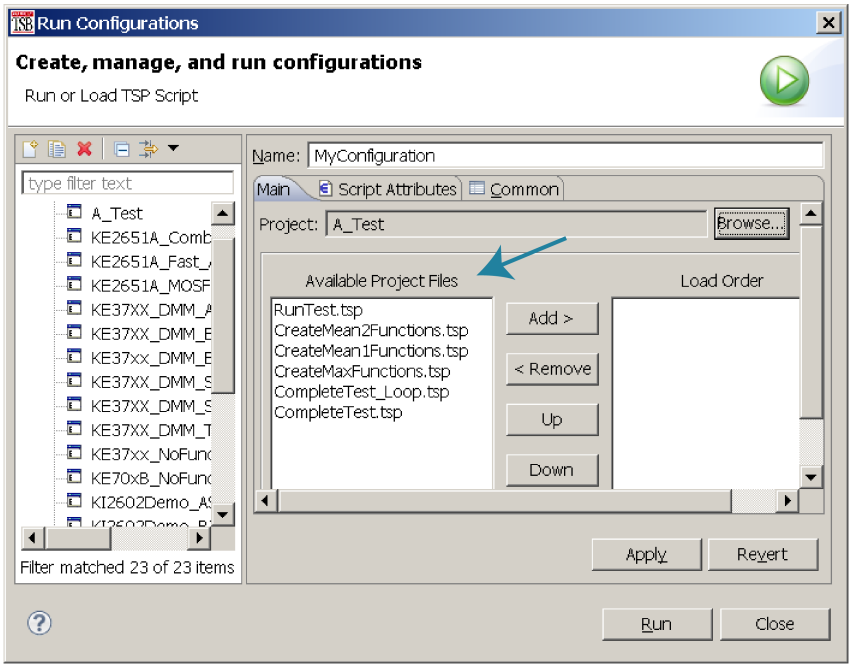
4. In the **Name** box, enter the name of your new run configuration.
5. Select the **Browse** button next the Project box.
6. Select a project from the list of available projects
7. Select **OK**.

Figure 147: Project Selection dialog box



The TSP files for the selected project are added to the Available Project Files list on the Main tab.

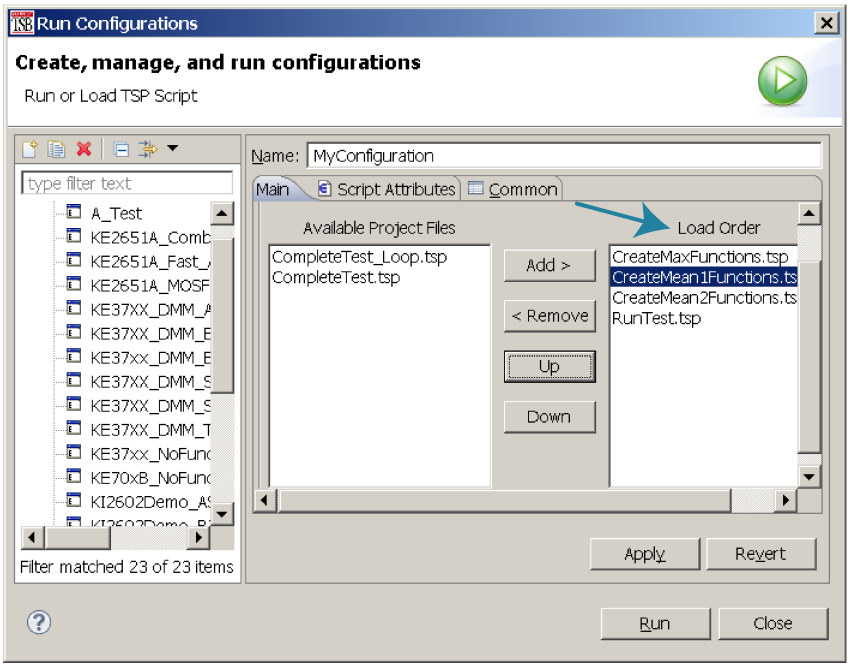
Figure 148: Available files for selected project



8. Select the files you want to add to the run configuration and select **Add** to add them to the Load Order list.

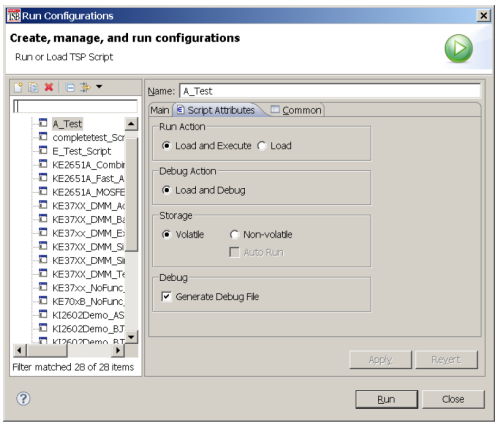
   To change the load order of the TSP files, choose the files you want to move and select **Up** or **Down** until the files are in the correct order.

Figure 149: Selected TSP files load order



9. Select **Apply**.
10. Select the **Script Attributes** tab.
11. Select one of the following:
    - **Load and Execute:** If you select this option, which is the default selection, the script automatically loads into the instrument's volatile memory (run-time environment) and executes when you select **Run**.
    - **Load:** If you select this option, the script is loaded into the instrument's volatile memory when you select **Run** but is not executed until you manually run it. To manually run it from the command line in the Instrument Console, type *`MyConfiguration`*`.run()` (where *`MyConfiguration`* is the name of your configuration).

Figure 150: Script Attributes tab



12. In the Storage area of the Script Attributes tab, select **Volatile** or **Non-volatile**. For products that support autorun scripts, if you select Non-volatile, you can select **Auto Run** to have the script run automatically when the instrument is turned on.
    Note that all scripts are initially stored in the volatile (runtime) memory of the instrument memory and are lost if you turn the instrument power off and then on again. If you want to keep the script on the instrument through a power cycle, select **Non-volatile** storage.
13. In the Debug area of the Script Attributes tab, you can select **Generate Debug File**.
    When you select this option, a Debug subfolder is created in your test folder, and a file with a `.DBG` extension is created in that folder. Note that this is a feature of the Eclipse platform, and you will not use this file to debug your script. It contains all the scripts in your run configuration, so that you can see them together in the order in which they will load.
14. Select **Close** or **Run**. The run configuration is added to the run configurations list.

NOTE

To run the last used run configuration, select the **Run** icon  in the main TSB toolbar. To run a different run configuration, right-click in the script editor area and select **Run As > Run Configurations**. Select a different run configuration, and then select **Run** in the Run Configurations dialog box.

Memory considerations for the run-time environment

The 2470 reserves a large amount of memory for use with interactions with the front panel, commands, and test scripts. The amount of memory usage is affected by the following product features:

- Reading buffers (including local default and user-created reading buffers; if they are on a remote node, they only affect the remote node); refer to Setting reading buffer capacity (on page 6-9) for additional information.
- Scripts that are loaded onto the instrument.
- Applications that are loaded onto the instrument.
- Configuration lists.
- Scripts that are actively running.
- Variables that reside in the run-time environment.
- The TriggerFlow trigger model.
- USB drives that contain files that use a lot of memory, even if the files are not related to instrument operation.

The more a feature is used or the larger its definition, the more memory it consumes. For normal usage, reading buffers commonly reserve large amounts of memory. The amount of memory used depends on the number of readings and the buffer style.

CAUTION

The 2470 notifies you when the system runs out of memory. If the instrument encounters memory allocation errors (errors that specifically state "Out of Memory"), the state of the instrument cannot be guaranteed. After attempting to save any important data, turn off power to the instrument and turn it back on to reset the runtime environment and return the instrument to a known state. Unsaved scripts and reading buffers will be lost.

If you encounter memory problems, examine the test script or SCPI commands that were being executed when the memory problems occurred. Take action to reduce the size of the elements that are consuming memory. If you are using TSP commands and scripting, also consider using the `collectgarbage()` command to clean up unused memory. For information on `collectgarbage()`, refer to Base library functions (on page 13-28).

The default size settings for the default reading buffers (`defbuffer1` and `defbuffer2`) are large. If your application does not use these buffers, you can set them to the minimum of 10 readings to conserve space. For information on adjusting the buffer size, refer to Setting reading buffer capacity (on page 6-9).

The buffer style is set when you create a user-defined reading buffer. The buffer style cannot be changed after the buffer has been created, so to eliminate memory problems caused by the style, you may need to delete or adjust the capacity of the buffers. Refer to Creating buffers (on page 6-5) for information on the effects of styles.

The amount of memory used by a sweep configuration is based on the number of source points. The actual memory consumption can vary greatly depending on how often the source-measure unit (SMU) take readings during the sweep, but as a general rule, each source point can be expected to consume at least 24 bytes.

It is possible for the memory used for the run-time environment, sweep configuration, and reading buffers to exceed 32 MB. When this occurs, there is a risk that memory allocation errors will occur and instrument will not execute commands as expected.

About TSP Commands

This section contains an overview of the TSP commands for the instrument. The commands are organized into groups, with a brief description of each group. Each section contains links to the detailed descriptions for each command in the TSP command reference section of this documentation (see TSP commands (on page 14-8)).

Beeper control

The beeper command allows you to sound the instrument beeper:

beeper.beep() (on page 14-8)

Digital I/O

The digital I/O port of the instrument can control external circuitry (such as a component handler for binning operations).

The I/O port has 6 lines. Each line can be at TTL logic state 1 (high) or 0 (low). See the pinout diagram in Digital I/O port configuration (on page 8-14) for additional information.

You can use these commands to read and write to each individual bit, and to read and write to the entire port.

digio.line[N].mode (on page 14-58)
digio.line[N].reset() (on page 14-59)
digio.line[N].state (on page 14-61)
digio.readport() (on page 14-61)
digio.writeport() (on page 14-62)

Configuration list

You can use these commands to create and recall configuration lists. A configuration list is a list of stored settings for the source or measurement function. You can restore these settings to change the active state of the instrument.

The measure configuration list commands are:

smu.measure.configlist.catalog() (on page 14-127)
smu.measure.configlist.create() (on page 14-128)
smu.measure.configlist.delete() (on page 14-129)
smu.measure.configlist.query() (on page 14-130)
smu.measure.configlist.recall() (on page 14-131)
smu.measure.configlist.size() (on page 14-132)
smu.measure.configlist.store() (on page 14-133)

The source configuration list commands are:

smu.source.configlist.catalog() (on page 14-172)
smu.source.configlist.create() (on page 14-172)
smu.source.configlist.delete() (on page 14-173)
smu.source.configlist.query() (on page 14-174)
smu.source.configlist.recall() (on page 14-175)
smu.source.configlist.size() (on page 14-176)
smu.source.configlist.store() (on page 14-177)

Display

The display functions and attributes allow you to change the format of information on the front-panel display of the instrument. You can also customize your display.

display.changescreen() (on page 14-64)
display.clear() (on page 14-65)
display.delete() (on page 14-65)
display.input.number() (on page 14-66)
display.input.option() (on page 14-68)
display.input.prompt() (on page 14-69)
display.input.string() (on page 14-70)
display.lightstate (on page 14-72)
display.prompt() (on page 14-73)
display.readingformat (on page 14-74)
display.settext() (on page 14-75)
display.waitevent() (on page 14-76)

Event log

You can use the event log to view specific details about LAN triggering events.

eventlog.clear() (on page 14-77)
eventlog.getcount() (on page 14-77)
eventlog.next() (on page 14-78)
eventlog.post() (on page 14-80)
eventlog.save() (on page 14-81)

File

You can use the file commands to open and close directories and files, write data, or to read a file on an installed USB flash drive.

The root folder of the USB flash drive has the absolute path:

```
"/usb1/"
```

NOTE

You can use either the slash (/) or backslash (\) as a directory separator. However, the backslash is also used as an escape character, so if you use it as a directory separator, you will generally need to use a double backslash (\\) when you are creating scripts or sending commands to the instrument.

file.close() (on page 14-82)
file.flush() (on page 14-83)
file.mkdir() (on page 14-84)
file.open() (on page 14-85)
file.read() (on page 14-86)
file.usbdriveexists() (on page 14-86)
file.write() (on page 14-88)

Instrument identification

These commands store strings that describe the instrument.

localnode.access (on page 14-99)
localnode.gettime() (on page 14-100)
localnode.linefreq (on page 14-100)
localnode.model (on page 14-101)
localnode.password (on page 14-101)
localnode.prompts (on page 14-102)
localnode.prompts4882 (on page 14-103)
localnode.serialno (on page 14-103)
localnode.settime() (on page 14-104)
localnode.showevents (on page 14-105)
localnode.version (on page 14-106)

Miscellaneous

The miscellaneous functions and attributes allow you to control instrument features and functions that are not category specific.

delay() (on page 14-57)
exit() (on page 14-82)
localnode.linefreq (on page 14-100)
localnode.password (on page 14-101)
localnode.prompts (on page 14-102)
localnode.prompts4882 (on page 14-103)
opc() (on page 14-108)
waitcomplete() (on page 14-319)
upgrade.previous() (on page 14-315)
upgrade.unit() (on page 14-316)

LAN

LAN commands provide options that allow you to review and configure network settings over the remote interface.

The following commands contain the status of the LAN.

lan.ipconfig() (on page 14-97)
lan.lxidomain (on page 14-98)
lan.macaddress (on page 14-98)

Set up and assert trigger events that are sent over the LAN.

trigger.lanin[N].clear() (on page 14-230)
trigger.lanin[N].edge (on page 14-231)
trigger.lanin[N].overrun (on page 14-231)
trigger.lanin[N].wait() (on page 14-232)
trigger.lanout[N].assert() (on page 14-233)
trigger.lanout[N].connect() (on page 14-234)
trigger.lanout[N].connected (on page 14-234)
trigger.lanout[N].disconnect() (on page 14-235)
trigger.lanout[N].ipaddress (on page 14-236)
trigger.lanout[N].logic (on page 14-236)
trigger.lanout[N].protocol (on page 14-237)
trigger.lanout[N].stimulus (on page 14-238)

GPIB

These commands store the GPIB address and indicate whether GPIB communication is enabled.

gpib.address (on page 14-96)

Reading buffer

Reading buffers capture measurements, ranges, instrument status, and output states of the instrument.

buffer.clearstats() (on page 14-9)
buffer.delete() (on page 14-10)
buffer.getstats() (on page 14-11)
buffer.make() (on page 14-13)
buffer.save() (on page 14-18)
buffer.saveappend() (on page 14-19)
bufferVar.capacity (on page 14-22)
bufferVar.clear() (on page 14-23)
bufferVar.dates (on page 14-24)
bufferVar.endindex (on page 14-25)
bufferVar.fillmode (on page 14-30)
bufferVar.formattedreadings (on page 14-31)
bufferVar.fractionalseconds (on page 14-32)
bufferVar.logstate (on page 14-33)
bufferVar.n (on page 14-33)
bufferVar.readings (on page 14-34)
bufferVar.relativetimestamps (on page 14-35)
bufferVar.seconds (on page 14-36)
bufferVar.sourceformattedvalues (on page 14-37)
bufferVar.sourcestatuses (on page 14-38)
bufferVar.sourceunits (on page 14-40)
bufferVar.sourcevalues (on page 14-41)
bufferVar.startindex (on page 14-42)

bufferVar.statuses (on page 14-43)
bufferVar.times (on page 14-44)
bufferVar.timestamps (on page 14-45)
bufferVar.units (on page 14-47)
buffer.write.format() (on page 14-48)
buffer.write.reading() (on page 14-50)
printbuffer() (on page 14-110)

Reset

Resets settings to their default settings.

digio.line[N].reset() (on page 14-59)
reset() (on page 14-114)
smu.reset() (on page 14-169)
timer.cleartime() (on page 14-216)
trigger.blender[N].reset() (on page 14-219)
trigger.timer[N].reset() (on page 14-283)
tsplink.line[N].reset() (on page 14-298)
tspnet.reset() (on page 14-309)

Queries and response messages

You can use the `print()`, `printbuffer()`, and `printnumber()` functions to query the instrument and generate response messages. The format attributes control how the data is formatted for the print functions used.

The localnode commands determine if generated errors are automatically sent and if prompts are generated.

format.asciiprecision (on page 14-88)
format.byteorder (on page 14-89)
format.data (on page 14-90)
localnode.serialno (on page 14-103)
localnode.settime() (on page 14-104)
localnode.showevents (on page 14-105)
localnode.version (on page 14-106)
print() (on page 14-109)
printbuffer() (on page 14-110)
printnumber() (on page 14-113)

Scripting

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument efficiently. These commands describe how to create, load, modify, run, and exit scripts.

For detail on using scripts, see Fundamentals of scripting for TSP (on page 13-4).

createconfigscript() (on page 14-52)
exit() (on page 14-82)
script.delete() (on page 14-115)
script.load() (on page 14-116)
scriptVar.run() (on page 14-116)
scriptVar.save() (on page 14-117)
scriptVar.source (on page 14-118)

SMU

SMU commands and functions are unique to SMU instruments. You can use the SMU commands to control the instrument source and measure functions.

localnode.linefreq (on page 14-100)
smu.breakdownprotection (on page 14-118)
smu.interlock.enable (on page 14-120)
smu.interlock.tripped (on page 14-121)
smu.terminals (on page 14-203)

SMU measure commands are:

smu.measure.autorange (on page 14-122)
smu.measure.autorangehigh (on page 14-123)
smu.measure.autorangelow (on page 14-124)
smu.measure.autozero.enable (on page 14-126)
smu.measure.autozero.once() (on page 14-127)
smu.measure.configlist.catalog() (on page 14-127)
smu.measure.configlist.create() (on page 14-128)
smu.measure.configlist.delete() (on page 14-129)
smu.measure.configlist.query() (on page 14-130)
smu.measure.configlist.recall() (on page 14-131)
smu.measure.configlist.size() (on page 14-132)
smu.measure.configlist.store() (on page 14-133)
smu.measure.count (on page 14-135)
smu.measure.displaydigits (on page 14-138)
smu.measure.filter.count (on page 14-139)
smu.measure.filter.enable (on page 14-139)
smu.measure.filter.type (on page 14-140)
smu.measure.func (on page 14-142)
smu.measure.limit[Y].audible (on page 14-143)
smu.measure.limit[Y].autoclear (on page 14-144)
smu.measure.limit[Y].clear() (on page 14-145)
smu.measure.limit[Y].enable (on page 14-146)

smu.measure.limit[Y].fail (on page 14-147)
smu.measure.limit[Y].high.value (on page 14-149)
smu.measure.limit[Y].low.value (on page 14-150)
smu.measure.math.enable (on page 14-151)
smu.measure.math.format (on page 14-152)
smu.measure.math.mxb.bfactor (on page 14-153)
smu.measure.math.mxb.mfactor (on page 14-154)
smu.measure.math.percent (on page 14-155)
smu.measure.nplc (on page 14-156)
smu.measure.offsetcompensation (on page 14-157)
smu.measure.range (on page 14-158)
smu.measure.read() (on page 14-159)
smu.measure.readwithtime() (on page 14-160)
smu.measure.rel.acquire() (on page 14-161)
smu.measure.rel.enable (on page 14-162)
smu.measure.rel.level (on page 14-163)
smu.measure.sense (on page 14-164)
smu.terminals (on page 14-203)
smu.measure.unit (on page 14-167)
smu.measure.userdelay[N] (on page 14-168)

SMU source commands are:

smu.source.autorange (on page 14-170)
smu.source.autodelay (on page 14-171)
smu.source.configlist.catalog() (on page 14-172)
smu.source.configlist.create() (on page 14-172)
smu.source.configlist.delete() (on page 14-173)
smu.source.configlist.query() (on page 14-174)
smu.source.configlist.recall() (on page 14-175)
smu.source.configlist.size() (on page 14-176)
smu.source.configlist.store() (on page 14-177)
smu.source.delay (on page 14-179)
smu.source.func (on page 14-180)
smu.source.highc (on page 14-181)
smu.source.level (on page 14-182)
smu.source.offmode (on page 14-183)
smu.source.output (on page 14-185)
smu.source.protect.level (on page 14-186)
smu.source.protect.tripped (on page 14-187)
smu.source.range (on page 14-187)
smu.source.readback (on page 14-189)
smu.source.sweeplinear() (on page 14-191)
smu.source.sweeplinearstep() (on page 14-193)
smu.source.sweeplist() (on page 14-196)
smu.source.sweeplog() (on page 14-198)
smu.source.userdelay[N] (on page 14-200)
smu.source.xlimit.level (on page 14-201)
smu.source.xlimit.tripped (on page 14-202)

Status model

The status model is a set of status registers and queues. You can use the following commands to manipulate and monitor these registers and queues to view and control various instrument events.

status.clear() (on page 14-203)
status.condition (on page 14-204)
status.operation.condition (on page 14-205)
status.operation.enable (on page 14-205)
status.operation.event (on page 14-206)
status.operation.getmap() (on page 14-207)
status.operation.setmap() (on page 14-207)
status.preset() (on page 14-208)
status.questionable.condition (on page 14-209)
status.questionable.enable (on page 14-209)
status.questionable.event (on page 14-210)
status.questionable.getmap() (on page 14-211)
status.questionable.setmap() (on page 14-211)
status.request_enable (on page 14-212)
status.standard.enable (on page 14-213)
status.standard.event (on page 14-215)

Time

bufferVar.fractionalseconds (on page 14-32)
bufferVar.n (on page 14-33)
bufferVar.relativetimestamps (on page 14-35)
bufferVar.seconds (on page 14-36)
bufferVar.times (on page 14-44)
bufferVar.timestamps (on page 14-45)
delay() (on page 14-57)
localnode.gettime() (on page 14-100)
timer.cleartime() (on page 14-216)
timer.gettime() (on page 14-216)
trigger.timer[N].clear() (on page 14-279)
trigger.timer[N].count (on page 14-279)
trigger.timer[N].delay (on page 14-281)
trigger.timer[N].delaylist (on page 14-281)
trigger.timer[N].reset() (on page 14-283)
trigger.timer[N].start.fractionalseconds (on page 14-284)
trigger.timer[N].start.generate (on page 14-285)
trigger.timer[N].start.overrun (on page 14-286)
trigger.timer[N].start.seconds (on page 14-286)
trigger.timer[N].start.stimulus (on page 14-287)
trigger.timer[N].wait() (on page 14-288)
tspnet.timeout (on page 14-311)

Triggering

The triggering commands allow you to set the conditions that the instrument uses to determine when measurements are captured.

trigger.blender[N].clear() (on page 14-217)
trigger.blender[N].orenable (on page 14-217)
trigger.blender[N].overrun (on page 14-218)
trigger.blender[N].reset() (on page 14-219)
trigger.blender[N].stimulus[M] (on page 14-219)
trigger.blender[N].wait() (on page 14-221)
trigger.clear() (on page 14-221)
trigger.digin[N].clear() (on page 14-223)
trigger.digin[N].edge (on page 14-224)
trigger.digin[N].overrun (on page 14-225)
trigger.digin[N].wait() (on page 14-225)
trigger.digout[N].assert() (on page 14-226)
trigger.digout[N].logic (on page 14-227)
trigger.digout[N].pulsewidth (on page 14-228)
trigger.digout[N].release() (on page 14-228)
trigger.digout[N].stimulus (on page 14-229)
trigger.lanin[N].clear() (on page 14-230)
trigger.lanin[N].edge (on page 14-231)
trigger.lanin[N].overrun (on page 14-231)
trigger.lanin[N].wait() (on page 14-232)
trigger.lanout[N].assert() (on page 14-233)
trigger.lanout[N].connect() (on page 14-234)
trigger.lanout[N].connected (on page 14-234)
trigger.lanout[N].disconnect() (on page 14-235)
trigger.lanout[N].ipaddress (on page 14-236)
trigger.lanout[N].logic (on page 14-236)
trigger.lanout[N].protocol (on page 14-237)
trigger.lanout[N].stimulus (on page 14-238)
trigger.timer[N].clear() (on page 14-279)
trigger.timer[N].count (on page 14-279)
trigger.timer[N].delay (on page 14-281)
trigger.timer[N].delaylist (on page 14-281)
trigger.timer[N].enable (on page 14-282)
trigger.timer[N].reset() (on page 14-283)
trigger.timer[N].start.fractionalseconds (on page 14-284)
trigger.timer[N].start.generate (on page 14-285)
trigger.timer[N].start.overrun (on page 14-286)
trigger.timer[N].start.seconds (on page 14-286)
trigger.timer[N].start.stimulus (on page 14-287)
trigger.timer[N].wait() (on page 14-288)
trigger.tsplinkin[N].clear() (on page 14-289)
trigger.tsplinkin[N].edge (on page 14-289)
trigger.tsplinkin[N].overrun (on page 14-290)

trigger.tsplinkin[N].wait() (on page 14-291)
trigger.tsplinkout[N].assert() (on page 14-291)
trigger.tsplinkout[N].logic (on page 14-292)
trigger.tsplinkout[N].pulsewidth (on page 14-293)
trigger.tsplinkout[N].release() (on page 14-293)
trigger.tsplinkout[N].stimulus (on page 14-294)
trigger.wait() (on page 14-295)

Trigger model

The trigger model commands allow you to control and set the trigger model options.

trigger.model.abort() (on page 14-239)
trigger.model.getblocklist() (on page 14-239)
trigger.model.getbranchcount() (on page 14-240)
trigger.model.initiate() (on page 14-241)
trigger.model.load() — Config List (on page 14-241)
trigger.model.load() — Duration Loop (on page 14-242)
trigger.model.load() — Empty (on page 14-243)
trigger.model.load() — GradeBinning (on page 14-244)
trigger.model.load() — LogicTrigger (on page 14-245)
trigger.model.load() — LoopUntilEvent (on page 14-246)
trigger.model.load() — SimpleLoop (on page 14-248)
trigger.model.load() — SortBinning (on page 14-250)
trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS (on page 14-253)
trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 14-253)
trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA (on page 14-254)
trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT (on page 14-255)
trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC (on page 14-257)
trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT (on page 14-258)
trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE (on page 14-260)
trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR (on page 14-261)
trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT (on page 14-262)
trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV (on page 14-263)
trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL (on page 14-265)
trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT (on page 14-266)
trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 14-267)
trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO (on page 14-268)
trigger.model.setblock() — trigger.BLOCK_LOG_EVENT (on page 14-269)
trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE (on page 14-270)
trigger.model.setblock() — trigger.BLOCK_NOP (on page 14-272)
trigger.model.setblock() — trigger.BLOCK_NOTIFY (on page 14-273)
trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT (on page 14-274)
trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT (on page 14-275)
trigger.model.setblock() — trigger.BLOCK_WAIT (on page 14-276)
trigger.model.state() (on page 14-278)

TSP-Link

TSP-link commands allow you to control and access the TSP-Link synchronization port. Use these commands to perform basic steady-state digital I/O operations; for example, you can program the instrument to read and write to a specific TSP-Link synchronization line or to the entire port.

trigger.tsplinkin[N].edge (on page 14-289)
trigger.tsplinkin[N].overrun (on page 14-290)
trigger.tsplinkin[N].wait() (on page 14-291)
trigger.tsplinkout[N].assert() (on page 14-291)
trigger.tsplinkout[N].logic (on page 14-292)
trigger.tsplinkout[N].pulsewidth (on page 14-293)
trigger.tsplinkout[N].release() (on page 14-293)
trigger.tsplinkout[N].stimulus (on page 14-294)
tsplink.group (on page 14-296)
tsplink.initialize() (on page 14-296)
tsplink.line[N].mode (on page 14-298)
tsplink.line[N].reset() (on page 14-298)
tsplink.line[N].state (on page 14-299)
tsplink.master (on page 14-300)
tsplink.node (on page 14-301)
tsplink.readport() (on page 14-301)
tsplink.state (on page 14-302)
tsplink.writeport() (on page 14-303)

TSP-net

TSP-Net provides a simple socket-like programming interface to Test Script Processor (TSP) enabled instruments.

tspnet.clear() (on page 14-303)
tspnet.connect() (on page 14-304)
tspnet.disconnect() (on page 14-305)
tspnet.execute() (on page 14-306)
tspnet.idn() (on page 14-307)
tspnet.read() (on page 14-308)
tspnet.readavailable() (on page 14-309)
tspnet.reset() (on page 14-309)
tspnet.termination() (on page 14-310)
tspnet.timeout (on page 14-311)
tspnet.tsp.abort() (on page 14-311)
tspnet.tsp.abortonconnect (on page 14-312)
tspnet.tsp.rbtablecopy() (on page 14-313)
tspnet.tsp.runscript() (on page 14-314)
tspnet.write() (on page 14-314)

User strings

The `userstring` functions allow you to add user-defined strings to nonvolatile memory, delete the strings and see a list of the user strings in memory.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

userstring.add() (on page 14-316)
userstring.catalog() (on page 14-317)
userstring.delete() (on page 14-318)
userstring.get() (on page 14-318)

Section 14

TSP command reference

In this section:

TSP command programming notes..... 14-1
Using the TSP command reference 14-4
TSP commands..... 14-8

TSP command programming notes

This section contains general information about using TSP commands.

TSP syntax rules

This section provides rules that explain what you can and cannot do when entering TSP commands.

Upper and lower case

Instrument commands are case sensitive.

Function and attribute names are in lowercase characters.

Parameters and attribute constants can use a combination of lowercase and uppercase characters. The correct case for a specific command is shown in its command description.

The following example shows the `beeper.beep()` function, where `2` is the duration in seconds and `2400` is the frequency. Note that the function is in lowercase characters:

```
beeper.beep(2, 2400)
```

The following command changes the display light state to be at level 50. Note that the attribute (`display.lightstate`) is lower case, but the constant (`display.STATE_LCD_50`) is a combination of lowercase and uppercase characters:

```
display.lightstate = display.STATE_LCD_50
```

White space

You can send commands with or without white spaces.

For example, the following functions, which set the length and frequency of the instrument beeper, are equivalent:

```
beeper.beep(2,2400)
beeper.beep (2,  2400)
```

Parameters for functions

All functions must have a set of parentheses `()` immediately following the function. If there are parameters for the function, they are placed between the parentheses. The parentheses are required even when no parameters are specified.

The following example shows the `beeper.beep()` function, where `2` is the duration in seconds and `2400` is the frequency. Note that the parameters are inside the parentheses:

```
beeper.beep(2, 2400)
```

The command below resets commands to their default values (no parameters are needed):

```
reset()
```

Multiple parameters

Multiple parameters must be separated by commas.

For example, the following commands set the beeper to emit a double-beep at 2400 Hz, with a beep sequence of 0.5 s on, a delay of 0.25 s, and then 0.5 s on:

```
beeper.beep(0.5, 2400)
delay(0.250)
beeper.beep(0.5, 2400)
```

Time and date values

Time and date values are represented as the number of seconds since some base. The time bases are:

- **UTC 12:00 am Jan 1, 1970:** Some examples of UTC time are reading buffer timestamps, calibration adjustment and verification dates, and the value returned by `os.time()`.
- **Event:** Time referenced to an event, such as the first reading stored in a reading buffer.

Local and remote control

The instrument can be controlled locally or remotely.

When the instrument is controlled locally, you operate the instrument using the front-panel controls. When it is controlled remotely, you operate the instrument through a controller (usually a computer). When the instrument is first powered on, it is controlled locally.

The type of control is displayed on the front panel. When the instrument is in local control, the REMOTE LED indicator in the upper right corner is off and the control indicator on the upper left of the screen shows Local.

When the instrument is in remote control, the front-panel REMOTE LED indicator is on and the control indicator at the top left of the screen shows the type of communication interface.

Remote control

When the instrument is controlled remotely, the front-panel controls are disabled. You can still view information on the front-panel display and move between the screens using the keys and touchscreen controls. If you change a selection, however, you are prompted to switch control to local.

The OUTPUT ON/OFF switch is always active. If you press it when the instrument is controlled remotely, the instrument turns the output off (if it is on) and switches to local control.

To switch to remote control:

- Send a command from the computer to the instrument.
- Open communications between the instrument and Test Script Builder.

Local control

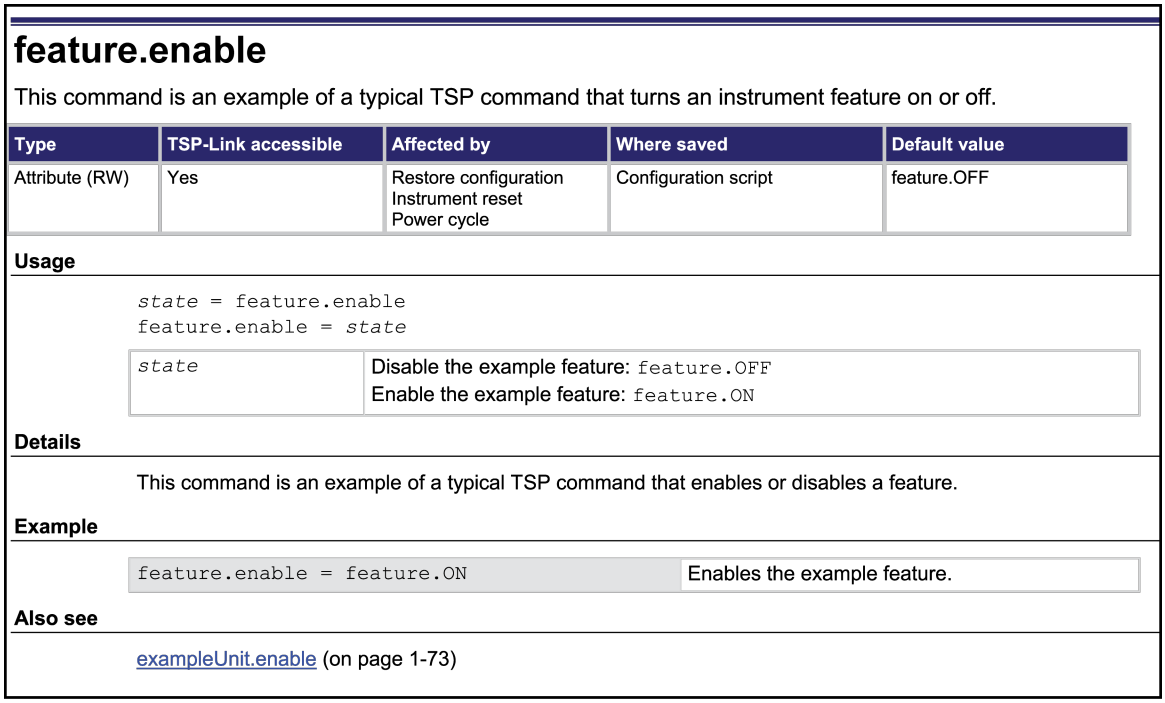
To change to local control, you can:

- Choose an option from the screens and try to change the value; select **Yes** on the dialog box that is displayed.
- Send the logout command from the computer.
- Turn the instrument off and on.

Using the TSP command reference

The TSP command reference contains detailed descriptions of each of the TSP commands that you can use to control your instrument. Each command description is broken into subsections. The figure below shows an example of a command description.

Figure 151: Example instrument command description



feature.enable

This command is an example of a typical TSP command that turns an instrument feature on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	feature.OFF

Usage

```
state = feature.enable
feature.enable = state
```

state	Disable the example feature: `feature.OFF` Enable the example feature: `feature.ON`

Details

This command is an example of a typical TSP command that enables or disables a feature.

Example

```
feature.enable = feature.ON
```
Enables the example feature.

Also see

exampleUnit.enable (on page 1-73)

The subsections contain information about the command. The subsections are:

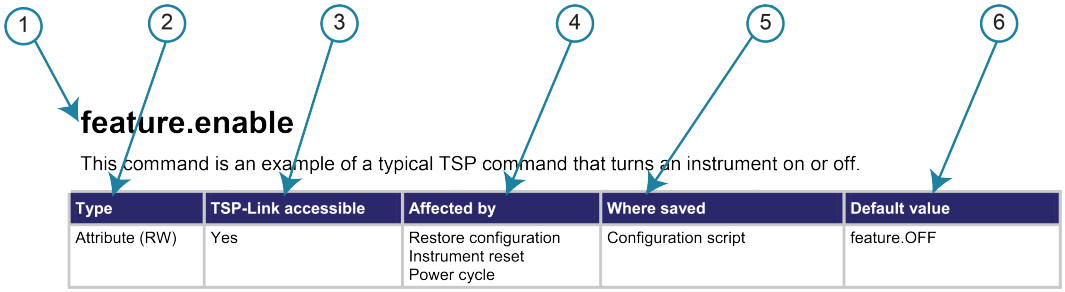
- Command name, brief description, and summary table
- Usage
- Details
- Example
- Also see

The content of each of these subsections is described in the following topics.

Command name, brief description, and summary table

Each instrument command description starts with the command name, followed by a brief description and a table with information for each command. Descriptions of the numbered items in the figure below are provided below.

Figure 152: TSP command name and summary table



1 **Instrument command name**. The beginning of the command description. It is followed by a brief description of what the command does.

2 **Type of command**. Commands can be functions, attributes, or constants. If the command is an attribute, it can be read-only (R), read-write (RW), or write-only (W).

3 **TSP-Link accessible**. Indicates whether or not the command can be accessed through a TSP-Link network (Yes or No).

4 **Affected by**. This column lists commands or actions that can change the value of the command, including.

- **Power cycle:** The command settings are not saved through a power cycle.
- **Restore configuration:** If you restore a configuration script, this setting changes to the stored setting.
- **Instrument reset:** When you reset the instrument, this command is reset to its default value. Reset can be done from the front panel or when you send `reset()` or `*RST`.
- **Source configuration list:** If you recall a source configuration list, this setting changes to the setting stored in the list.
- **Measure configuration list:** If you recall a measure configuration list, this setting changes to the setting stored in the list.
- **Function:** This command changes value when the function is changed (for example, changing from a voltage source to a current source or changing from a current measurement to a resistance measurement).

5 **Where saved**. Indicates where the command settings reside once they are used on an instrument. Options include:

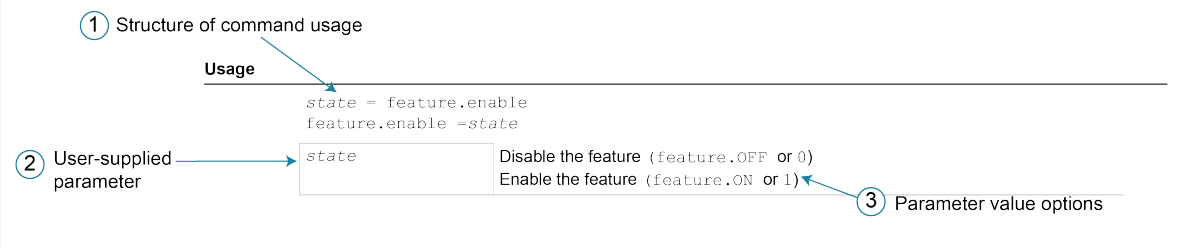
- **Not saved:** Command is not saved and must be typed each time you use it.
- **Nonvolatile memory:** The command is stored in a storage area in the instrument where information is saved even when the instrument is turned off.
- **Configuration script:** Command is saved as part of the configuration script.
- **Source configuration list:** This command is stored in source configuration lists.
- **Measure configuration list:** This command is stored in measure configuration lists.

6 **Default value:** Lists the default value or constant for the command.

Command usage

The Usage section of the remote command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage. All possible command usage options are shown.

Figure 153: TSP usage description



1 **Structure of command usage:** Shows how the parts of the command should be organized. If a parameter is shown to the left of the command, it is the return when you print the command. Information to the right is the parameters or other items you need to enter when setting the command.

2 **User-supplied parameters:** Indicated by italics. For example, for the function `beeper.beep(`*`duration, frequency`*`)`, replace *`duration`* with the number of seconds and *`frequency`* with the frequency of the tone. Send `beeper.beep(2, 2400)` to generate a two-second, 2400 Hz tone.

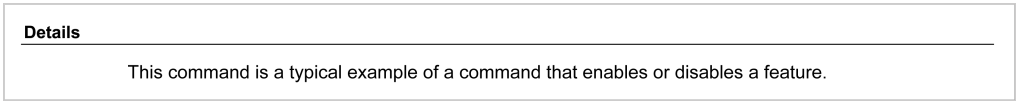
Some commands have optional parameters. If there are optional parameters, they must be entered in the order presented in the Usage section. You cannot leave out any parameters that precede the optional parameter. Optional parameters are shown as separate lines in usage, presented in the required order with each valid permutation of the optional parameters.
For example:
`printbuffer(`*`startIndex, endIndex, buffer1`*`)`
`printbuffer(`*`startIndex, endIndex, buffer1, buffer2`*`)`

3 **Parameter value options:** Descriptions of the options that are available for the user-defined parameter.

Command details

This section lists additional information you need to know to successfully use the remote command.

Figure 154: TSP Details description



Example section

The Example section of the remote command description shows examples of how you can use the command.

Figure 155: TSP example code



1 Actual example code that you can copy from this table and paste into your own programming application.

2 Description of the code and what it does. This may also contain example output of the code.

Related commands and information

The Also see section of the remote command description lists additional commands or sections that are related to the command.

Figure 156: TSP Also see description



TSP commands

The TSP commands available for the instrument are listed in alphabetic order.

available()

This function checks for the presence of specific instrument functionality.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

presence = available(*functionality*)

presence	If the functionality is present, returns `true`; if not, returns `false`
functionality	The functionality to check for: ■ Digital I/O: `digio` ■ GPIB communications: `gpib` ■ TSP-Link: `tsplink`

Example

```
print(available(gpib))
```

Returns `true` if GPIB communications are available. Returns `false` if GPIB communications are not available.

Also see

None

beeper.beep()

This function generates an audible tone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

beeper.beep(*duration*, *frequency*)

duration	The amount of time to play the tone (0.001 s to 100 s)
frequency	The frequency of the beep (20 Hz to 8000 Hz)

Details

You can use the beeper of the instrument to provide an audible signal at a specific frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

Using this function from a remote interface does not affect audible errors or key click settings that were made from the 2470 front panel.

Example

`beeper.beep(2, 2400)`	Generates a 2 s, 2400 Hz tone.

Also see

None

buffer.clearstats()

This function clears the statistical information associated with the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.clearstats()
buffer.clearstats(bufferVar)
```

bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1` if not specified

Details

This command clears the statistics without clearing the readings.

Example

`buffer.clearstats()`	Clears statistics for `defbuffer1`.
`buffer.clearstats(testData)`	Clears statistics for `testData`.

Also see

buffer.getstats() (on page 14-11)

buffer.delete()

This function deletes a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.delete(bufferName)
```

bufferName	The name of a user-defined reading buffer

Details

You cannot delete the default reading buffers, `defbuffer1` and `defbuffer2`.

After deleting the buffer, call `collectgarbage()` to reclaim the memory the buffer was using.

Example

```
buf400 = buffer.make(400)
smu.measure.read(buf400)
printbuffer(1, buf400.n, buf400.relativetimestamps)
buffer.delete(buf400)
collectgarbage()
```

Create a 400-element reading buffer named `buf400`.
Make measurements and store the readings in `buf400`.
Print the relative timestamps for each reading in the buffer.
Example output, assuming five readings are stored in the buffer:
`0, 0.412850017, 0.821640085, 1.230558058, 1.629523236`
Delete `buf400`.
Use `collectgarbage()` to unallocate the buffer.

Also see

buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

buffer.getstats()

This function returns statistics from a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
statsVar = buffer.getstats()
statsVar = buffer.getstats(bufferVar)
statsVar = buffer.getstats(bufferVar, absStartTime, absStartFractional, absEndTime,
    absEndFractional)
statsVar = buffer.getstats(bufferVar, relStartTime, relEndTime)
```

statsVar	A table that contains the entries for buffer statistics; see **Details** for information on the entries
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
absStartTime	An integer that represents the absolute start time in seconds
absStartFractional	An integer that represents the portion of the absolute start time that is in fractional seconds
absEndTime	An integer that represents the absolute end time in seconds
absEndFractional	An integer that represents the portion of the absolute end time that is in fractional seconds
relStartTime	The start time in seconds relative to the start time of the data in the buffer
relEndTime	The end time in seconds relative to the start time of the data in the buffer

Details

This function returns a table with statistical data about the data that was placed in the reading buffer.

The instrument automatically updates reading buffer statistics as data is added to the reading buffer.

When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make.

The table returned from this function provides statistics at the time the function is called. Although the instrument continues to update the statistics, the table that is returned is not updated. To get fresh statistics, call this function again.

The *statsVar* parameter contains the values described in the following table.

Attribute	When returned	Description
min	n > 0	A table that contains data about the minimum reading value that was added to the buffer; the table includes: ■ reading: The reading value ■ timestamp: The timestamp of the minimum data point in the buffer ■ seconds: The time in seconds ■ fractionalseconds: The time in fractional seconds
mean	n > 0	The average of all readings added to the buffer
stddev	n > 1	The standard deviation of all readings that were added to the buffer
n	Always	The number of data points on which the statistics are based
max	n > 0	A table that contains data about the maximum reading value that was added to the buffer; the table includes: ■ reading: The reading value ■ timestamp: The timestamp of the maximum data point in the buffer ■ seconds: The time in seconds ■ fractionalseconds: The fractional seconds

If n equals zero (0), all other values are nil. If n equals 1, stddev is nil because the standard deviation of a sample size of 1 is undefined.

Use the following command to get values from *statsVar*; a table with the following entries in it: n, min, max, mean, and stddev:

```
statsVar = buffer.getstats(bufferVar)
```

Use the following commands to print these entries:

```
print(statsVar.n)
print(statsVar.mean)
print(statsVar.stddev)
print(statsVar.min.reading)
print(statsVar.min.timestamp)
print(statsVar.min.seconds)
print(statsVar.min.fractionalseconds)
print(statsVar.max.reading)
print(statsVar.max.seconds)
print(statsVar.max.fractionalseconds)
print(statsVar.max.timestamp)
```

Example

```
reset()
trigger.model.load("SimpleLoop", 12, 0.001, defbuffer1)
trigger.model.initiate()
waitcomplete()

stats = buffer.getstats(defbuffer1)
print(stats)
```

Reset the instrument.
Set up the SimpleLoop trigger-model template to make 12 readings with a 0.001 s delay. Readings are stored in `defbuffer1`.
Start the trigger model.
Assign the name `stats` to the table.
Get statistics for the default reading buffer named `defbuffer1`.
Example output:

```
["min"]={["seconds"]=1561123956, ["fractionalseconds"]=0.010184587,
   ["timestamp"]=1561123956, ["reading"]=8.4974199416e-05},
   ["mean"]=0.000132948335, ["stddev"]=4.4270141937e-05,
   ["max"]={["seconds"]=1561123955, ["fractionalseconds"]=0.833083981,
   ["timestamp"]=1561123955.8, ["reading"]=0.0002192359033}, ["n"]=12
```

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

buffer.make()

This function creates a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar = buffer.make(bufferSize)
bufferVar = buffer.make(bufferSize, style)
```

bufferVar	The name of the buffer
bufferSize	The maximum number of readings that can be stored in *bufferVar*; minimum is `10`; `0` to maximize buffer size (see **Details**)
style	The type of reading buffer to create: ■ Store readings with reduced accuracy (6.5 digits) with no formatting information, 1 µs accurate timestamp: `buffer.STYLE_COMPACT` ■ Store readings with full accuracy with formatting: `buffer.STYLE_STANDARD` (default) ■ Store the same information as standard, plus additional information: `buffer.STYLE_FULL` ■ Store external reading buffer data: `buffer.STYLE_WRITABLE` ■ Store external reading buffer data with two reading values: `buffer.STYLE_WRITABLE_FULL`

Details

You cannot assign user-defined reading buffers the name `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

If you create a reading buffer that has the same name as an existing user-defined buffer, the existing buffer is overwritten by the new buffer. Any data in the existing buffer is lost.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you create becomes the active buffer.

If you select `0`, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The default fill mode of a user-defined buffer is once. You can change it to continuous later.

Once the buffer style is selected, it cannot be changed.

Once you store the first reading in a compact buffer, you cannot change certain measurement settings, including range, display digits, and units; you must clear the buffer first.

Not all remote commands are compatible with the compact, writable, and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

Writable reading buffers are used to bring external data into the instrument. You cannot assign them to collect data from the instrument.

You can change the buffer capacity for an existing buffer through the front panel or by using the *`bufferVar`*`.capacity` command.

Example

```
capTest2 = buffer.make(200, buffer.STYLE_FULL)
```

Creates a 200-element reading buffer that stores readings with full accuracy named `capTest2`.

Also see

bufferVar.capacity (on page 14-22)
bufferVar.fillmode (on page 14-30)
buffer.write.format() (on page 14-48)
buffer.write.reading() (on page 14-50)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
Writable reading buffers (on page 6-31)

buffer.math()

This function allows you to run a mathematical expression on a measurement. The expression is applied when the measurement is placed in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.math(readingBuffer, unit, buffer.EXPR_ADD)
buffer.math(readingBuffer, unit, buffer.EXPR_AVERAGE)
buffer.math(readingBuffer, unit, buffer.EXPR_DIVIDE)
buffer.math(readingBuffer, unit, buffer.EXPR_EXPONENT)
buffer.math(readingBuffer, unit, buffer.EXPR_LOG10)
buffer.math(readingBuffer, unit, buffer.EXPR_MULTIPLY)
buffer.math(readingBuffer, unit, buffer.EXPR_NONE)
buffer.math(readingBuffer, unit, buffer.EXPR_POLY, constant0, constant1, constant2,
   constant3, constant4, constant5)
buffer.math(readingBuffer, unit, buffer.EXPR_POWER, constant0)
buffer.math(readingBuffer, unit, buffer.EXPR_RATE)
buffer.math(readingBuffer, unit, buffer.EXPR_RECIPROCAL)
buffer.math(readingBuffer, unit, buffer.EXPR_SQROOT)
buffer.math(readingBuffer, unit, buffer.EXPR_SUBTRACT)
```

readingBuffer	The name of the reading buffer; the reading buffer selected must be set to the style `FULL`
unit	The units to be applied to the value generated by the expression: ■ DC current: `buffer.UNIT_AMP` ■ AC current: `buffer.UNIT_AMP_AC` ■ Celsius: `buffer.UNIT_CELSIUS` ■ Custom unit 1 (defined by `buffer.unit()`): `buffer.UNIT_CUSTOM1` ■ Custom unit 2 (defined by `buffer.unit()`): `buffer.UNIT_CUSTOM2` ■ Custom unit 3 (defined by `buffer.unit()`): `buffer.UNIT_CUSTOM3` ■ DAC (voltage): `buffer.UNIT_DAC` ■ Decibel-milliwatts: `buffer.UNIT_DBM` ■ Decibels: `buffer.UNIT_DECIBEL` ■ Digital I/O: `buffer.UNIT_DIO` ■ Fahrenheit: `buffer.UNIT_FAHRENHEIT` ■ Capacitance: `buffer.UNIT_FARAD` ■ Frequency: `buffer.UNIT_HERTZ` ■ Kelvin: `buffer.UNIT_KELVIN` ■ No unit: `buffer.UNIT_NONE` ■ Resistance: `buffer.UNIT_OHM` ■ Percent: `buffer.UNIT_PERCENT` ■ DC voltage ratio: `buffer.UNIT_RATIO` ■ Reciprocal: `buffer.UNIT_RECIPROCAL`

	■ Period: `buffer.UNIT_SECOND` ■ Totalizer: `buffer.UNIT_TOT` ■ DC voltage: `buffer.UNIT_VOLT` ■ AC voltage: `buffer.UNIT_VOLT_AC` ■ Power: `buffer.UNIT_WATT` ■ `buffer.UNIT_X`
constant0	The constant to be used for `c0` in the expression
constant1	The constant to be used for `c1` in the expression
constant2	The constant to be used for `c2` in the expression
constant3	The constant to be used for `c3` in the expression
constant4	The constant to be used for `c4` in the expression
constant5	The constant to be used for `c5` in the expression

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading
- a = previous reading
- t = timestamp of the reading
- c = constant

Expression	TSP parameter	Formula
Remove math expression	`buffer.EXPR_NONE`	Not applicable
Add	`buffer.EXPR_ADD`	$r + p$
Average	`buffer.EXPR_AVERAGE`	$\frac{(r+a)}{2}$
Divide	`buffer.EXPR_DIVIDE`	$\frac{r}{a}$
Exponent	`buffer.EXPR_EXPONENT`	10^r
Log10	`buffer.EXPR_LOG10`	$\log_{10} r$
Multiply	`buffer.EXPR_MULTIPLY`	$r * a$
Polynomial	`buffer.EXPR_POLY`	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	`buffer.EXPR_POWER`	r^c
Rate of change	`buffer.EXPR_RATE`	$\frac{(r-r_{-1})}{(t - t_{-1})}$
Reciprocal	`buffer.EXPR_RECIPROCAL`	$\frac{1}{r}$

Expression	TSP parameter	Formula
Square Root	buffer.EXPR_SQROOT	$\sqrt{r}$
Subtract	buffer.EXPR_SUBTRACT	r – a

Example

```
reset()
mathExp = buffer.make(200, buffer.STYLE_FULL)
smu.measure.func = smu.FUNC_DC_VOLTAGE

buffer.math(mathExp, buffer.UNIT_NONE, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
     print("Reading: ", smu.measure.read(mathExp))
end

display.changescreen(display.SCREEN_READING_TABLE)

print("Extra value reading 1: ", mathExp.extravalues[1])
print("Extra value reading 2: ", mathExp.extravalues[2])
print("Extra value reading 3: ", mathExp.extravalues[3])
```

Reset the instrument.
Make a buffer named `mathExp` set to hold 200 readings with a buffer style of FULL.
Set the measure function to DC voltage.
Set the buffer math expression to multiply readings against the previous readings.
Make three readings.
Display the reading table on the front panel of the instrument, where you can view the extra readings.
Print the extra values (the calculated values).
Example output:

```
Reading: 6.3863430578e-05
Reading: 6.7818055872e-05
Reading: 1.9871571784e-05
Extra value reading 1: 6.3863430578e-05
Extra value reading 2: 4.3310937031e-09
Extra value reading 3: 1.3476513655e-09
```

Also see

buffer.unit() (on page 14-21)

buffer.save()

This function saves data from the specified reading buffer to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.save(bufferVar, "fileName")
buffer.save(bufferVar, "fileName", timeFormat)
buffer.save(bufferVar, "fileName", timeFormat, start, end)
```

bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
fileName	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
timeFormat	Defines how date and time information from the buffer is saved in the file on the USB flash drive; the options are: ■ Save dates, times, and fractional seconds; `buffer.SAVE_FORMAT_TIME` ■ Save relative timestamps; `buffer.SAVE_RELATIVE_TIME` ■ Save seconds and fractional seconds; `buffer.SAVE_RAW_TIME` ■ Save timestamps; `buffer.SAVE_TIMESTAMP_TIME`
start	Defines the starting point in the buffer to start saving data
end	Defines the ending point in the buffer to stop saving data

Details

The file name must specify the full path (including `/usb1/`). If included, the file extension must be set to `.csv`. If no file extension is specified, `.csv` is added.

Examples of valid destination file names:

```
buffer.save(MyBuffer, "/usb1/myData")
buffer.save(MyBuffer, "/usb1/myData.csv")
```

The 2470 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing CSV files on the flash drive.

Example 1

```
buffer.save(MyBuffer, "/usb1/myData.csv")
```

Save all reading and default time information from a buffer named `MyBuffer` to a file named `myData.csv` on the USB flash drive.

Example 2

```
buffer.save(MyBuffer, "/usb1/myDataRel.csv", buffer.SAVE_RELATIVE_TIME)
```

Save all readings and relative timestamps from `MyBuffer` to a file named `myDataRel.csv` on the USB flash drive.

Example 3

```
buffer.save(defbuffer1, "/usb1/defbuf1data", buffer.SAVE_RAW_TIME)
```

Save readings and raw time stamps from `defbuffer1` to a file named `defbuf1data` on the USB flash drive.

Also see

buffer.make() (on page 14-13)
buffer.saveappend() (on page 14-19)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

buffer.saveappend()

This function appends data from the reading buffer to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.saveappend(bufferVar, "filename")
buffer.saveappend(bufferVar, "filename", timeFormat)
buffer.saveappend(bufferVar, "filename", timeFormat, start, end)
```

bufferVar	Indicates the reading buffer to use; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used
fileName	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
timeFormat	Indicates how date and time information from the buffer is saved in the file on the USB flash drive; the options are: ■ Save dates, times, and fractional seconds: `buffer.SAVE_FORMAT_TIME` ■ Save relative timestamps: `buffer.SAVE_RELATIVE_TIME` ■ Save seconds and fractional seconds: `buffer.SAVE_RAW_TIME` ■ Save timestamps: `buffer.SAVE_TIMESTAMP_TIME`
start	Defines the starting point in the buffer to start saving data
end	Defines the ending point in the buffer to stop saving data

Details

If the file you specify does not exist on the USB flash drive, this command creates the file.

For options that save more than one item of time information, each item is comma-delimited. For example, the default format is date, time, and fractional seconds for each reading.

The file extension `.csv` is appended to the file name if necessary. Any file extension other than `.csv` generates an error.

The index column entry in the `.csv` file starts at 1 for each append operation.

Examples of valid destination file names:

```
buffer.saveappend(bufferVar, "/usb1/myData")
buffer.saveappend(bufferVar, "/usb1/myData.csv")
```

Invalid destination file name examples:

```
buffer.saveappend(bufferVar, "/usb1/myData.")
```

— The period is not followed by csv.

```
buffer.saveappend(bufferVar, "/usb1/myData.txt")
```

— The only allowed extension is .csv. If .csv is not assigned, it is automatically added.

Example 1

```
buffer.saveappend(MyBuffer, "/usb1/myData.csv")
```

Append reading and default time information from a buffer named MyBuffer to a file named myData.csv on the USB flash drive.

Example 2

```
buffer.saveappend(MyBuffer, "/usb1/myDataRel.csv", buffer.SAVE_RELATIVE_TIME)
```

Append readings and relative timestamps from MyBuffer to the file myDataRel.csv on the USB flash drive.

Example 3

```
reset()
if file.usbdriveexists() == 1 then
   testDir = "TestData11"
-- Create a directory on the USB drive for the data.
   file.mkdir(testDir)
-- Build the full file and path.
   fileName = "/usb1/" .. testDir .. "/myTestData.csv"
-- Open the file where the data will be stored.
   fileNumber = file.open(fileName, file.MODE_WRITE)
-- Write the string data to a file.
   file.write(fileNumber, "Tested to Company Standard ABC.101\n")
-- Write the header separator to a file.
   file.write(fileNumber,
    "======================================================================\n")
-- Write the string data to a file.
   file.write(fileNumber, "\t1. Connect HI/LO to respective DUT terminals.\n")
   file.write(fileNumber, "\t2. Set power supply to 5 VDC @ 1 A.\n")
   file.write(fileNumber, "\t3. Wait 30 minutes.\n")
   file.write(fileNumber, "\t4. Capture 100 readings and analyze data.\n\n\n")
-- Write buffering data to a file.
   file.flush(fileNumber)
-- Close the data file.
   file.close(fileNumber)
end
-- Fix the range to 10 V.
smu.measure.range = 10.0
-- Set the measurement count to 100.
smu.measure.count = 100
-- Set up reading buffers.
-- Ensure the default measurement buffer size matches the count.
defbuffer1.capacity = smu.measure.count
smu.measure.read()
buffer.saveappend(defbuffer1, fileName)
```

Write string data to a file with information about a test file.

Also see

buffer.make() (on page 14-13)
buffer.save() (on page 14-18)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

buffer.unit()

This function allows you to create up to three custom units of measure for use in buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.unit(buffer.UNIT_CUSTOMN, unitOfMeasure)
```

N	The number of the custom unit, 1, 2, or 3
unitOfMeasure	A string that defines the custom unit; up to three characters; defaults are X for custom unit 1, Y for unit 2, and Z for unit 3

Details

You can use custom units of measures in buffer math and writable buffers.

If you specify more than two characters, the additional characters are ignored. Some characters are converted to other symbols:

- u is displayed as µ.
- dC is displayed as °C.
- dF is displayed as °F.
- RA is displayed as V/V.

This unit is reset when power is cycled. It is not affected by reset.

Example

```
reset()
mathExp = buffer.make(200, buffer.STYLE_FULL)
smu.measure.func = smu.FUNC_DC_VOLTAGE
buffer.unit(buffer.UNIT_CUSTOM1, "fb")

buffer.math(mathExp, buffer.UNIT_CUSTOM1, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
     print("Reading "..x..":", smu.measure.read(mathExp))
end

display.changescreen(display.SCREEN_READING_TABLE)
for x = 1, 3 do
     print("Extra value reading "..x..":", mathExp.extravalues[x])
end
```

```
Instrument has terminals set to FRONT.
Reset the instrument.
Make a buffer named mathExp set to hold 200 readings with a buffer style of FULL.
Set the measure function to DC voltage.
Set the customer 1 buffer unit to fb.
Set the buffer math expression to multiply readings against the previous readings.
Make 3 readings.
Display the reading table on the front panel of the instrument, where you can view the extra readings.
Print the extra values (the calculated values).
Example output:
Reading 1: 0.00015611271869
Reading 2: 9.0539004907e-05
Reading 3: 0.30001141669554
Extra value reading 1: 0.00015611271869
Extra value reading 2: 1.4134290203e-08
Extra value reading 3: 1.0336562635e-08
```

Also see

buffer.math() (on page 14-15)
buffer.write.format() (on page 14-48)

bufferVar.capacity

This attribute sets the number of readings a buffer can store.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferCapacity = bufferVar.capacity
bufferVar.capacity = bufferCapacity
```

bufferCapacity	The maximum number of readings the buffer can store; set to `0` to maximize the buffer size (see **Details**)
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

This command allows you to change or view how many readings a buffer can store. Changing the size of a buffer will cause any existing data in the buffer to be lost.

If you select `0`, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The overall capacity of all buffers stored in the instrument can be up to 4,500,000 readings for standard reading buffers and 20,000,000 for compact reading buffers.

For more information about buffer capacity, see Setting reading buffer capacity (on page 6-9).

Example

`reset()`
`testData = buffer.make(500)`
`capTest = buffer.make(300)`
`bufferCapacity = capTest.capacity`
 	
`print(bufferCapacity)`
 	
`print(testData.capacity)`
 	
`testData.capacity = 600`
`print(testData.capacity)`
`print(defbuffer1.capacity)`
	Create two user-defined reading buffers: *`testData`* and *`capTest`*.
	Create a variable called `bufferCapacity` to hold the capacity of the `capTest` buffer.
	Print `bufferCapacity`.
	Output:
	`300`
	Print the capacity of `testData`.
	Output:
	`500`
	Changes the capacity of `testData` to 600.
	Print the capacity of `testData`.
	Output:
	`600`
	Print the capacity of the default buffer `defbuffer1`.
	Output:
	`10000`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.clear()

This function clears all readings and statistics from the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`bufferVar``.clear()`

`bufferVar`	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData)
testData.clear()
print("Readings in buffer after clear ="
      .. testData.n)
```

```
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData)
```

Create a reading buffer named `testData`, make three readings and store them in `testData`, and then view the readings.
Print number of readings in `testData`.
Output:
`-4.5010112303956e-10, -3.9923108222095e-12, -4.5013931471161e-10`
Clear the readings in `testData`.
Verify that there are no readings in `testData`.
Output:
`Readings in buffer after clear = 0`
Store three new readings in `testData` and view those when complete.
Output:
`4.923509754e-07, 3.332266330e-07, 3.974883867e-07`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.dates

This attribute contains the dates of readings that are stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

date = *bufferVar*.dates[*N*]

date	The date of readings stored in *bufferVar* element *N*
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

The dates are formatted as month, day, year.

This is not available if the reading buffer style is set to compact.

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 1, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.dates[1])`
`printbuffer(1, testData.n, testData.dates)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	Print the first reading date.
	Example output:
	`11/27/2017`
	Prints the dates for readings 1 through the last reading in the buffer.
	Example output:
	`11/27/2017, 11/27/2017, 11/27/2017`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.endindex

This attribute indicates the last index in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

bufferVar`.endindex = `*endIndex*

endIndex	Ending index of the buffer
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

Use this attribute to find the ending index in a reading buffer.

Example

```
test1 = buffer.make(100)
smu.measure.count = 6
smu.measure.read(test1)
print(test1.startindex, test1.endindex, test1.capacity)
smu.measure.read(test1)
print(test1.startindex, test1.endindex)
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index, end index, and capacity of `test1`.
Output:
`1, 6, 100`
Make six more measurements and store them in buffer `test1`.
Get the start index and end index of `test1`.
Output:
`1, 12`

Also see

bufferVar.startindex (on page 14-42)
buffer.make() (on page 14-13)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.extraformattedvalues

This attribute contains the measurement and the unit of measure of the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

extraFormat = *bufferVar*.extraformattedvalues[*N*]

extraFormat	The measurement and unit of measure of the extra values for readings
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute contains the measurement and the unit of measure of an additional value. The reading buffer style must be set to full to use this option.

Example

```
reset()
mathExp = buffer.make(400, buffer.STYLE_FULL)
smu.measure.func = smu.FUNC_DC_VOLTAGE
buffer.math(mathExp, buffer.UNIT_NONE, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
     print("Reading: ", smu.measure.read(mathExp))
end
display.changescreen(display.SCREEN_READING_TABLE)
print("Extra value reading 1: ", mathExp.extraformattedvalues[1])
print("Extra value reading 2: ", mathExp.extraformattedvalues[2])
print("Extra value reading 3: ", mathExp.extraformattedvalues[3])
```

Reset the instrument.
Make a buffer named `mathExp` set to hold 400 readings with a buffer style of FULL.
Set the measure function to DC voltage.
Set the buffer math expression to multiply readings against the previous readings.
Make three readings.
Display the reading table on the front panel of the instrument, where you can view the extra readings.
Print the extra values (the calculated values).
Example output:

```
Reading:     7.1233589551e-06
Reading:     7.1233080234e-06
Reading:     7.2616603575e-06
Extra value reading 1:     +7.1233590 u
Extra value reading 2:     +50.741880 p
Extra value reading 3:     +51.727043 p
```

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.extravalues

This attribute contains the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraValue = bufferVar.extravalues[N]
```

extravalue	The extra values for readings
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute contains an additional value, such as the extra value written to a writable buffer. The reading buffer style must be set to full to use this option.

Example

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
      buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
      extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:

```
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC,
   10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.extravalueunits

This attribute contains the units of the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraUnits = bufferVar.extravalueunits[N]
```

extraUnits	The units of the extra values for readings
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute contains the unit of measure of an additional value. The reading buffer style must be set to full to use this option.

Example

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
      buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.extravalueunits)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and extra value units.
Read the buffer.
Output:

```
1, Watt DC, 2, Watt DC, 3, Watt DC, 4, Watt DC, 5, Watt DC, 6, Watt DC
```

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.fillmode

This attribute determines if a reading buffer is filled continuously or is filled once and stops.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	User-defined buffer: buffer.FILL_ONCE (0) defbuffer1: buffer.FILL_CONTINUOUS (1) defbuffer2: buffer.FILL_CONTINUOUS (1)

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

fillMode	Fill the buffer, then stop: `buffer.FILL_ONCE` or `0` Fill the buffer continuously: `buffer.FILL_CONTINUOUS` or `1`
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

When a reading buffer is set to fill once, no data is overwritten in the buffer. When the buffer is filled, no more data is stored in that buffer and new readings are discarded.

When a reading buffer is set to fill continuously, the oldest data is overwritten by the newest data after the buffer fills.

When you change the fill mode of a buffer, any data in the buffer is cleared.

Example

```
reset()
testData = buffer.make(50)
print(testData.fillmode)
testData.fillmode = buffer.FILL_CONTINUOUS
print(testData.fillmode)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer. Print the fill mode setting for the `testData` buffer.
Output:
`0`
Set fill mode to continuous.
Print the fill mode setting for the `testData` buffer.
Output:
`1`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.formattedreadings

This attribute contains the stored readings shown as numbers with units and prefixes.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
reading = bufferVar.formattedreadings[N]
```

reading	Buffer reading formatted with numbers and prefixes with units for element *N*
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This read-only attribute is an array that contains the stored readings. The readings are shown as numbers with prefixes before the units symbol.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.formattedreadings[1])
printbuffer(1, testData.n, testData.formattedreadings)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Print the first reading.
Example output:
`+00.0028 nA`
Print all readings in the reading buffer.
Example output:
`+00.0028 nA, +00.0039 nA, +00.0040 nA`

Also see

bufferVar.readings (on page 14-34)
buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.fractionalseconds

This attribute contains the fractional second portion of the timestamp of each reading in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
fractionalSec = bufferVar.fractionalseconds[N]
```

fractionalSec	The fractional second portion of the timestamp to 1 ns resolution
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This read-only attribute is an array of the fractional portion of the timestamp, in seconds, when each reading occurred. Seconds are shown as fractions.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.fractionalseconds[1])
printbuffer(1, 6, testData.fractionalseconds)
```

Create a reading buffer named `testData` and make six measurements.
Print the fractional portion of the timestamp for the first reading in the buffer.
Example output:
`0.647118937`
Print the fractional portion of the timestamp for the first six readings in the buffer.
Example output:
`0.647118937, 0.064543, 0.48196127, 0.89938724, 0.316800064, 0.734218263`

Also see

bufferVar.seconds (on page 14-36)
buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.logstate

This attribute indicates if information events are logged when the specified reading buffer is at 0% or 100% filled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Power cycle	Not applicable	defbuffer1: buffer.ON (1) defbuffer2: buffer.ON (1) User-created buffer: buffer.OFF (0)

Usage

```
logState = bufferVar.logstate
bufferVar.logstate = logState
```

logState	Do not log information events: `buffer.OFF` or `0` Log information events: `buffer.ON` or `1`
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

If this is set to on, when the reading buffer is cleared (0% filled) or full (100% filled), an event is logged in the event log. If this is set to off, reading buffer status is not reported in the event log.

Example

```
reset()
MyBuffer = buffer.make(500)
print(MyBuffer.logstate)
```

Create the user-defined buffer `MyBuffer`.
Print the log state of `MyBuffer`.
Output:
`0`

Also see

Using the event log (on page 3-50)

bufferVar.n

This attribute contains the number of readings in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
numberOfReadings = bufferVar.n
```

numberOfReadings	The number of readings stored in the reading buffer
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

You can call this command to return the number of readings stored in the specified reading buffer.

You can use the *bufferVar*.n attribute in other commands. For example, to print all the readings in a buffer, use the following command:

```
printbuffer(1, bufferVar.n, bufferVar.readings)
```

Where *bufferVar* is the name of the buffer to use.

Example

`reset()`
`testData = buffer.make(100)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.n)`
`print(defbuffer1.n)`
`print(defbuffer2.n)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	Print the number of readings in `testData`.
	Output:
	`3`
	Print the number of readings in `defbuffer1`.
	Example output:
	`0`
	Print the number of readings in `defbuffer2`.
	Example output:
	`0`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.readings

This attribute contains the readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
reading = bufferVar.readings[N]
```

reading	The value of the reading in the specified reading buffer
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
for x = 1, 3 do printbuffer(x, x, testData.readings, testData.sourcevalues,
   testData.relativetimestamps) end
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Print the three readings in `testData`, including the measurement, source value, and relative timestamp.
Output:

```
-9.6420389034124e-12, 2, 0
-4.5509945811872e-10, 2, 0.277194856
-9.1078204006445e-12, 2, 0.569614783
```

Also see

bufferVar.n (on page 14-33)
buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.relativetimestamps

This attribute contains the timestamps, in seconds, when each reading occurred, relative to the timestamp of the first entry in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

timestamp = *bufferVar*.relativetimestamps[*N*]

timestamp	The timestamp, in seconds
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This read-only attribute is an array of timestamps, in seconds, of when each reading occurred relative to the timestamp of the first entry in the reading buffer. These timestamps are equal to the time that has lapsed for each reading since the first reading was stored in the buffer. Therefore, the relative timestamp for the first entry number in the reading buffer equals 0.

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.relativetimestamps[1])`
`printbuffer(1, 3, testData.relativetimestamps)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	
	Print the relative timestamp for the first reading in the buffer.
	Example output:
	`0`
	Print the relative timestamp for the reading 1 through 3 in the buffer.
	Example output:
	`0, 0.383541, 0.772005`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.seconds

This attribute contains the timestamp of a reading in seconds, in UTC format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

nonFracSeconds = *bufferVar*.seconds[*N*]

nonFracSeconds	The nonfractional seconds portion of the timestamp when the reading was stored
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute contains the nonfractional seconds portion of the timestamp when the reading was stored in Coordinated Universal Time (UTC) format.

The nonfractional seconds portion of the timestamp gives the lowest resolution down to 1 second. To access additional resolution of a timestamp, see *bufferVar*.fractionalseconds.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 6, testData.seconds)
```

Create a reading buffer named `testData`, configure the instrument to make six measurements, and store the readings in the buffer.
Print the seconds portion for readings 1 to 6 in `testData`.
Example output:
`1362261492, 1362261492, 1362261493, 1362261493, 1362261493, 1362261494`

Also see

bufferVar.fractionalseconds (on page 14-32)
bufferVar.relativetimestamps (on page 14-35)

bufferVar.sourceformattedvalues

This attribute contains the source levels formatted as they appear on the front-panel display when the readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

values = *bufferVar*.sourceformattedvalues[*N*]

values	The output value of the source when reading *N* of the specified buffer was acquired
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

The attribute is an array (a Lua table) of the sourced value that was in effect at the time of the reading. The source levels are formatted the same way the readings are formatted when they appear on the front-panel display.

Example

```
reset()
trigger.model.load("SimpleLoop", 6, 0)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.sourceformattedvalues[1])
printbuffer(1,6,defbuffer1.sourceformattedvalues)
```

Example output:
`-00.00041 mV`
Example output:
`-00.00041 mV, +00.00010 mV, -00.00033 mV, +00.00003 mV, -00.00028 mV, -00.00045 mV`

Also see

bufferVar.n (on page 14-33)
buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.sourcestatuses

This attribute contains the source status conditions of the instrument for the reading point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
statusInfo = bufferVar.sourcestatuses[N]
```

statusInfo	The status value when reading *N* of the specified buffer was acquired; Source status values are: ■ Overvoltage protection was active: `buffer.STAT_PROTECTION` ■ Measured source value was read: `buffer.STAT_READBACK` ■ Overtemperature condition was active: `buffer.STAT_OVER_TEMP` ■ Source function level was limited: `buffer.STAT_LIMIT` ■ Four-wire sense was used: `buffer.STAT_SENSE` ■ Output was on: `buffer.STAT_OUTPUT`
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This buffer recall attribute holds an array (a Lua table) of the status values for all the readings in the buffer. The status values are floating-point numbers that encode the status value for each measurement made. See the following table for values.

Buffer status bits for source function when reading was acquired

Bit	Name	Decimal value	Description
2	STAT_PROTECTION	4	Overvoltage protection was active
3	STAT_READBACK	8	Measured source value was read
4	STAT_OVER_TEMP	16	Overtemperature condition was active
5	STAT_LIMIT	32	Source function level was limited
6	STAT_SENSE	64	Four-wire sense was used
7	STAT_OUTPUT	128	Output was on

Example

```
reset()
testData = buffer.make(50)
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 2, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 2, testData.sourcestatuses)
```

Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.
Turn on the source output.
Print the source status for the readings in `testData`.
Output:
`136, 136`
Indicating that the status is `buffer.STAT_READBACK` and `buffer.STAT_OUTPUT`.

Also see

buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
buffer.delete() (on page 14-10)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.sourceunits

This attribute contains the units of measure of the source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingUnits = bufferVar.sourceunits[N]
```

readingUnits	The units of measure of the source: ■ `Volt DC` ■ `Amp DC`
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

The attribute is an array (a Lua table) of strings indicating the units of measure at the time of the reading.

Example

```
reset()
testData = buffer.make(50)
smu.source.output = smu.ON
testData.fillmode = buffer.FILL_CONTINUOUS
trigger.model.load("SimpleLoop", 3, 0, testData)
smu.source.func = smu.FUNC_DC_CURRENT
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n,
   testData.sourceunits)
trigger.model.load("SimpleLoop", 3, 0, testData)
smu.source.func = smu.FUNC_DC_VOLTAGE
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n,
   testData.sourceunits)
smu.source.output = smu.OFF
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Set the source output to ON.
Set the buffer to fill continuously.
Set the source function to current.
Take three readings.
Print the units for the first three readings in the buffer.
Output:
`Amp DC, Amp DC, Amp DC`
Set the source function to voltage.
Take three readings.
Print the units for the readings in the buffer.
Output:
`Volt DC, Volt DC, Volt DC`

Also see

bufferVar.sourceunits (on page 14-40)
bufferVar.sourcevalues (on page 14-41)
bufferVar.statuses (on page 14-43)
Reading buffers (on page 6-1)

bufferVar.sourcevalues

This attribute contains the source levels being output when readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
sourceValue = bufferVar.sourcevalues[N]
```

sourceValue	The output value of the source when reading *N* of the specified buffer was acquired
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute is like an array (a Lua table) of the sourced value in effect at the time of the reading.

The values returned by this command depend on the source readback state:

- If readback is off, the value is the programmed value
- If readback is on, the value is the actual measured source value

Example

```
reset()
testData = buffer.make(50)
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.level = 1e-6
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 3, testData.sourcevalues)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Set the source value to 1e-6 A.
Print the source values being output when readings in the reading buffer were acquired.
Example output:

```
9.9999874692e-07,
    1.0000017028e-06,
    1.0000054544e-06
```

Also see

bufferVar.sourcestatuses (on page 14-38)
bufferVar.sourceunits (on page 14-40)
bufferVar.statuses (on page 14-43)
bufferVar.formattedreadings (on page 14-31)
buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)
smu.source.readback (on page 14-189)

bufferVar.startindex

This attribute indicates the starting index in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferVar.startindex = startIndex
```

startIndex	Starting index of the buffer
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

Use this attribute to find the starting index in a reading buffer.

Example

```
test1 = buffer.make(100)
smu.measure.count = 6
smu.measure.read(test1)
print(test1.startindex, test1.endindex, test1.capacity)
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index, end index, and capacity of `test1`.
Output:

```
1, 6, 100
```

Also see

bufferVar.endindex (on page 14-25)
buffer.make() (on page 14-13)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.statuses

This attribute contains the status values of readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
statusInformation = bufferVar.statuses[N]
```

statusInformation	The status value when reading *N* of the specified buffer was acquired; refer to **Details**
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This command is not available if the buffer style is set to compact.

This read-only attribute is an array of status values for the readings in the buffer. The status values are floating-point numbers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description	*statusInformation* parameter
0x0001	STAT_QUESTIONABLE	1	Measure status questionable	buffer.STAT_QUESTIONABLE
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the 2470, this will always be 0 (main))	buffer.STAT_ORIGIN
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0	buffer.STAT_TERMINAL
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low	buffer.STAT_LIMIT2_LOW
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high	buffer.STAT_LIMIT2_HIGH
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low	buffer.STAT_LIMIT1_LOW
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high	buffer.STAT_LIMIT1_HIGH
0x0100	STAT_START_GROUP	256	First reading in a group	buffer.STAT_START_GROUP

Example

`reset()`
`testData = buffer.make(50)`
`smu.source.output = smu.ON`
`trigger.model.load("SimpleLoop", 2, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`printbuffer(1, 2, testData.statuses)`
	Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.
	Turn on the source output.
	Print the source status for the readings in `testData`.
	Output:
	`64, 64`
	Indicating that the status is `buffer.STAT_LIMIT1_LOW`.

Also see

buffer.make() (on page 14-13)
buffer.delete() (on page 14-10)
bufferVar.clear() (on page 14-23)
bufferVar.sourcestatuses (on page 14-38)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.times

This attribute contains the time when the instrument made the reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

readingTime = *bufferVar*.times[*N*]

readingTime	The time of the reading in hours, minutes, and seconds
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.times[1])`
`printbuffer(1, 3, testData.times)`
	This example creates a reading buffer named `testData` and makes three measurements.
	The `print()` command outputs the time of the first reading.
	Output:
	`23:09:43`
	The `printbuffer()` command outputs the time of readings 1 to 3 in the reading buffer.
	Output:
	`23:09:43, 23:09:43, 23:09:43`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.timestamps

This attribute contains the timestamp when each reading saved in the specified reading buffer occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

readingTimestamp = *bufferVar*.timestamps[*N*]

readingTimestamp	The complete timestamp (including date, time, and fractional seconds) of reading number *N* in the specified reading buffer when the reading was acquired
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute contains the timestamps (date, hours, minutes, seconds, and fractional seconds) of readings stored in the reading buffer.

NOTE

When using the compact buffer style, there is a very small drift between the triggering clock and the timestamp clock, which may result in timestamp truncation and discontinuities over time. The triggering of the measurement remains periodic based on the sample period without any apparent discontinuities.

Example 1

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.timestamps[1])
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Print the first reading date.
Output:

```
03/01/2018 14:46:07.714614838
```

Example 2

```
for x = 1, 3 do printbuffer(x, x, testData.timestamps) end
```

For the buffer created in Example 1, print the timestamps for the readings.
Output:

```
03/01/2018 14:46:07.714614838
03/01/2018 14:46:08.100468838
03/01/2018 14:46:08.487631838
```

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

bufferVar.units

This attribute contains the unit of measure that is stored with readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

readingUnit = *bufferVar*.units[*N*]

readingUnit	`Volt DC`: Voltage measurement `Ohm`: Resistance measurement `Watt DC`: Power measurement `Amp DC`: Current measurement `%`: Math is set to percent for the measurements `X`: Math is set to mx+b for the measurements `Reciprocal`: Math is set to reciprocal for the measurements
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar.*n command to determine the number of readings in the buffer

Details

This attribute contains the unit of measure that is stored with readings in the reading buffer.

Example

```
reset()
testData = buffer.make(50)
testData.fillmode = buffer.FILL_CONTINUOUS
smu.measure.func = smu.FUNC_DC_CURRENT
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.units)
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.units)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Set the buffer to fill continuously.
Set the measure function to current.
Make three readings.
Print the units for the readings.
Output:
`Amp DC, Amp DC, Amp DC`
Set the measure function to voltage.
Make three readings.
Output:
`Volt DC, Volt DC, Volt DC`

Also see

buffer.delete() (on page 14-10)
buffer.make() (on page 14-13)
bufferVar.clear() (on page 14-23)
print() (on page 14-109)
printbuffer() (on page 14-110)
Reading buffers (on page 6-1)
Remote buffer operation (on page 6-25)

buffer.write.format()

This function sets the units and number of digits of the readings that are written into the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.write.format(bufferVar, units, displayDigits)
buffer.write.format(bufferVar, units, displayDigits, extraUnits)
buffer.write.format(bufferVar, units, displayDigits, extraUnits, extraDigits)
```

bufferVar	The name of the buffer
units	The units for the first measurement in the buffer index: ■ buffer.UNIT_AMP ■ buffer.UNIT_AMP_AC ■ buffer.UNIT_CELSIUS ■ buffer.UNIT_CUSTOM1 ■ buffer.UNIT_CUSTOM2 ■ buffer.UNIT_CUSTOM3 ■ buffer.UNIT_DAC ■ buffer.UNIT_DBM ■ buffer.UNIT_DECIBEL ■ buffer.UNIT_DIO ■ buffer.UNIT_FAHRENHEIT ■ buffer.UNIT_FARAD ■ buffer.UNIT_HERTZ ■ buffer.UNIT_KELVIN ■ buffer.UNIT_NONE ■ buffer.UNIT_OHM ■ buffer.UNIT_PERCENT ■ buffer.UNIT_RATIO ■ buffer.UNIT_RECIPROCAL ■ buffer.UNIT_SECOND ■ buffer.UNIT_TOT ■ buffer.UNIT_VOLT ■ buffer.UNIT_VOLT_AC ■ buffer.UNIT_WATT ■ buffer.UNIT_X
displayDigits	The number of digits to use for the first measurement: ■ buffer.DIGITS_3_5 ■ buffer.DIGITS_4_5 ■ buffer.DIGITS_5_5 ■ buffer.DIGITS_6_5 ■ buffer.DIGITS_7_5 ■ buffer.DIGITS_8_5

extraUnits	The units for the second measurement in the buffer index; the selections are the same as *units* (only valid for buffer style WRITABLE_FULL); if not specified, uses the value for *units*
extraDigits	The number of digits to use for the second measurement; the selections are the same as *displayDigits* (only valid for buffer style WRITABLE_FULL); if not specified, uses the value for *displayDigits*

Details

This command is valid when the buffer style is writable or full writable. When the buffer style is set to full writable, you can include an extra value.

The format defines the units and the number of digits that are reported for the data. This command affects how the data is shown in the reading buffer and what is shown on the front-panel Home, Histogram, Reading Table, and Graph screens.

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1)
buffer.write.reading(extBuffer, 2)
buffer.write.reading(extBuffer, 3)
buffer.write.reading(extBuffer, 4)
buffer.write.reading(extBuffer, 5)
buffer.write.reading(extBuffer, 6)
printbuffer(1, 6, extBuffer.readings, extBuffer.units)
```

Creates a 100-point reading buffer named extBuffer. Style is writable.
Set the data format to show units of watts with 3½ digit resolution.
Write 6 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:

```
1.0000000000e+00, Watt DC, 2.0000000000e+00, Watt DC, 3.0000000000e+00, Watt DC,
   4.0000000000e+00, Watt DC, 5.0000000000e+00, Watt DC, 6.0000000000e+00, Watt DC
```

Example 2

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
      buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
      extBuffer.units)
```

Creates a 100-point reading buffer named extBuffer. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:

```
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC,
   10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Also see

buffer.make() (on page 14-13)
buffer.write.reading() (on page 14-50)
Reading buffers (on page 6-1)
Writable reading buffers (on page 6-31)

buffer.write.reading()

This function allows you to write readings into the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

For buffers that are set to the writable buffer style:

```
buffer.write.reading(bufferVar, readingValue)
buffer.write.reading(bufferVar, readingValue, seconds)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds, status)
```

For buffers that are set to the full writable buffer style:

```
buffer.write.reading(bufferVar, readingValue, extraValue)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds, fractionalSeconds)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds, fractionalSeconds,
   status)
```

bufferVar	The name of the buffer
readingValue	The first value that is recorded in the buffer index
extraValue	A second value that is recorded in the buffer index (only valid for buffer style WRITABLE_FULL)
seconds	An integer that represents the seconds
fractionalSeconds	The portion of the time that represents the fractional seconds
status	Additional information about the reading; see **Details**

Details

This command writes the data you specify into a reading buffer. The reading buffer must be set to the writable or full writable style, which is set when you make the buffer.

Data must be added in chronological order. If the time is not specified for a reading, it is set to one integer second after the last reading. As you write the data, the front-panel home screen updates and displays the reading you entered.

The *status* parameter provides additional information about the reading. The options are shown in the following table.

Parameter	Description
buffer.STAT_LIMIT1_HIGH	Reading is above high limit 1
buffer.STAT_LIMIT1_LOW	Reading is below low limit 1
buffer.STAT_LIMIT2_HIGH	Reading is above high limit 2
buffer.STAT_LIMIT2_LOW	Reading is below low limit 2

Parameter	Description
buffer.STAT_ORIGIN	A/D converter from which reading originated; for the 2470, this will always be 0 (main)
buffer.STAT_QUESTIONABLE	Measure status questionable
buffer.STAT_REL	Relative offset
buffer.STAT_SCAN	Scan
buffer.STAT_START_GROUP	First reading in a group
buffer.STAT_TERMINAL	Measure terminal, front is 1, rear is 0

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1)
buffer.write.reading(extBuffer, 2)
buffer.write.reading(extBuffer, 3)
buffer.write.reading(extBuffer, 4)
buffer.write.reading(extBuffer, 5)
buffer.write.reading(extBuffer, 6)
printbuffer(1, 6, extBuffer.readings, extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is writable.
Set the data format to show units of watts with 3½ digit resolution.
Write 6 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:

```
1, Watt DC, 2, Watt DC, 3, Watt DC, 4, Watt DC, 5, Watt DC, 6, Watt DC
```

Example 2

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
      buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
      extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:

```
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC,
   10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Also see

buffer.make() (on page 14-13)
buffer.write.format() (on page 14-48)
Reading buffers (on page 6-1)
Writable reading buffers (on page 6-31)

createconfigscript()

This function creates a setup file that captures most of the present settings of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
createconfigscript("scriptName")
```

scriptName	A string that represents the name of the script that will be created

Details

This function does not automatically overwrite existing scripts with the same name. If *scriptName* is set to the name of an existing script, an event message is returned. You must delete the existing script before using the same script name. This includes the `autoexec` script, which runs automatically when the instrument power is turned on. You can set *scriptName* to `autoexec`, but you must delete the existing `autoexec` script first using the `script.delete("autoexec")` command.

Once created, the script that contains the settings can be run and edited like any other script.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press HOME and ENTER to save an image of the settings with the screen capture feature. Refer to Save screen captures to a USB flash drive (on page 3-44) for additional information.

Example

```
createconfigscript("myConfigurationScript")
reset()
myConfigurationScript()
```

Capture the present settings of the instrument into a script named `myConfigurationScript`.
Reset the instrument.
Restore the settings stored in `myConfigurationScript`.

Also see

Saving setups (on page 3-45)
script.delete() (on page 14-115)

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = dataqueue.add(value)
result = dataqueue.add(value, timeout)
```

result	The resulting value of `true` or `false` based on the success of the function
value	The data item to add; *value* can be of any type
timeout	The maximum number of seconds to wait for space in the data queue

Details

You cannot use the *timeout* value when accessing the data queue from a remote node (you can only use the *timeout* value while adding data to the local data queue).

The *timeout* value is ignored if the data queue is not full.

The `dataqueue.add()` function returns `false`:

- If the timeout expires before space is available in the data queue
- If the data queue is full and a *timeout* value is not specified

If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
dataqueue.add(10)
dataqueue.add(11, 2)
result = dataqueue.add(12, 3)
if result == false then
   print("Failed to add 12 to the dataqueue")
end
print("The dataqueue contains:")
while dataqueue.count > 0 do
   print(dataqueue.next())
end
```

Clear the data queue.
Each line adds one item to the data queue.
Output:

```
The dataqueue contains:
10
11
12
```

Also see

dataqueue.CAPACITY (on page 14-54)
dataqueue.clear() (on page 14-54)
dataqueue.count (on page 14-55)
dataqueue.next() (on page 14-56)
Using the data queue for real-time communication (on page 9-10)

dataqueue.CAPACITY

This constant is the maximum number of entries that you can store in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
count = dataqueue.CAPACITY
```

count	The variable that is assigned the value of `dataqueue.CAPACITY`

Details

This constant always returns the maximum number of entries that can be stored in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

This example fills the data queue until it is full and prints the number of items in the queue.
Output:
`There are 128 items in the data queue`

Also see

dataqueue.add() (on page 14-53)
dataqueue.clear() (on page 14-54)
dataqueue.count (on page 14-55)
dataqueue.next() (on page 14-56)
Using the data queue for real-time communication (on page 9-10)

dataqueue.clear()

This function clears the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dataqueue.clear()
```

Details

This function forces all `dataqueue.add()` commands that are in progress to time out and deletes all data from the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.
Output:

```
There are 128 items in the data
   queue
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 14-53)
dataqueue.CAPACITY (on page 14-54)
dataqueue.count (on page 14-55)
dataqueue.next() (on page 14-56)
Using the data queue for real-time communication (on page 9-10)

dataqueue.count

This attribute contains the number of items in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
count = dataqueue.count
```

count	The number of items in the data queue

Details

The count is updated as entries are added with `dataqueue.add()` and read from the data queue with `dataqueue.next()`. It is also updated when the data queue is cleared with `dataqueue.clear()`.

A maximum of `dataqueue.CAPACITY` items can be stored at any one time in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.
Output:

```
There are 128 items in the data queue
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 14-53)
dataqueue.CAPACITY (on page 14-54)
dataqueue.clear() (on page 14-54)
dataqueue.next() (on page 14-56)
Using the data queue for real-time communication (on page 9-10)

dataqueue.next()

This function removes the next entry from the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dataqueue.next()
value = dataqueue.next(timeout)
```

value	The next entry in the data queue
timeout	The number of seconds to wait for data in the queue

Details

If the data queue is empty, the function waits up to the *timeout* value.

If data is not available in the data queue before the *timeout* expires, the return value is `nil`.

The entries in the data queue are removed in first-in, first-out (FIFO) order.

If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
for i = 1, 10 do
   dataqueue.add(i)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")

while dataqueue.count > 0 do
   x = dataqueue.next()
   print(x)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

Clears the data queue, adds ten entries, then reads the entries from the data queue. Note that your output may differ depending on the setting of `format.asciiprecision`.

Output:

```
There are 10 items in the data queue
1
2
3
4
5
6
7
8
9
10
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 14-53)
dataqueue.CAPACITY (on page 14-54)
dataqueue.clear() (on page 14-54)
dataqueue.count (on page 14-55)
format.asciiprecision (on page 14-88)
Using the data queue for real-time communication (on page 9-10)

delay()

This function delays the execution of the commands that follow it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
delay(seconds)
```

seconds	The number of seconds to delay (0 to 100 ks)

Details

The instrument delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the instrument to delay 5 µs to 10 µs (typical) more than the requested delay.

Example 1

```
beeper.beep(0.5, 2400)
delay(0.250)
beeper.beep(0.5, 2400)
```

Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.
Output:

```
Delay time was 0.500099
35
```

Also see

None

digio.line[N].mode

This attribute sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	digio.MODE_DIGITAL_IN

Usage

```
lineMode = digio.line[N].mode
digio.line[N].mode = lineMode
```

lineMode	The digital line control type and line mode: Digital control, input: `digio.MODE_DIGITAL_IN` Digital control, output: `digio.MODE_DIGITAL_OUT` Digital control, open-drain: `digio.MODE_DIGITAL_OPEN_DRAIN` Trigger control, input: `digio.MODE_TRIGGER_IN` Trigger control, output: `digio.MODE_TRIGGER_OUT` Trigger control, open-drain: `digio.MODE_TRIGGER_OPEN_DRAIN` Synchronous master: `digio.MODE_SYNCHRONOUS_MASTER` Synchronous acceptor: `digio.MODE_SYNCHRONOUS_ACCEPTOR`
N	The digital I/O line: 1 to 6

Details

You can use this command to place each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

The following settings of *lineMode* set the line for direct control as a digital line:

- `digio.MODE_DIGITAL_IN`: The instrument automatically detects externally generated logic levels. You can read an input line, but you cannot write to it.
- `digio.MODE_DIGITAL_OUT`: You can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low.
- `digio.MODE_DIGITAL_OPEN_DRAIN`: Configures the line to be an open-drain signal. The line can serve as an input, an output or both. When a digital I/O line is used as an input in open-drain mode, you must write a `1` to it.

The following settings of *lineMode* set the line as a trigger line:

- `digio.MODE_TRIGGER_IN`: The line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This line state uses the edge setting specified by the `trigger.digin[N].edge` attribute.
- `digio.MODE_TRIGGER_OUT`: The line is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
- `digio.MODE_TRIGGER_OPEN_DRAIN`: Configures the line to be an open-drain signal. You can use the line to detect input triggers or generate output triggers. This line state uses the edge setting specified by the `trigger.digin[N].edge` attribute.

When the line is set as a synchronous acceptor, the line detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting an output trigger releases the latched line.

When the line is set as a synchronous master, the line detects rising-edge triggers as input. For output, the line asserts a TTL-low pulse.

Example

`digio.line[1].mode = digio.MODE_TRIGGER_OUT`	Set digital I/O line 1 to be an output trigger line.

Also see

Digital I/O lines (on page 8-16)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].edge (on page 14-224)

digio.line[N].reset()

This function resets digital I/O line values to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`digio.line[N].reset()`

N	The digital I/O line: 1 to 6

Details

This function resets the following attributes to their default values:

- `digio.line[N].mode`
- `trigger.digin[N].edge`
- `trigger.digout[N].logic`
- `trigger.digout[N].pulsewidth`
- `trigger.digout[N].stimulus`

It also clears `trigger.digin[N].overrun`.

Example

```
-- Set the digital I/O trigger line 3 for a falling edge
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.digout[3].logic = trigger.LOGIC_NEGATIVE
-- Set the digital I/O trigger line 3 to have a pulsewidth of 50 microseconds.
trigger.digout[3].pulsewidth = 50e-6
-- Use digital I/O line 5 to trigger the event on line 3.
trigger.digout[3].stimulus = trigger.EVENT_DIGIO5
-- Print configuration (before reset).
print(digio.line[3].mode, trigger.digout[3].pulsewidth, trigger.digout[3].stimulus)
-- Reset the line back to factory default values.
digio.line[3].reset()
-- Print configuration (after reset).
print(digio.line[3].mode, trigger.digout[3].pulsewidth, trigger.digout[3].stimulus)
```

Output before reset:

```
digio.MODE_TRIGGER_OUT   5e-05   trigger.EVENT_DIGIO5
```

Output after reset:

```
digio.MODE_TRIGGER_IN   1e-05   trigger.EVENT_NONE
```

Also see

digio.line[N].mode (on page 14-58)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].overrun (on page 14-225)
trigger.digout[N].pulsewidth (on page 14-228)
trigger.digout[N].stimulus (on page 14-229)

digio.line[N].state

This function sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	See **Details**

Usage

```
digio.line[N].state = state
state = digio.line[N].state
```

N	The digital I/O line: 1 to 6
state	Set the line low: `digio.STATE_LOW` or `0` Set the line high: `digio.STATE_HIGH` or `1`

Details

When a reset occurs, the digital line state can be read as high because the digital line is reset to a digital input. A digital input floats high if nothing is connected to the digital line.

This returns the integer equivalent values of the binary states on all six digital I/O lines.

Set the state to `digio.STATE_LOW` to clear the bit; set the state to `digio.STATE_HIGH` to set the bit.

Example

```
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[1].state = digio.STATE_HIGH
```

Sets line 1 (bit B1) of the digital I/O port high.

Also see

digio.line[N].mode (on page 14-58)
digio.readport() (on page 14-61)
digio.writeport() (on page 14-62)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].edge (on page 14-224)

digio.readport()

This function reads the digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readport()
```

data	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the digital I/O port. The least significant bit (bit B1) of the binary number corresponds to digital I/O line 1; bit B6 corresponds to digital I/O line 6.

For example, a returned value of 42 has a binary equivalent of 101010, which indicates that lines 2, 4, 6 are high (1), and the other lines are low (0).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

`data = digio.readport()` `print(data)`	Assume lines 2, 4, and 6 are set high when the I/O port is read. Output: 42 This is binary 101010

Also see

digio.writeport() (on page 14-62)
Digital I/O port configuration (on page 8-14)

digio.writeport()

This function writes to all digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`digio.writeport(`*`data`*`)`

`data`	The value to write to the port (`0` to `63`)

Details

This function writes to the digital I/O port by setting the binary state of each digital line from an integer equivalent value.

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a *`data`* value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other three lines are set low).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

You can also enter the data parameter as a binary value.

Example 1

`digio.writeport(63)`	Sets digital I/O lines 1 through 6 high (binary 111111).

Example 2

`digio.writeport(0b111111)`	Sets digital I/O lines 1 through 6 high (digital 63).

Also see

digio.readport() (on page 14-61)
Digital I/O port configuration (on page 8-14)
Outputting a bit pattern (on page 8-23)

display.activebuffer

This attribute determines which buffer is used for measurements that are displayed on the front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	defbuffer1

Usage

```
bufferName = display.activebuffer
display.activebuffer = bufferName
```

bufferName	The name of the buffer to make active

Details

The buffer defined by this command is used to store measurements data and is shown in the reading buffer indicator on the home screen of the instrument.

Example

`display.activebuffer = buffer2`	Set the front panel to use `buffer2` as the active reading buffer.

Also see

None

display.changescreen()

This function changes which front-panel screen is displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.changescreen(screenName)
```

screenName	The screen to display: ■ Home screen: `display.SCREEN_HOME` ■ Home screen with large readings: `display.SCREEN_HOME_LARGE_READING` ■ Reading table screen: `display.SCREEN_READING_TABLE` ■ Graph screen (opens last selected tab): `display.SCREEN_GRAPH` ■ Histogram: `display.SCREEN_HISTOGRAM` ■ GRAPH swipe screen: `display.SCREEN_GRAPH_SWIPE` ■ SETTINGS swipe screen: `display.SCREEN_SETTINGS_SWIPE` ■ SOURCE swipe screen: `display.SCREEN_SOURCE_SWIPE` ■ STATISTICS swipe screen: `display.SCREEN_STATS_SWIPE` ■ USER swipe screen: `display.SCREEN_USER_SWIPE` ■ Open a screen that uses minimal CPU resources: `display.SCREEN_PROCESSING`

Example

```
display.clear()
display.settext(display.TEXT1, "Batch A122")
display.settext(display.TEXT2, "Test running")
display.changescreen(display.SCREEN_USER_SWIPE)
```

Clear the USER swipe screen.
Set the first line of the USER swipe screen to read "Batch A122" and the second line to display "Test running".
Display the USER swipe screen.



Also see

display.settext() (on page 14-75)

display.clear()

This function clears the text from the front-panel USER swipe screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.clear()
```

Example

```
display.clear()
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, "Serial number:")
display.settext(display.TEXT2, localnode.serialno)
```

Clear the USER swipe screen.
Set the first line to read "Serial number:" and the second line to display the serial number of the instrument.

Also see

display.settext() (on page 14-75)

display.delete()

This function allows you to remove a prompt on the front-panel display that was created with `display.prompt()`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.delete(promptID)
```

promptID	The identifier defined by `display.prompt()`

Details

You can use this command to remove the presently displayed prompt.

Example

```
removePrompt3 = display.prompt(display.BUTTONS_NONE, "This prompt will disappear in
    3 seconds")
delay(3)
display.delete(removePrompt3)
```

This example displays a prompt that is automatically removed in three seconds.



Also see

display.prompt() (on page 14-73)

display.input.number()

This function allows you to create a prompt that requests a number from the user on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
numberEntered = display.input.number("dialogTitle")
numberEntered = display.input.number("dialogTitle", numberFormat)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue,
    minimumValue)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue,
    minimumValue, maximumValue)
```

numberEntered	The number that is entered from the front-panel display; `nil` if Cancel is pressed on the keypad
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; can be up to 32 characters
numberFormat	The format of the displayed number: ■ Allow integers (negative or positive) only: `display.NFORMAT_INTEGER` (default) ■ Allow decimal values: `display.NFORMAT_DECIMAL` ■ Display numbers in exponent format: `display.NFORMAT_EXPONENT` ■ Display numbers with prefixes before the units symbol, such as n, m, or µ: `display.NFORMAT_PREFIX`
defaultValue	The value that is initially displayed in the displayed keypad
minimumValue	The lowest value that can be entered
maximumValue	The highest value that can be entered

Details

This command prompts the instrument operator to enter a value.

The prompt is displayed until it has been responded to.

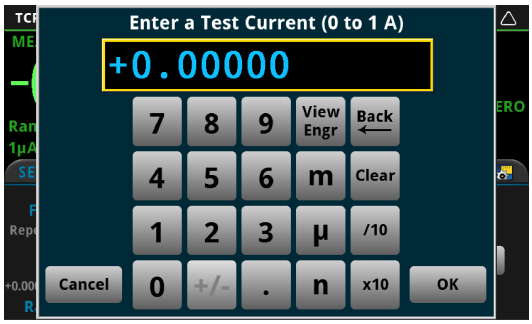
NOTE

On the prompt, the operator can move the cursor in the entry box by touching the screen. The cursor is moved to the spot where the operator touched the screen.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
testcurrent = display.input.number("Enter a Test Current (0 to 1 A)",
   display.NFORMAT_PREFIX, 0, 0, 1)
smu.source.level = testcurrent
```

This example displays a number pad on the screen that defaults to 0 and allows entries from 0 to 1. The number that the operator enters is assigned to the source current level. If the operator enters a value outside of the range, an error message is displayed.



Also see

display.input.option() (on page 14-68)
display.input.prompt() (on page 14-69)
display.input.string() (on page 14-70)

display.input.option()

This function allows you to create an option dialog box with customizable buttons on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.BUTTON_OPTIONn = display.input.option("dialogTitle", "buttonTitle1",
   "buttonTitle2")
display.BUTTON_OPTIONn = display.input.option("dialogTitle", "buttonTitle1",
   "buttonTitle2", "buttonTitleN", ... "buttonTitleN")
```

n	The number of the button that is selected from the front-panel display; `nil` if Cancel is pressed on the keypad; buttons are numbered top to bottom, left to right
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 32 characters
buttonTitle1	A string that contains the name of the first button; up to 15 characters
buttonTitle2	A string that contains the name of the second button; up to 15 characters
buttonTitleN	A string that contains the names of subsequent buttons, where *N* is a number from 3 to 10; you can define up to 10 buttons; each button can be up to 15 characters

Details

Buttons are created from top to bottom, left to right. If you have more than five buttons, they are placed into two columns.

The prompt is displayed until it has been responded to. You can only send one input prompt command at a time.

Example

```
optionID = display.input.option("Select an option", "Apple", "Orange", "Papaya",
   "Pineapple", "Blueberry", "Banana", "Grapes", "Peach", "Apricot", "Guava")
print(optionID)
```

This example displays the following dialog box:



If the user selects `Peach`, the return is `display.BUTTON_OPTION8`.

Also see

display.input.number() (on page 14-66)
display.input.prompt() (on page 14-69)
display.input.string() (on page 14-70)

display.input.prompt()

This function allows you to create a prompt that accepts a user response from the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buttonReturn = display.input.prompt(buttonSet, "dialogTitle")
```

buttonReturn	Indicates which button was pressed: ■ OK: `display.BUTTON_OK` ■ Cancel: `display.BUTTON_CANCEL` ■ Yes: `display.BUTTON_YES` ■ No: `display.BUTTON_NO`
buttonSet	The set of buttons to display: ■ OK button only: `display.BUTTONS_OK` ■ Cancel button only: `display.BUTTONS_CANCEL` ■ OK and Cancel buttons: `display.BUTTONS_OKCANCEL` ■ Yes and No buttons: `display.BUTTONS_YESNO` ■ Yes, No, and Cancel buttons: `display.BUTTONS_YESNOCANCEL`
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 63 characters

Details

This command waits for a user response to the prompt. You can use the text to ask questions that can be used to configure your test.

The prompt is displayed until it has been responded to by the user. You can only send one input prompt command at a time.

Example

```
result = display.input.prompt(display.BUTTONS_YESNO, "Do you want to display the graph
    screen?")
if result == display.BUTTON_YES then
   display.changescreen(display.SCREEN_GRAPH)
end
```

This displays the prompt "Do you want to display the graph screen?" on the front-panel display:



If the operator selects **Yes**, the graph screen is displayed.

Also see

display.input.number() (on page 14-66)
display.input.option() (on page 14-68)
display.input.string() (on page 14-70)

display.input.string()

This function allows you to create a dialog box that requests text from the user through the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
textEntered = display.input.string("dialogTitle")
textEntered = display.input.string("dialogTitle", textFormat)
```

textEntered	The text that is entered from the front-panel display; `nil` if Cancel is pressed on the keypad
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 32 characters
textFormat	The format of the entered text: ■ Allow any characters: `display.SFORMAT_ANY` (default) ■ Allow both upper- and lower-case letters (no special characters): `display.SFORMAT_UPPER_LOWER` ■ Allow only upper-case letters: `display.SFORMAT_UPPER` ■ Allow both upper- and lower-case letters, no special characters, no spaces, and limited to 32 characters: `display.SFORMAT_BUFFER_NAME`

Details

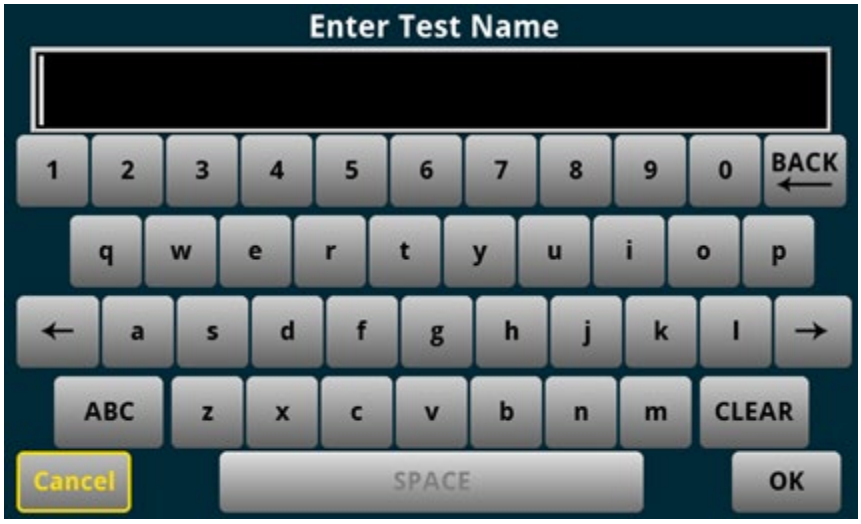
This command creates a prompt to the instrument operator to enter a string value.

The prompt is displayed until it has been responded to. You can only send one input prompt command at a time.

Example

```
value = display.input.string("Enter Test Name", display.SFORMAT_ANY)
print(value)
```

This example displays the prompt "Enter Test Name" and a keyboard that the operator can use to enter a response.



The return is the response from the operator.

Also see

display.input.number() (on page 14-66)
display.input.option() (on page 14-68)
display.input.prompt() (on page 14-69)

display.lightstate

This attribute sets the light output level of the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	display.STATE_LCD_50

Usage

```
brightness = display.lightstate
display.lightstate = brightness
```

brightness	The brightness of the display: ■ Full brightness: `display.STATE_LCD_100` ■ 75% brightness: `display.STATE_LCD_75` ■ 50% brightness: `display.STATE_LCD_50` ■ 25% brightness: `display.STATE_LCD_25` ■ Display off: `display.STATE_LCD_OFF` ■ Display and all indicators off: `display.STATE_BLACKOUT`

Details

This command changes the light output of the front panel when a test requires different instrument illumination levels.

The change in illumination is temporary. The normal backlight settings are restored after a power cycle. You can use this to reset a display that is already dimmed by the front-panel Backlight Dimmer.

NOTE

Screen life is affected by how long the screen is on at full brightness. The higher the brightness setting and the longer the screen is bright, the shorter the screen life.

Example

`display.lightstate = display.STATE_LCD_50`	Set the display brightness to 50%.

Also see

Adjust the backlight brightness and dimmer (on page 3-7)

display.prompt()

This function allows you to create an interactive dialog prompt that displays a custom message on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
promptID = display.prompt(buttonID, "promptText")
```

promptID	A set of characters that identifies the prompt; up to 63 characters
buttonID	The type of prompt to display; choose one of the following options: ■ `display.BUTTONS_NONE` ■ `display.BUTTONS_OK` ■ `display.BUTTONS_CANCEL` ■ `display.BUTTONS_OKCANCEL` ■ `display.BUTTONS_YESNO` ■ `display.BUTTONS_YESNOCANCEL`
promptText	A string that contains the text that is displayed above the prompts

Details

This command displays buttons and text on the front panel. You can set up scripts that respond to the buttons when they are selected.

If you send `display.BUTTONS_NONE`, the operator needs to press the EXIT key to clear the message from the front-panel display. You can also use the `display.delete()` command to remove the prompt.

Only one prompt can be active at a time.

When the user presses a button, the button presses are returned as one of the following options:

- OK: `display.BUTTON_OK`
- Cancel: `display.BUTTON_CANCEL`
- Yes: `display.BUTTON_YES`
- No: `display.BUTTON_NO`

To capture return values, you need to use `display.waitevent()` to wait for the user button selection.

Example

```
smu.source.sweeplinear("test", 1, 10, 10)
display.prompt(display.BUTTONS_YESNO, "Would you like to start the sweep now?")
sweepTest, result = display.waitevent()
if result == display.BUTTON_YES then
      trigger.model.initiate()
end
display.prompt(display.BUTTONS_YESNO, "Would you like to switch to the Graph screen?")
promptID, result = display.waitevent()
if result == display.BUTTON_YES then
   display.changescreen(display.SCREEN_GRAPH)
end
```

Create a linear sweep.
Display the prompt "Would you like to start the sweep now?"
If the user presses Yes, the sweep starts.
If the user presses No, the sweep does not start and the message is removed.
Display the prompt "Would you like to switch to the Graph screen?"
If the user presses Yes, the Graph screen is displayed.
If the user presses No, the user remains on the present screen.

Also see

display.delete() (on page 14-65)
display.waitevent() (on page 14-76)

display.readingformat

This attribute determines the format that is used to display measurement readings on the front-panel display of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	display.FORMAT_PREFIX

Usage

```
format = display.readingformat
display.readingformat = format
```

format	Use exponent format: `display.FORMAT_EXPONENT` Add a prefix to the units symbol, such as k, m, or µ: `display.FORMAT_PREFIX`

Details

This setting persists through `reset()` and power cycles.

When Prefix is selected, prefixes are added to the units symbol, such as k (kilo) or m (milli). When Exponent is selected, exponents are used instead of prefixes. When the prefix option is selected, very large or very small numbers may be displayed with exponents.

Example

`display.readingformat = display.FORMAT_EXPONENT`	Change front-panel display to show readings in exponential format.

Also see

Setting the display format (on page 3-41)

display.settext()

This function defines the text that is displayed on the front-panel USER swipe screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.settext(display.TEXT1, "userDisplayText1")
display.settext(display.TEXT2, "userDisplayText2")
```

userDisplayText1	String that contains the message for the top line of the USER swipe screen (up to 20 characters)
userDisplayText2	String that contains the message for the bottom line of the USER swipe screen (up to 32 characters)

Details

This command defines text messages for the USER swipe screen.

If you enter too many characters, the instrument displays a warning event and shortens the message to fit.

When the instrument is reset, the user test is removed and the USER swipe screen is hidden until another message is defined.

You can send use the following codes to create special characters in the message.

Code	Special character
\018	Ω
\019	°
\020	µ
\021	Thin space
\178	Superscript 2 (for example, x^2)
\179	Superscript 3 (for example, x^3)
\185	Δ
\188	1/x
\189	Ratio

Example

```
display.clear()
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, "A122 \185 A123")
display.settext(display.TEXT2, "Results in \018")
```

Clear the USER swipe screen.
Display the USER swipe screen.
Set the first line to read "A122 Δ A123" and the second line to display ""Results in Ω":



Also see

display.clear() (on page 14-65)
display.changescreen() (on page 14-64)

display.waitevent()

This function causes the instrument to wait for a user to respond to a prompt that was created with a prompt command.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
objectID, subID = display.waitevent()
objectID, subID = display.waitevent(timeout)
```

objectID	A number that identifies the object, such as a prompt message, that is displayed on the front panel
subID	The returned value after a button is pressed on the front panel: ■ `display.BUTTON_YES` ■ `display.BUTTON_NO` ■ `display.BUTTON_OK` ■ `display.BUTTON_CANCEL`
timeout	The amount of time to wait before timing out; time is 0 to 300 s, where the default of 0 waits indefinitely

Details

This command waits until a user responds to a front-panel prompt that was created with the `display.prompt()` command.

Example

```
smu.source.sweeplinear("test", 1, 10, 10)
display.prompt(display.BUTTONS_YESNO, "Would you like to start the sweep now?")
sweepTest, result = display.waitevent()
if result == display.BUTTON_YES then
      trigger.model.initiate()
end
display.prompt(display.BUTTONS_YESNO, "Would you like to switch to the Graph screen?")
promptID, result = display.waitevent()
if result == display.BUTTON_YES then
   display.changescreen(display.SCREEN_GRAPH)
end
```

Create a linear sweep.
Display the prompt "Would you like to start the sweep now?"
If the user presses Yes, the sweep starts.
If the user presses No, the sweep does not start, and the message is removed.
Display the prompt "Would you like to switch to the Graph screen?"
If the user presses Yes, the Graph screen is displayed.
If the user presses No, the user remains on the present screen.

Also see

display.input.prompt() (on page 14-69)
display.prompt() (on page 14-73)

eventlog.clear()

This function clears the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.clear()
```

Details

This command removes all events from the event log, including entries in the front-panel event log.

Also see

eventlog.next() (on page 14-78)
eventlog.save() (on page 14-81)
Using the event log (on page 3-50)

eventlog.getcount()

This function returns the number of unread events in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.getcount()
eventlog.getcount(eventType)
```

eventType	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: ■ Errors only: `eventlog.SEV_ERROR` or `1` ■ Warnings only: `eventlog.SEV_WARN` or `2` ■ Errors and warnings only: `eventlog.SEV_WARN|eventlog.SEV_ERROR` or `3` ■ Information only: `eventlog.SEV_INFO` or `4` ■ Errors and information only: `eventlog.SEV_INFO|eventlog.SEV_ERROR` or `5` ■ Warnings and information only: `eventlog.SEV_INFO|eventlog.SEV_WARN` or `6` ■ All events: `eventlog.SEV_ALL` or `7`

Details

A count finds the number of unread events in the event log. You can specify the event types to return or return the count for all events.

This command reports the number of events that have occurred since the command was last sent or since the event log was last cleared.

Events are read automatically when `localnode.showevents` is enabled. You can also read them individually with `eventlog.next()`.

Example

`print(eventlog.getcount(eventlog.SEV_INFO))`	Displays the present number of unread information messages in the instrument event log. If there are three information messages in the event log, output is: 3

Also see

eventlog.clear() (on page 14-77)
eventlog.next() (on page 14-78)
localnode.showevents (on page 14-105)
Using the event log (on page 3-50)

eventlog.next()

This function returns the oldest unread event message from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventNumber, message, severity, nodeID, timeSeconds, timeNanoSeconds =
  eventlog.next()
eventNumber, message, severity, nodeID, timeSeconds, timeNanoSeconds =
  eventlog.next(eventType)
```

eventNumber	The event number
message	A description of the event
severity	The severity of the event: ■ Error: 1 ■ Warning: 2 ■ Information: 4
nodeID	The TSP-Link node where the event occurred or 0 if the event occurred on the local node
timeSeconds	The seconds portion of the time when the event occurred
timeNanoSeconds	The fractional seconds portion of the time when the event occurred

eventType	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: ■ Errors only: `eventlog.SEV_ERROR` or `1` ■ Warnings only: `eventlog.SEV_WARN` or 2 ■ Errors and warnings only: `eventlog.SEV_WARN|eventlog.SEV_ERROR` or 3 ■ Information only: `eventlog.SEV_INFO` or 4 ■ Errors and information only: `eventlog.SEV_INFO|eventlog.SEV_ERROR` or 5 ■ Warnings and information only: `eventlog.SEV_INFO|eventlog.SEV_WARN` or 6 ■ All events: `eventlog.SEV_ALL` or 7

Details

When an event occurs on the instrument, it is placed in the event log. The `eventlog.next()` command retrieves an unread event from the event log. Once an event is read, it can no longer be accessed remotely. However, it can be viewed on the front panel. When `localnode.showevents` is enabled, this command never returns an event because those events are automatically read and sent to the remote interface.

To read multiple events, execute this command multiple times.

If there are no entries in the event log, the following is returned:

```
0   No error      0      0      0      0
```

If the event type is not defined, an event of any type is returned.

Example

```
print(eventlog.next(5))
```

Get the oldest error or information event from the event log.
Example output:

```
-285 TSP Syntax error at line 1: unexpected symbol near `0' 1 0 1367806152 652040060
```

Also see

eventlog.clear() (on page 14-77)
eventlog.getcount() (on page 14-77)
eventlog.save() (on page 14-81)
Using the event log (on page 3-50)

eventlog.post()

This function allows you to post your own text to the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.post("message")
eventlog.post("message", eventType)
```

message	String that contains the message
eventType	The type of event; if no event is defined, defaults to `eventlog.SEV_INFO`: ■ Error: `eventlog.SEV_ERROR` or `1` ■ Warning: `eventlog.SEV_WARN` or `2` ■ Information: `eventlog.SEV_INFO` or `4`

Details

You can use this command to create your own event log entries and assign a severity level to them. This can be useful for debugging and status reporting.

From the front panel, you must set the Log Warnings and Log Information options on to have the custom warning and information events placed into the event log.

You can send use the following codes to create special characters in the message.

Code	Special character
\018	Ω
\019	°
\020	µ
\021	Thin space
\178	Superscript 2 (for example, x^2)
\179	Superscript 3 (for example, x^3)
\185	Δ
\188	1/x
\189	Ratio

These characters are displayed on the front-panel display only.

Example

```
eventlog.clear()
eventlog.post("Results in \018", eventlog.SEV_ERROR)
print(eventlog.next())
```

Posts an event that states "Results in Ω".
Output:

```
1005 User: Results in Ω
```

Also see

Using the event log (on page 3-50)

eventlog.save()

This function saves the event log to a file on a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.save("filename")
eventlog.save("filename", eventType)
```

filename	A string that represents the name of the file to be saved
eventType	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: ■ Errors only: `eventlog.SEV_ERROR` or `1` ■ Warnings only: `eventlog.SEV_WARN` or 2 ■ Errors and warnings only: `eventlog.SEV_WARN|eventlog.SEV_ERROR` or 3 ■ Information only: `eventlog.SEV_INFO` or 4 ■ Errors and information only: `eventlog.SEV_INFO|eventlog.SEV_ERROR` or 5 ■ Warnings and information only: `eventlog.SEV_INFO|eventlog.SEV_WARN` or 6 ■ All events: `eventlog.SEV_ALL` or 7 (default)

Details

This command saves all event log entries to a USB flash drive.

If you do not define an event type, the instrument saves all event log entries.

The extension `.csv` is automatically added to the file name.

Example

`eventlog.save("/usb1/WarningsApril", eventlog.SEV_WARN)`	Save warning messages to a `.csv` file on a USB flash drive.

Also see

eventlog.next() (on page 14-78)

exit()

This function stops a script that is presently running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
exit()
```

Details

Terminates script execution when called from a script that is being executed.

This command does not wait for overlapped commands to complete before terminating script execution. If overlapped commands are required to finish, use the `waitcomplete()` function before calling `exit()`.

Also see

waitcomplete() (on page 14-319)

file.close()

This function closes a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.close(fileNumber)
```

fileNumber	The file number returned from the `file.open()` function to close

Details

Note that files are automatically closed when the file descriptors are garbage collected.

Example

```
file_num = file.open("/usb1/GENTRIGGER", file.MODE_WRITE)
file.close(file_num)
```

Open the file `GENTRIGGER` for writing, then close it.

Also see

file.open() (on page 14-85)

file.flush()

This function writes buffering data to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.flush(fileNumber)
```

fileNumber	The file number returned from the `file.open()` function to flush

Details

The `file.write()` function may be buffering data instead of writing immediately to the USB flash drive. Use `file.flush()` to flush this data. Data may be lost if the file is not closed or flushed before a script ends.

If there is going to be a time delay before more data is written to a file, flush the file to prevent loss of data because of an aborted test.

Example

```
reset()

-- Fix the range to 10 V
smu.measure.range = 10

-- Set the measurement count to 100
smu.measure.count = 100

-- Set up reading buffers
-- Ensure the default measurement buffer size matches the count
defbuffer1.capacity = 100
smu.measure.read()

testDir = "TestData5"

-- create a directory on the USB drive for the data
file.mkdir(testDir)
fileName = "/usb1/" .. testDir .. "/myTestData.csv"
buffer.save(defbuffer1, fileName)

if file.usbdriveexists() != 0 then
   -- testDir = "TestData3"

   -- Create a directory on the USB drive for the data
   -- file.mkdir(testDir)
   -- Open the file where the data will be stored
   -- fileName = "/usb1/" .. testDir .. "/myTestData.csv"
   fileNumber = file.open(fileName, file.MODE_APPEND)
-- Write header separator to file
   file.write(fileNumber,
   "\n\n========================================================================\n")
-- Write the string data to a file
```

```
   file.write(fileNumber, "Tested to Company Standard ABC.123\n")
-- Ensure a hurry-up of data written to the file before close or script end
   file.flush(fileNumber)
-- Close the data file
   file.close(fileNumber)

end
```

This example writes a string that indicates that the readings were made for a certain reason, such as to test to a company standard.

Also see

None

file.mkdir()

This function creates a directory at the specified path on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`file.mkdir("`*`path`*`")`

path	A string that contains the path of the directory

Details

The directory path must be absolute. The name of the directory must not already exist on the flash drive.

Example

`file.mkdir("TestData")`	Create a new directory named `TestData`.

Also see

None

file.open()

This function opens a file on the USB flash drive for later reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileNumber = file.open("fileName", accessType)
```

fileNumber	A number identifying the open file that you use with other file commands to write, read, flush, or close the file after opening
fileName	A string that contains the file name to open, including the full path of file
accessType	The type of action to do: ■ Append the file: `file.MODE_APPEND` ■ Read the file: `file.MODE_READ` ■ Write to the file: `file.MODE_WRITE`

Details

The path to the file to open must be absolute.

The root folder of the USB flash drive has the following absolute path:

```
"/usb1/"
```

Example

```
file_num = file.open("/usb1/testfile.txt", file.MODE_WRITE)
if file_num != nil then
   file.write(file_num, "This is my test file")
   file.close(file_num)
end
```

Opens file `testfile.txt` for writing. If no errors were found while opening, writes `This is my test file` and closes the file.

Also see

None

file.read()

This function reads data from a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileContents = file.read(fileNumber, readAction)
```

fileContents	The contents of the file based on the *readAction* parameter
fileNumber	The file number returned from the `file.open()` function to read
readAction	The action: ■ Return the next line; returns `nil` if the present file position is at the end of the file: `file.READ_LINE` ■ Return a string that represents the number found; returns an event string if no number was found; returns `nil` if the current file position is at the end of file: `file.READ_NUMBER` ■ Return the whole file, starting at the present position; returns `nil` if the present file position is at the end of the file: `file.READ_ALL`

Details

This command reads data from a file.

Example

```
file_num = file.open("/usb1/testfile.txt", file.MODE_READ)
if file_num != nil then
   file_contents = file.read(file_num, file.READ_ALL)
   file.close(file_num)
end
```

Open `testfile.txt` on the USB flash drive for reading. If it opens successfully, read the entire contents of the file and store it in variable `file_contents`.
Close the file.

Also see

None

file.usbdriveexists()

This function detects if a USB flash drive is inserted into the front-panel USB port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
driveInserted = file.usbdriveexists()
```

driveInserted	`0`: No flash drive is detected `1`: Flash drive is detected

Details

You can call this command from a script to verify that a USB flash drive is inserted before attempting to write data to it.

Example

```
local response
local xMax = 10

for x = 1, xMax do
 -- Make xMax readings and store them in defbuffer1.
 smu.measure.read()
end

if (file.usbdriveexists() == 1) then
   response = display.BUTTON_YES
else
   response = display.input.prompt(display.BUTTONS_YESNO, "Insert a USB flash
    drive.\nPress Yes to write data or No to not write data.")
end

if (response == display.BUTTON_YES) then
   if (file.usbdriveexists() == 1) then
   FileNumber = file.open("/usb1/TenReadings.csv", file.MODE_WRITE)
 file.write(FileNumber,"Reading,Seconds\n")

 -- Print out the measured values in a two-column format.
 print("\nIteration:\tReading:\tTime:\n")

 for i = 1, xMax do
 print(i, defbuffer1[i], defbuffer1.relativetimestamps[i])
 file.write(FileNumber, string.format("%g, %g\r\n",defbuffer1.readings[i],
   defbuffer1.relativetimestamps[i]))
 end
 file.close(FileNumber)
   else
      response = display.input.prompt(display.BUTTONS_OK,
      "No drive detected. Allow more time after inserting a drive.")
   end
end
```

Make measurements.
Verify that a flash drive is inserted into the instrument.
If the flash drive is inserted, write the data to the flash drive.
Print data into a two-column format.
If the flash drive is not inserted after selecting `Yes`, another prompt is displayed.

Also see

None

file.write()

This function writes data to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.write(fileNumber, "string")
```

fileNumber	The file number returned from the `file.open()` function to write
string	A string that contains the data to write to the file

Details

The `file.write()` function may include data that is buffering; it may not be written to the USB flash drive immediately. Use the `file.flush()` function to immediately write buffered data to the drive.

You must use the `file.close()` command to close the file after writing.

Example

```
file_num = file.open("testfile.txt",
   file.MODE_WRITE)
if file_num != nil then
   file.write(file_num, "This is my test file")
   file.close(file_num)
end
```

Opens file `testfile.txt` for writing. If no errors were found while opening, writes `This is my test file` and closes the file.

Also see

file.close() (on page 14-82)
file.flush() (on page 14-83)

format.asciiprecision

This attribute sets the precision (number of digits) for all numbers returned in the ASCII format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	0 (Automatic)

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

precision	A number representing the number of digits to be printed for numbers printed with the `print()`, `printbuffer()`, and `printnumber()` functions; must be a number from `1` to `16`; set to `0` to have the instrument select the precision automatically based on the number that is being formatted

Details

This attribute specifies the precision (number of digits) for numeric data printed with the `print()`, `printbuffer()`, and `printnumber()` functions. The `format.asciiprecision` attribute is only used with the ASCII format. The precision value must be a number from 0 to 16.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x)
```

Output:
```
2.540000000e+00

2.54e+00
```

Also see

format.byteorder (on page 14-89)
format.data (on page 14-90)
print() (on page 14-109)
printbuffer() (on page 14-110)
printnumber() (on page 14-113)

format.byteorder

This attribute sets the binary byte order for the data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	format.LITTLEENDIAN

Usage

```
order = format.byteorder
format.byteorder = order
```

order	Byte order value as follows: ■ Most significant byte first: `format.BIGENDIAN` ■ Least significant byte first: `format.LITTLEENDIAN`

Details

This attribute selects the byte order in which data is written when you are printing data values with the `printnumber()` and `printbuffer()` functions. The byte order attribute is only used with the `format.REAL32` and `format.REAL64` data formats.

If you are sending data to a computer with a Microsoft Windows operating system, select the `format.LITTLEENDIAN` byte order.

Example

`x = 1.23`
`format.data = format.REAL32`
`format.byteorder = format.LITTLEENDIAN`
`printnumber(x)`
`format.byteorder = format.BIGENDIAN`
`printnumber(x)`
	Output depends on the terminal program you use, but will look something like:
	`#0¤p??`
	`#0??p¤`

Also see

format.asciiprecision (on page 14-88)
format.data (on page 14-90)
printbuffer() (on page 14-110)
printnumber() (on page 14-113)

format.data

This attribute sets the data format for data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	format.ASCII

Usage

```
value = format.data
format.data = value
```

value	The format to use for data, set to one of the following values: ■ ASCII format: `format.ASCII` ■ Single-precision IEEE Std 754 binary format: `format.REAL32` ■ Double-precision IEEE Std 754 binary format: `format.REAL64`

Details

You can control the precision of numeric values with the `format.asciiprecision` attribute. If `format.REAL32` or `format.REAL64` is selected, you can select the byte order with the `format.byteorder` attribute.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with `#0` and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

Example

```
format.asciiprecision = 10
x = 3.14159265
format.data = format.ASCII
printnumber(x)
format.data = format.REAL64
printnumber(x)
```

Output a number represented by *x* in ASCII using a precision of 10, then output the same number in binary using double precision format.
Output:
```
3.141592650e+00
#0ñÔÈSû!    @
```

Also see

format.asciiprecision (on page 14-88)
format.byteorder (on page 14-89)
printbuffer() (on page 14-110)
printnumber() (on page 14-113)

fs.chdir()

This function sets the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

workingDirectory = fs.chdir("*path*")

workingDirectory	Returned value containing the working path
path	A string indicating the new working directory path

Details

The new working directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist.

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.
Verify that `/usb1/temp` is a directory and change it to be the current working directory.
Set the variable for the current working directory to be `testPath`.
The return should be:
```
/usb1/temp
```
If `/usb1/temp` is not a directory, set the variable for the current working directory to be `testPath`.
The return should be:
```
/usb1
```

Also see

None

fs.cwd()

This function returns the absolute path of the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.cwd()
```

path	The absolute path of the current working directory

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.
Verify that `/usb1/temp` is a directory and change it to be the current working directory.
Set the variable for the current working directory to be `testPath`.
The return should be:
`/usb1/temp`
If `/usb1/temp` is not a directory, set the variable for the current working directory to be `testPath`.
The return should be:
`/usb1`

Also see

None

fs.is_dir()

This function tests whether or not the specified path refers to a directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_dir("path")
```

status	Whether or not the given path is a directory (`true` or `false`)
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

Example 1

```
print("Is directory: ", fs.is_dir("/usb1/"))
```

Because `/usb1/` is always the root directory of an inserted flash drive, you can use this command to verify that USB flash drive is inserted.

Example 2

```
if fs.is_dir("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.
Check to see if the `temp` directory exists.
If it does not exist, create a directory named `temp`.

Also see

fs.is_file() (on page 14-93)

fs.is_file()

Tests whether the specified path refers to a file (as opposed to a directory).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

status = fs.is_file("*path*")

status	`true` if the given path is a file; otherwise, `false`
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

Example

```
rootDirectory = "/usb1/"
print("Is file: ", fs.is_file(rootDirectory))
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Check to see if `rootDirectory` is a file. Because `rootDirectory` was set up as a directory, the return is `false`.

Also see

fs.is_dir() (on page 14-92)

fs.mkdir()

This function creates a directory at the specified path.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.mkdir("newPath")
```

path	The returned path of the new directory
newpath	Location (path) of where to create the new directory

Details

The directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

Example

```
if fs.is_dir("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.
Check to see if the `temp` directory exists.
If it does not exist, create a directory named `temp`.

Also see

fs.rmdir() (on page 14-95)

fs.readdir()

This function returns a list of the file system entries in the directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
files = fs.readdir("path")
```

files	A table containing the names of all the file system entries in the specified directory
path	The directory path

Details

The directory path may be absolute or relative to the current working directory.

This command is nonrecursive. For example, entries in subfolders are not returned.

An error is logged to the error queue if the given path does not exist or does not represent a directory.

Example

```
rootDirectory = "/usb1/"
entries = fs.readdir(rootDirectory)
count = table.getn(entries)
print("Found a total of "..count.." files and directories")
for i = 1, count do
   print(entries[i])
end
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Set `entries` as the variable for the file system entries in `rootDirectory`.
Return the number of files and directories in the directory.

Also see

None

fs.rmdir()

This function removes a directory from the file system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`fs.rmdir("`*`path`*`")`

`path`	The path of the directory to remove

Details

This path may be absolute or relative to the present working directory.

An error is logged to the error queue if the given path does not exist or does not represent a directory. An error is also logged if the directory is not empty.

Example

```
rootDirectory = "/usb1/"
tempDirectoryName = "temp"
if fs.is_dir(rootDirectory..tempDirectoryName) == false then
    fs.mkdir(rootDirectory..tempDirectoryName)
end
fs.rmdir(rootDirectory..tempDirectoryName)
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Set `tempDirectoryName` to be equivalent to `temp`.
Check to see if `tempDirectoryName` exists.
If it does not exist, create a directory named `temp`.
Remove the directory.

Also see

fs.mkdir() (on page 14-94)

gpib.address

This attribute contains the GPIB address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Nonvolatile memory	18

Usage

```
address = gpib.address
gpib.address = address
```

address	The GPIB address of the instrument (1 to 30)

Details

The address can be set to any address value from 1 to 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow sufficient time for the command to be processed before attempting to communicate with the instrument again.

The `reset()` function does not affect the GPIB address.

Example

`gpib.address = 26`
`address = gpib.address`
`print(address)`
	Sets the GPIB address and reads the address.
	Output:
	26

Also see

GPIB setup (on page 2-9)

lan.ipconfig()

This function specifies the LAN configuration for the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No	Rear panel LAN reset	Nonvolatile memory	lan.MODE_AUTO

Usage

```
method, ipV4Address, subnetMask, gateway = lan.ipconfig()
lan.ipconfig(method)
lan.ipconfig(method, "ipV4Address")
lan.ipconfig(method, "ipV4Address", "subnetMask")
lan.ipconfig(method, "ipV4Address", "subnetMask", "gateway")
```

method	The method for configuring LAN settings; it can be one of the following values: `lan.MODE_AUTO`: The instrument automatically assigns LAN settings `lan.MODE_MANUAL`: You specify the LAN settings
ipV4Address	LAN IP address; must be a string specifying the IP address in dotted decimal notation
subnetMask	The LAN subnet mask; must be a string in dotted decimal notation
gateway	The LAN default gateway; must be a string in dotted decimal notation

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used.

Example

```
lan.ipconfig(lan.MODE_AUTO)
print(lan.ipconfig())
lan.ipconfig(lan.MODE_MANUAL, "192.168.0.7", "255.255.240.0", "192.168.0.3")
print(lan.ipconfig())
```

Set the IP configuration method to automatic. Request the IP configuration. Example output:

```
lan.MODE_AUTO 134.63.78.136   255.255.254.0   134.63.78.1
```

Set the IP configuration method to manual. Request the IP configuration. Output:

```
lan.MODE_MANUAL 192.168.0.7    255.255.240.0  192.168.0.3
```

Also see

None

lan.lxidomain

This attribute contains the LXI domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0

Usage

```
domain = lan.lxidomain
lan.lxidomain = domain
```

domain	The LXI domain number (0 to 255)

Details

This attribute sets the LXI domain number.

All outgoing LXI packets are generated with this domain number. All inbound LXI packets are ignored unless they have this domain number.

Example

`print(lan.lxidomain)`	Displays the LXI domain.

Also see

None

lan.macaddress

This attribute describes the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	Not applicable	Not applicable	Not applicable

Usage

```
MACaddress = lan.macaddress
```

MACaddress	The MAC address of the instrument

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets.

Example

`print(lan.macaddress)`	Returns the MAC address. For example: `08:00:11:00:00:57`

Also see

lan.ipconfig() (on page 14-97)

localnode.access

This attribute contains the type of access users have to the instrument through different interfaces.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.ACCESS_FULL

Usage

```
accessType = localnode.access
localnode.access = accessType
```

accessType	The type of access: ■ Full access for all users from all interfaces: `localnode.ACCESS_FULL` ■ Allows access by one remote interface at a time with logins required from other interfaces: `localnode.ACCESS_EXCLUSIVE` ■ Allows access by one remote interface at a time with passwords required on all interfaces: `localnode.ACCESS_PROTECTED` ■ Allows access by one interface (including the front panel) at a time with passwords required on all interfaces: `localnode.ACCESS_LOCKOUT`

Details

When access is set to full, the instrument accepts commands from any interface with no login or password.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. You do not need a password with this access.

Protected access is similar to exclusive access, except that you must enter a password when logging in.

When the access is set to locked out, a password is required to change interfaces, including the front-panel interface.

Under any access type, if a script is running on one remote interface when a command comes in from another remote interface, the command is ignored and the message "FAILURE: A script is running, use ABORT to stop it" is generated.

Example

```
localnode.access = localnode.ACCESS_LOCKOUT
login admin
logout
```

Set the instrument access to locked out.
Log into the interface using the default password.
Log out of the interface.

Also see

localnode.password (on page 14-101)

localnode.gettime()

This function retrieves the instrument date and time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.gettime()
```

Details

The time is returned in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

```
print(os.date('%c', gettime()))
```

Example output:
```
Tue Dec  5 03:44:37 2017
```

Also see

localnode.settime() (on page 14-104)

localnode.linefreq

This attribute contains the power line frequency setting that is used for NPLC calculations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle	Not applicable	Not applicable

Usage

```
frequency = localnode.linefreq
```

frequency	The detected line frequency: 50 or 60

Details

The instrument automatically detects the power line frequency when the instrument is powered on. Power line frequency can be 50 Hz or 60 Hz.

Example

```
frequency = localnode.linefreq
print(frequency)
```

Reads the line frequency setting.

Also see

None

localnode.model

This attribute stores the model number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
model = localnode.model
```

model	The model number of the instrument

Example

```
print(localnode.model)
```

Outputs the model number of the local node. For example:

```
2470
```

Also see

localnode.serialno (on page 14-103)

localnode.password

This attribute stores the instrument password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	No	Rear-panel LAN reset	Nonvolatile memory	"admin"

Usage

```
localnode.password = "password"
```

password	A string that contains the instrument password (maximum 30 characters)

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

If you forget the password, you can reset the password to the default:

1. On the front panel, press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **Password Reset**.

You can also reset the password and the LAN settings from the rear panel by inserting a straightened paper clip into hole below LAN RESET.

Example

```
localnode.password = "N3wpa55w0rd"
```

Changes the password to `N3wpa55w0rd`.

Also see

localnode.access (on page 14-99)

localnode.prompts

This attribute determines if the instrument generates prompts in response to command messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.DISABLE

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting	Do not generate prompts: `localnode.DISABLE` Generate prompts: `localnode.ENABLE`

Details

When the prompting mode is enabled, the instrument generates prompts when the instrument is ready to take another command. Because the prompt is not generated until the previous command completes, enabling prompts provides handshaking with the instrument to prevent buffer overruns.

When prompting is enabled, the instrument might generate the following prompts:

- **TSP>.** The standard prompt, which indicates that the previous command completed normally.
- **TSP?.** The prompt that is issued if there are unread entries in the event log when the prompt is issued. Like the TSP> prompt, it indicates that processing of the command is complete. It does not mean the previous command generated an error, only that there were still errors in the event log when command processing completed.
- **>>>>.** The continuation prompt, which occurs when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the present command.

Commands do not generate prompts. The instrument generates prompts in response to command completion.

Prompts are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable prompts when the LAN connection is active, they will not be enabled for a subsequent USB connection.

NOTE

Do not disable prompting when using Test Script Builder. Test Script Builder requires prompts and sets the prompting mode automatically. If you disable prompting, the instrument will stop responding when you communicate using Test Script Builder because it is waiting for a common complete prompt from Test Script Builder.

Example

`localnode.prompts = localnode.ENABLE`	Enable prompting.

Also see

tsplink.initialize() (on page 14-296)

localnode.prompts4882

This attribute enables and disables the generation of prompts for IEEE Std 488.2 common commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.ENABLE

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

prompting	IEEE Std 488.2 prompting mode: ■ Disable prompting: `localnode.DISABLE` ■ Enable prompting: `localnode.ENABLE`

Details

When this attribute is enabled, the IEEE Std 488.2 common commands generate prompts if prompting is enabled with the `localnode.prompts` attribute. If `localnode.prompts4882` is enabled, limit the number of `*trg` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When this attribute is disabled, IEEE Std 488.2 common commands will not generate prompts. When using the `*trg` command with a script that executes `trigger.wait()` repeatedly, disable prompting to avoid problems associated with the command interface input queue filling.

Example

`localnode.prompts4882 = localnode.DISABLE`	Disables IEEE Std 488.2 common command prompting.

Also see

localnode.prompts (on page 14-102)

localnode.serialno

This attribute stores the instrument's serial number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
serialno = localnode.serialno
```

serialno	The serial number of the instrument

Details

This indicates the instrument serial number.

Example

```
display.clear()
display.settext(display.TEXT2, "Serial #: " ..localnode.serialno)
display.changescreen(display.SCREEN_USER_SWIPE)
```

Clears the instrument display.
Places the serial number of this instrument on the bottom line of the USER swipe screen display. Displays the USER swipe screen.

Also see

localnode.model (on page 14-101)
localnode.version (on page 14-106)

localnode.settime()

This function sets the date and time of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.settime()
localnode.settime(year, month, day, hour, minute, second)
localnode.settime(hour, minute, second)
localnode.settime(os.time({year, month, day}))
localnode.settime(os.time({year = year, month = month, day = day, hour = hour, min =
   minute, sec = second}))
```

year	Year; must be more than 1970
month	Month (1 to 12)
day	Day (1 to 31)
hour	Hour in 24-hour time format (0 to 23)
minute	Minute (0 to 59)
second	Second (0 to 59)

Details

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

When called without a parameter (the first form), the function returns the current time.

Example 1

```
localnode.settime(2017, 12, 5, 15, 48, 20)
print(localnode.settime())
```

Sets the date and time to December 5, 2017 at 3:48:20 pm and verifies the time.
Output:
`Tue Dec  5 15:48:20 2017`

Example 2

`systemTime = os.time({year = 2018,`
`  month = 3,`
`  day = 31,`
`  hour = 14,`
`  min = 25})`
`localnode.settime(systemTime)`
`print(os.date('%c', gettime()))`
	Sets the date and time to Mar 31, 2018 at 2:25 pm.
	Output:
	`Sat Mar 31 14:25:00 2018`

Also see

localnode.gettime() (on page 14-100)

localnode.showevents

This attribute sets whether or not the instrument automatically outputs generated events to the remote interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	0 (no events sent)

Usage

```
errorMode = localnode.showevents
localnode.showevents = errorMode
```

errorMode	The errors that are returned: ■ No events: `0` ■ Errors only: `1` (`eventlog.SEV_ERROR`) ■ Warnings only: `2` (`eventlog.SEV_WARN`) ■ Errors and warnings: `3` (`eventlog.SEV_ERROR | eventlog.SEV_WARN`) ■ Information only: `4` (`eventlog.SEV_INFO`) ■ Information and errors: `5` (`eventlog.SEV_INFO | eventlog.SEV_ERROR`) ■ Warnings and information: `6` (`eventlog.SEV_INFO | eventlog.SEV_WARN`) ■ All events: `7` (`eventlog.SEV_ALL`)

Details

Enable this attribute to have the instrument output generated events to the remote interface.

Events are output after a command message is executed but before prompts are issued (if prompts are enabled with `localnode.prompts`).

If this attribute is disabled, use `eventlog.next()` to retrieve unread events from the event log.

Events are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable show events when the GPIB connection is active, they will not be enabled for a subsequent USB connection.

Example

```
localnode.showevents = eventlog.SEV_ERROR | eventlog.SEV_INFO
trigger.digin[3].edge = trigger.EDGE_EITHER
```

Send generated error and warning messages.
Example output if the edge cannot be sent to either:

```
1805, Settings conflict: setting input edge when line 3 set for digital
```

Also see

eventlog.clear() (on page 14-77)
localnode.prompts (on page 14-102)

localnode.version

This attribute stores the firmware version of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
version = localnode.version
```

version	Instrument version level

Details

This attribute indicates the version number of the firmware that is presently running in the instrument.

Example

`print(localnode.version)`	Outputs the present version level. Example output: `1.0.0a`

Also see

localnode.model (on page 14-101)
localnode.serialno (on page 14-103)

node[N].execute()

This function starts test scripts on a remote TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
node[N].execute("scriptCode")
```

N	The node number of this instrument (1 to 63)
scriptCode	A string containing the source code

Details

This command is only applicable to TSP-Link systems. You can use this command to use the remote master node to run a script on the specified node. This function does not run test scripts on the master node; only on the subordinate node when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

Example 1

`node[2].execute(sourcecode)`	Runs script code on node 2. The code is in a string variable called `sourcecode`.

Example 2

`node[3].execute("x = 5")`	Runs script code in string constant (`"x = 5"`) to set `x` equal to 5 on node 3.

Example 3

`node[32].execute(TestDut.source)`	Runs the test script stored in the variable `TestDut` (previously stored on the master node) on node 32.

Also see

tsplink.group (on page 14-296)

node[N].getglobal()

This function returns the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

value = node[*N*].getglobal("*name*")

value	The value of the variable
N	The node number of this instrument (1 to 63)
name	The global variable name

Details

This function retrieves the value of a global variable from the run-time environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

`print(node[5].getglobal("test_val"))`	Retrieves and outputs the value of the global variable named `test_val` from node 5.

Also see

node[N].setglobal() (on page 14-108)

node[N].setglobal()

This function sets the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
node[N].setglobal("name", value)
```

N	The node number of this instrument (1 to 63)
name	The global variable name to set
value	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

`node[3].setglobal("x", 5)`	Sets the global variable x on node 3 to the value of 5.

Also see

node[N].getglobal() (on page 14-107)

opc()

This function sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
opc()
```

Details

This function causes the operation complete bit in the Status Event Status Register to be set when all previously started local overlapped commands are complete.

Note that each node independently sets its operation complete bits in its own status model. Any nodes that are not actively performing overlapped commands set their bits immediately. All remaining nodes set their own bits as they complete their own overlapped commands.

Example

`opc()`
`waitcomplete()`
`print("1")`
	Output:
	1

Also see

*OPC (on page 15-6)
Status model (on page 16-1)
waitcomplete() (on page 14-319)

print()

This function generates a response message.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like other scripting environments, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments is separated with a tab character.

Numbers are printed using the `format.asciiprecision` attribute. If you want use Lua formatting, print the return value from the `tostring()` function.

Example 1

```
x = 10
print(x)
```

Example of an output response message:
`10`
Note that your output might be different if you set your ASCII precision setting to a different value.

Example 2

```
x = true
print(tostring(x))
```

Example of an output response message:
`true`

Also see

format.asciiprecision (on page 14-88)

printbuffer()

This function prints data from tables or reading buffer subtables.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

startIndex	Beginning index of the buffer to print; this must be more than one and less than *endIndex*
endIndex	Ending index of the buffer to print; this must be more than *startIndex* and less than the index of the last entry in the tables
bufferVar	Name of first table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
bufferVar2	Second table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
bufferVarN	The last table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
...	One or more tables or reading buffer subtables separated with commas

Details

If *startIndex* is set to less than 1 or if *endIndex* is more than the size of the index, 9.910000e+37 is returned for each value outside the allowed index and an event is generated.

If overlapped commands use the specified reading buffers and the commands are not complete (at least to the specified index), this function outputs data as it becomes available.

When there are outstanding overlapped commands to acquire data, *n* refers to the index that the last entry in the table has after all the readings have completed.

If you pass a reading buffer instead of a reading buffer subtable, the default subtable for that reading buffer is used.

This command generates a single response message that contains all data.

The output of `printbuffer()` is affected by the data format selected by `format.data`. If you set `format.data` to `format.REAL32` or `format.REAL64`, you have fewer options for buffer elements. With these formats, the only buffer elements available are `readings`, `relativetimestamps`, and `extravalues`. If you request a buffer element that is not permitted for the selected data format, the instrument returns `9.91e37`.

You can use the *bufferVar* attributes that are listed in the following table with the print buffer command. For example, if `testData` is the buffer, you can use `testData.dates` attribute to print the date of each reading in the `testData` buffer.

You can use *bufferVar*`.n` to retrieve the number of readings in the specified reading buffer.

Attribute	Description
bufferVar.readings	The readings stored in a specified reading buffer. See bufferVar.readings (on page 14-34).
bufferVar.dates	The dates of readings stored in the reading buffer. See bufferVar.dates (on page 14-24).
bufferVar.statuses	The status values of readings in the reading buffer. See bufferVar.statuses (on page 14-43).
bufferVar.formattedreadings	The stored readings formatted as they appear on the front-panel display. See bufferVar.formattedreadings (on page 14-31).
bufferVar.sourceformattedvalues	The source levels formatted as they appear on the front-panel display when the readings in the reading buffer were acquired. See bufferVar.sourceformattedvalues (on page 14-37).
bufferVar.sourcevalues	The source levels that were being output when readings in the reading buffer were acquired. See bufferVar.sourcevalues (on page 14-41).
bufferVar.sourcestatuses	The source status conditions of the instrument for the reading point. See bufferVar.sourcestatuses (on page 14-38).
bufferVar.times	The time when the instrument made the readings. See bufferVar.times (on page 14-44).
bufferVar.timestamps	The timestamps of readings stored in the reading buffer. See bufferVar.timestamps (on page 14-45).
bufferVar.relativetimestamps	The timestamps, in seconds, when each reading occurred relative to the timestamp of reading buffer entry number 1. See bufferVar.relativetimestamps (on page 14-35).
bufferVar.sourceunits	The units of measure of the source. See bufferVar.sourceunits (on page 14-40).
bufferVar.seconds	The nonfractional seconds portion of the timestamp when the reading was stored in UTC format. See bufferVar.seconds (on page 14-36).
bufferVar.fractionalseconds	The fractional portion of the timestamp (in seconds) of when each reading occurred. See bufferVar.fractionalseconds (on page 14-32).
bufferVar.units	The unit of measure that is stored with readings in the reading buffer. See bufferVar.units (on page 14-47).

Example 1

```
reset()
testData = buffer.make(200)
format.data = format.ASCII
format.asciiprecision = 6
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.readings, testData.units,
      testData.relativetimestamps)
```

Reset the instrument.
Set the data format and ASCII precision.
Use trigger model SimpleLoop to create a 6-count loop with no delays that stores data in the reading buffer `testBuffer`.
Start the trigger model, wait for the commands to complete, and output the readings.
Use of `testData.n` (*`bufferVar`*`.n`) indicates that the instrument should output all readings in the reading buffer. In this example, `testBuffer.n` equals 6.
Example of output data:

```
1.10458e-11, Amp DC, 0.00000e+00, 1.19908e-11, Amp DC, 1.01858e-01, 1.19908e-11, Amp DC,
2.03718e-01, 1.20325e-11, Amp DC, 3.05581e-01, 1.20603e-11, Amp DC, 4.07440e-01, 1.20325e-11,
Amp DC, 5.09299e-01
```

Example 2

```
for x = 1, testData.n do
   printbuffer(x,x,testData, testData.units, testData.relativetimestamps)
end
```

Using the same buffer created in Example 1, output the readings, units and relative timestamps on a separate line for each reading.

```
1.10458e-11, Amp DC, 0.00000e+00
1.19908e-11, Amp DC, 1.01858e-01
1.19908e-11, Amp DC, 2.03718e-01
1.20325e-11, Amp DC, 3.05581e-01
1.20603e-11, Amp DC, 4.07440e-01
1.20325e-11, Amp DC, 5.09299e-01
```

Also see

bufferVar.n (on page 14-33)
bufferVar.readings (on page 14-34)
format.asciiprecision (on page 14-88)
format.byteorder (on page 14-89)
format.data (on page 14-90)
printnumber() (on page 14-113)

printnumber()

This function prints numbers using the configured format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printnumber(value1)
printnumber(value1, value2)
printnumber(value1, ..., valueN)
```

value1	First value to print in the configured format
value2	Second value to print in the configured format
valueN	Last value to print in the configured format
...	One or more values separated with commas

Details

There are multiple ways to use this function, depending on how many numbers are to be printed.

This function prints the given numbers using the data format specified by `format.data` and `format.asciiprecision`.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x, 2.54321, 3.1)
```

Configure the ASCII precision to 10 and set x to 2.54.
Read the value of x based on these settings.
Change the ASCII precision to 3.
View how the change affects the output of x and some numbers.
Output:

```
2.540000000e+00
2.54e+00, 2.54e+00, 3.10e+00
```

Also see

format.asciiprecision (on page 14-88)
format.byteorder (on page 14-89)
format.data (on page 14-90)
print() (on page 14-109)
printbuffer() (on page 14-110)

reset()

This function resets commands to their default settings and clears the buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
reset()
reset(system)
```

system	If the node is the master, the entire system is reset: `true` Only the local group is reset: `false`

Details

The `reset()` command in its simplest form resets the entire TSP-enabled system, including the controlling node and all subordinate nodes.

If you want to reset a specific instrument, use the `node[N].reset()` command. Also use the `node[N].reset()` command to reset an instrument on a subordinate node.

When no value is specified for *system*, the default value is `true`.

You can only reset the entire system using `reset(true)` if the node is the master. If the node is not the master node, executing this command generates an error event.

Example

`reset(true)`	If the node is the master node, the entire system is reset; if the node is not the master node, an error event is generated.

Also see

Resets (on page 3-49)

script.catalog()

This function returns an iterator that can be used in a `for` loop to iterate over all the scripts stored in nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in script.catalog() do body end
```

name	String representing the name of the script returned from `script.catalog()`
body	Code that implements the body of the `for` loop to process the return names from the catalog command

Details

This function accesses the catalog of scripts stored in nonvolatile memory, which allows you to process all scripts in nonvolatile memory. The entries are enumerated in no particular order.

Each time the body of the function executes, *name* takes on the name of one of the scripts stored in nonvolatile memory. The `for` loop repeats until all scripts have been iterated.

Example

`for name in script.catalog() do`
`  print(name)`
`end`
	Retrieve the catalog listing for user scripts. print(name) represent the body parameter shown in the Usage.

Also see

None

script.delete()

This function deletes a script from the run-time memory and nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`script.delete("`*`scriptName`*`")`

`scriptName`	A string that represents the name of the script

Details

When a script is deleted, the global variable referring to this script is also deleted.

You must delete an existing script before you can use the name of that script again. Scripts are not automatically overwritten.

Example

`script.delete("test8")`	Deletes a user script named `test8` from nonvolatile memory and the global variable named `test8`.

Also see

Deleting a user script using a remote interface (on page 13-10)
scriptVar.save() (on page 14-117)

script.load()

This function creates a script from a specified file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.load("file")
scriptVar = script.load("file")
```

file	A string that contains the path and file name of the script file to load; if *scriptVar* is not defined, this name is used as the global variable name for this script
scriptVar	The created script; a global variable with this name is used to reference the script

Details

The name that is used for *scriptVar* must not already exist as a global variable. In addition, the *scriptVar* name must be a global reference and not a local variable, table, or array.

For external scripts, the root folder of the USB flash drive has the absolute path `/usb1/`.

Example

`test8 = script.load("/usb1/testSetup.tsp")`	Loads the script with the file name `testSetup.tsp` that is on the USB flash drive and names it `test8`.

Also see

None

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.run()
scriptVar()
```

scriptVar	The name of the variable that references the script

Details

The *scriptVar*`.run()` function runs the script referenced by *scriptVar*. You can also run the script by using *scriptVar()*.

Example

`test8.run()`	Runs the script referenced by the variable `test8`.

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory or to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.save()
scriptVar.save("filename")
```

scriptVar	The name of variable that references the script
filename	A string that contains the file name to use when saving the script to a USB flash drive

Details

The *scriptVar*.save() function saves a script to nonvolatile memory or a USB flash drive. The root folder of the USB flash drive has the absolute path /usb1/.

If no *filename* is specified, the script is saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

If you set *scriptVar* to autoexec, the script is run when the instrument powers up. You must delete the existing autoexec script before saving the new one. Note that performing a system reset does not delete the autoexec script.

You can add the file extension, but it is not required. The only allowed extension is .tsp (see Example 2).

Example 1

`test8.save()`	Saves the script referenced by the variable test8 to nonvolatile memory.

Example 2

`test8.save("/usb1/myScript.tsp")`	Saves the script referenced by the variable test8 to a file named myScript.tsp on your USB flash drive.

Also see

Working with scripts (on page 13-5)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
code = scriptVar.source
```

code	The body of the script
scriptVar	The name of the variable that references the script that contains the source code

Details

The body of the script is a single string with lines separated by the new line character.

Example

```
print(test7.source)
```

Assuming a script named `test7` was created on the instrument, this example retrieves the source code.

Output:

```
reset()
display.settext(display.TEXT1, "Text on line 1")
display.settext(display.TEXT2, "Text on line 2")
```

Also see

scriptVar.save() (on page 14-117)

smu.breakdownprotection

This attribute allows you to enable the breakdown protection in situations where the current may exceed the programmed current or the limit current value due to a breakdown condition.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	smu.BREAKDOWN_OFF

Usage

```
setting = smu.breakdownprotection
smu.breakdownprotection = setting
```

setting	The breakdown protection setting: ■ Operate breakdown protection automatically (see **Details**): `smu.BREAKDOWN_AUTO` ■ Turn breakdown protection off: `smu.BREAKDOWN_OFF` ■ Turn breakdown protection on (not recommended): `smu.BREAKDOWN_ON`

Details

In some tests, the limited bandwidth of the SMU may cause current to exceed either the programmed current or the limit current value. To prevent this from occurring, you can turn on the breakdown protection function. This adds a 500 Ω resistor in series with the SMU force lead. This resistor limits, at a wide bandwidth, the breakdown current to a maximum value of V_{OUTPUT}/500.

When the breakdown protection is set for AUTO operation, the resistor is in place for the 200 V and 1000 V ranges and current ranges less than or equal to the 10 mA range. Above the 10 mA range or on lower voltage ranges, the breakdown protection resistor is automatically taken out of series with the SMU force lead.

An example of when breakdown protection is appropriate is when you are testing components to specify the DUT breakdown voltage. One method is to test the component at the specified breakdown voltage and determine if the leakage current has exceeded a threshold level. For example, if you are testing a 1000 V rated MOSFET, you can short the gate to the source and apply 1050 V from the drain to the source while measuring the actual drain current. Testing at 1050 V instead of 1000 V provides some assurance that the component both meets and exceeds the breakdown requirement by some user-specified margin. In this test, if the breakdown protection is off, you may find that at the exact moment of component breakdown, the current may exceed the limit current value. With the breakdown function on, the peak current is limited to V_{OUTPUT}/500.

A more comprehensive method of testing components to specify the DUT breakdown voltage is to measure the actual component breakdown voltage. To do this on a 1000 V rated MOSFET, you need to switch the sourcing method to current and the limit voltage to 1100 V (higher than the highest expected breakdown voltage). Sourcing the value of the breakdown current permits the SMU to measure the actual breakdown voltage of the component, which occurs precisely when the SMU reaches the programmed source current. When this operating point is reached, a voltage measurement records the actual breakdown voltage of the component, after which the small source current used to find this breakdown voltage can be reduced to zero while waiting for the next component to test. The entire test should be done quickly with the lowest possible breakdown current to limit device self-heating. When performing this test, if the breakdown protection is off, you may find that at the exact moment of component breakdown, the current may exceed the limit current value. With the breakdown function on, the peak current is limited to V_{OUTPUT}/500.

Example

```
selection = display.input.option("Breakdown Control", "Auto", "Always On", "Always
   Off")

if selection ~= nil then
   if selection == display.BUTTON_OPTION1 then
      smu.breakdownprotection = smu.BREAKDOWN_AUTO
  elseif selection == display.BUTTON_OPTION2 then
      smu.breakdownprotection = smu.BREAKDOWN_ON
   elseif selection == display.BUTTON_OPTION3 then
      smu.breakdownprotection = smu.BREAKDOWN_OFF
   end
end
```

Display a prompt that allows the user to select which type of breakdown protection to set.

Also see

None

smu.interlock.enable

This attribute determines if the output can be turned on when the interlock is not engaged.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	None	Nonvolatile memory	smu.OFF

Usage

```
state = smu.interlock.enable
smu.interlock.enable = state
```

state	Allow the output to be turned on when the interlock is not engaged: `smu.OFF` Only allow the output to be turned on if the interlock is engaged: `smu.ON`

Details

The instrument provides an interlock circuit on the rear panel. You must enable this circuit in order for the instrument to set source voltages greater than ±42 V DC.

When the safety interlock signal is asserted, the following actions occur:

- All voltage ranges of the instrument are available.
- The green front-panel INTERLOCK indicator is on.

The action when the interlock signal is not asserted depends on the Interlock setting. If Interlock is set to Off, if the safety interlock signal is not asserted, the following occurs:

- The nominal output is limited to less than ±42 V.
- The front-panel INTERLOCK indicator is not illuminated.
- You can output voltages less than ±42.

If Interlock is set to On, when the safety interlock signal is not asserted, the following occurs:

- You cannot turn on the source output.
- The front-panel INTERLOCK indicator is not illuminated.
- Whenever the interlock changes state (from asserted to not asserted or vice versa), the output is turned off.

If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, event code 5074, “Output voltage limited by interlock,” is generated.

⚠ WARNING

The 2470 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

Example

```
smu.interlock.enable = smu.ON
```

The source output is disabled unless the interlock is engaged.

Also see

smu.interlock.tripped (on page 14-121)

smu.interlock.tripped

This attribute indicates that the interlock has been tripped.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

interlockStatus = smu.interlock.tripped

interlockStatus	The status of the interlock: ▪ smu.OFF: The interlock is not asserted and the 200 V and 1000 V range is limited; lower voltage ranges are available ▪ smu.ON: The interlock signal is asserted and all voltage ranges are available

Details

This command gives you the status of the interlock. When the safety interlock signal is asserted, all voltage ranges of the instrument are available. However, when the safety interlock signal is not asserted, the 200 V and 1000 V ranges are hardware limited to a nominal output of less than ±42 V.

When the interlock is not asserted:

- The front-panel INTERLOCK indicator is off.
- High voltage ranges are disabled.
- If you attempt to turn on the source with a voltage more than ±21 V, an event message is generated.

Example

print(smu.interlock.tripped)	If the interlock is not asserted, returns smu.OFF. If the interlock is asserted, returns smu.ON.

Also see

smu.interlock.enable (on page 14-120)

smu.measure.autorange

This attribute determines if the measurement range is set manually or automatically for the selected measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
autoRange = smu.measure.autorange
smu.measure.autorange = autoRange
autoRange = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_AUTO)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_AUTO, autoRange)
```

autoRange	Set the measurement range manually: `smu.OFF` Set the measurement range automatically: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

Autorange selects the best range in which to measure the signal that is applied to the input terminals of the instrument. When autorange is enabled, the range increases at 100 percent of range. The range decreases occur when the reading is <10 percent of nominal range.

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.ON
```

Set the measurement function to current.
Set the range to be set automatically.

Also see

Ranges (on page 4-39)
smu.measure.range (on page 14-158)

smu.measure.autorangehigh

When autorange is selected, this attribute represents the highest measurement range that is used when the instrument selects the measurement range automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute Resistance: (RW) Voltage and current: (R)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Resistance: 200e6 Ω

Usage

```
highRange = smu.measure.autorangehigh
smu.measure.autorangehigh = highRange
highRange = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_HIGH)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_HIGH, highRange)
```

highRange	The highest measurement range that is used when the range is set automatically: ■ Current: 1e-8 A to 1 A ■ Resistance: 2 Ω to 200e6 Ω ■ Voltage: 0.20 V to 1000 V
function	The measurement function: ■ Voltage measurement (read only): `smu.FUNC_DC_VOLTAGE` ■ Current measurement (read only): `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command can be written to and read for resistance measurements. For current and voltage measurements, it can only be read.

For current and voltage measurements, the upper limit is controlled by the current or voltage limit.

For resistance measurements, you can use this command when automatic range selection is enabled to put an upper bound on the range that is used for resistance measurements.

The upper limit must be more than the lower limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.autorange = smu.ON`
`print(smu.measure.autorangehigh)`
	Sets the measurement function to voltage and turn autorange on. Check the high range for voltage measurements.

Also see

Ranges (on page 4-39)
reset() (on page 14-114)
smu.measure.autorange (on page 14-122)
smu.reset() (on page 14-169)

smu.measure.autorangelow

This attribute selects the lower limit for measurements of the selected function when the range is selected automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: 10 nA Voltage: 0.2 V

Usage

```
lowRange = smu.measure.autorangerangelow
smu.measure.autorangerangelow = lowRange
lowRange = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_LOW)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_LOW, lowRange)
```

lowRange	The lower limit: ■ Current: 1e-8 to 100 mA ■ Voltage: 200 mV to 200 V
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

You can use this command when automatic range selection is enabled. It prevents the instrument from selecting a range that is below this limit. Because the lowest ranges generally require longer settling times, setting the low limit that is appropriate for your application but above the lowest possible range can make measurements require less settling time.

The lower limit must be less than the upper limit.

Though you can send any value when you send this command, the instrument selects the next highest range value. For example, if you send 15 for the lowest voltage range, the instrument will be set to the 20 V range as the low limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.autorange = smu.ON`
`smu.measure.autorangelow = 2`
	Sets the low range for voltage measurements to 2 V.

Also see

Ranges (on page 4-39)
smu.measure.autorange (on page 14-122)

smu.measure.autorangerebound

This attribute determines if the instrument restores the measure range to match the limit range after making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.autorangerebound
smu.measure.autorangerebound = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_REBOUND)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_REBOUND, state)
```

state	Do not restore the measurement range after each measurement: `smu.OFF` Restore the measurement range after each measurement: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

The effective source limit is the lesser of either the programmed source limit or 105% of the active measure range. If you use fixed measure ranges, the instrument prevents you from selecting different limit and measure ranges. However, if measure autorange is selected, it is possible for the autorange process to cause the ranges to differ because the instrument may go down to a range that is lower than the one on which the source limit is programmed. This causes the effective source limit to drop to 105% of the newly selected measure range. For example, the output may be limited if you make a measurement that causes the range to be lowered and then increase the source level or make a change to the device under test or an external source. In this situation, the source voltage or current is limited to conform with the lower measurement range until another measurement causes the instrument to select a higher range to accommodate the new source value. If no further measurement is made, the source limit may remain limited to the present measurement range indefinitely.

When autorange rebound is enabled, it prevents the source from being limited to a value that is below the source limit. After an autoranged measurement is made, the measure range is restored to match the limit range once the autoranged measurement is complete. This ensures that the source does not limit at less than the full limit setting.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.autorange = smu.ON`
`smu.measure.autorangerebound = smu.ON`
	Set the measurement function to current.
	Set the range to be set automatically.
	Set the measure range to be automatically restored to match the source limit value after each measurement.

Also see

Ranges (on page 4-39)
smu.measure.autorange (on page 14-122)

smu.measure.autozero.enable

This attribute enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
state = smu.measure.autozero.enable
smu.measure.autozero.enable = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_AUTO_ZERO)
smu.measure.setattribute(function, smu.ATTR_MEAS_AUTO_ZERO, state)
```

state	The status of autozero; set to one of the following values: ■ Disable autozero: `smu.OFF` ■ Enable autozero: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The 2470 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

The time to make the reference measurements is in addition to the normal measurement time. If timing is critical, such as in sweeps, you can disable autozero to avoid this time penalty.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.autozero.enable = smu.OFF`	Set autozero off for voltage measurements.

Also see

smu.measure.autozero.once() (on page 14-127)
smu.measure.nplc (on page 14-156)

smu.measure.autozero.once()

This function causes the instrument to refresh the reference and zero measurements once.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.autozero.once()
```

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting for the selected function.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

`smu.measure.autozero.once()`	Do a one-time refresh of the reference and zero measurements.

Also see

Automatic reference measurements (on page 4-44)
smu.measure.autozero.enable (on page 14-126)

smu.measure.configlist.catalog()

This function returns the name of one measure configuration list that is stored on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.catalog()
```

Details

You can use this command to retrieve the names of measure configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no measure configuration lists have been created for the instrument.

Example

`print(smu.measure.configlist.catalog())`	Request the name of one measure configuration list that is stored in the instrument. Send the command again until it returns `nil` to get all stored lists.
`print(smu.measure.configlist.catalog())`	If there are two configuration lists on the instrument. Example output: `testMeasList`
`print(smu.measure.configlist.catalog())`	`myMeasList`
`print(smu.measure.configlist.catalog())`	`nil`

Also see

Configuration lists (on page 4-82)
createconfigscript() (on page 14-52)
smu.measure.configlist.create() (on page 14-128)

smu.measure.configlist.create()

This function creates an empty measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu.measure.configlist.create("listName")`

listName	A string that represents the name of a measure configuration list

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Configuration list names must be unique. For example, the name of a source configuration list cannot be the same as the name of a measure configuration list.

Example

`smu.measure.configlist.create("MyMeasList")`
Create a measure configuration list named `MyMeasList`.

Also see

Configuration lists (on page 4-82)
smu.measure.configlist.catalog() (on page 14-127)
smu.measure.configlist.delete() (on page 14-129)
smu.measure.configlist.query() (on page 14-130)
smu.measure.configlist.recall() (on page 14-131)
smu.measure.configlist.size() (on page 14-132)
smu.measure.configlist.store() (on page 14-133)

smu.measure.configlist.delete()

This function deletes a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.delete("listName")
smu.measure.configlist.delete("listName", index)
```

listName	A string that represents the name of a measure configuration list
index	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

`smu.measure.configlist.delete("myMeasList")`	Delete a measure configuration list named `myMeasList`.
`smu.measure.configlist.delete("myMeasList", 2)`	Delete configuration index 2 from the measure configuration list named `myMeasList`.

Also see

Configuration lists (on page 4-82)
createconfigscript() (on page 14-52)
smu.measure.configlist.create() (on page 14-128)

smu.measure.configlist.query()

This function returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.query("listName", index)
smu.measure.configlist.query("listName", index, "fieldSeparator")
```

listName	A string that represents the name of a measure configuration list
index	A number that defines a specific configuration index in the configuration list
fieldSeparator	String that represents the separator for the data; use one of the following: ■ Comma (default): `,` ■ Semicolon: `;` ■ New line: `\n`

Details

This command recalls data for one configuration index.

For additional information about the information this command recalls when using a configuration list query command, see Settings stored in a measure configuration index (on page 4-83).

Example

```
print(smu.measure.configlist.query("testMeasList", 2, "\n"))
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.
Example output:

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 1.000000e-08
smu.measure.autorange = smu.ON
smu.measure.autorangelow = 1.000000e-08
smu.measure.autozero.enable = smu.ON
smu.measure.displaydigits = smu.DIGITS_5_5
smu.measure.filter.enable = smu.OFF
smu.measure.filter.count = 10
smu.measure.filter.type = smu.FILTER_REPEAT_AVG
smu.measure.limit[1].autoclear = smu.ON
smu.measure.limit[1].enable = smu.OFF
smu.measure.limit[1].high.value = 1.000000e+00
smu.measure.limit[1].low.value = -1.000000e+00
smu.measure.limit[2].autoclear = smu.ON
smu.measure.limit[2].enable = smu.OFF
smu.measure.limit[2].high.value = 1.000000e+00
smu.measure.limit[2].low.value = -1.000000e+00
smu.measure.math.enable = smu.OFF
smu.measure.math.format = smu.MATH_PERCENT
smu.measure.math.mxb.bfactor = 0.000000e+00
smu.measure.math.mxb.mfactor = 1.000000e+00
```

```
smu.measure.math.percent = 1.000000e+00
smu.measure.userdelay[1] = 0.000000e+00
smu.measure.userdelay[2] = 0.000000e+00
smu.measure.userdelay[3] = 0.000000e+00
smu.measure.userdelay[4] = 0.000000e+00
smu.measure.userdelay[5] = 0.000000e+00
smu.measure.nplc = 1.000000e+00
smu.measure.offsetcompensation = smu.OFF
smu.measure.sense = smu.SENSE_2WIRE
smu.terminals = smu.TERMINALS_FRONT
smu.measure.unit = smu.UNIT_AMP
smu.measure.rel.enable = smu.OFF
smu.measure.rel.level = 0.000000e+00
```

Also see

Configuration lists (on page 4-82)
createconfigscript() (on page 14-52)
smu.measure.configlist.create() (on page 14-128)

smu.measure.configlist.recall()

This function recalls a configuration index in a measure configuration list and an optional source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.recall("listName")
smu.measure.configlist.recall("listName", index)
smu.measure.configlist.recall("listName", index, "sourceListName")
smu.measure.configlist.recall("listName", index, "sourceListName", sourceIndex)
```

listName	A string that represents the name of a measure configuration list
index	A number that defines a specific configuration index in the measure configuration list
sourceListName	A string that represents the name of a source configuration list
sourceIndex	A number that defines a specific configuration index in the source configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a measure configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified measure configuration list.

You can optionally specify a source configuration list and index to recall with the measure settings. If you do not specify a source index, the source index defaults to match the measure index. Specify a measure and source list together with this command to allow the instrument to coordinate the application of the settings in the two lists appropriately. If you do not need the source and measure configuration lists coordinated, you can specify just the measure configuration list with this command and use the smu.source.configlist.recall() (on page 14-175) command to recall source settings separately in your application.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, an error message is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. A single index stores the settings associated with a single measure function.

NOTE

To recall a source configuration list separately (not with this command), recall the source configuration list before the measure configuration list. This order ensures that dependencies between source and measure settings will be properly handled.

This command recalls data for one configuration index from the specified measure configuration list and from the source configuration list (if specified).

For additional information about the information this command recalls when using a configuration list query command, see Settings stored in a measure configuration index (on page 4-83).

Example

`smu.measure.configlist.recall("MyMeasList")`	Because an index was not specified, this command recalls configuration index 1 from a configuration list named `MyMeasList`.
`smu.measure.configlist.recall("MyMeasList", 5)`	Recalls configuration index 5 in a configuration list named `MyMeasList`.

Also see

Configuration lists (on page 4-82)
createconfigscript() (on page 14-52)
smu.measure.configlist.create() (on page 14-128)
smu.measure.configlist.store() (on page 14-133)

smu.measure.configlist.size()

This function returns the size (number of configuration indexes) of a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
indexCount = smu.measure.configlist.size("listName")
```

indexCount	A number that represents the total count of indexes stored in the specified measure configuration list
listName	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration indexes) of a measure configuration list.

The size of the list is equal to the number of configuration indexes in a configuration list.

Example

```
print(smu.measure.configlist.size("testMeasList"))
```

Returns the number of configuration indexes in a measure configuration list named `testMeasList`.
Example output:
1

Also see

Configuration lists (on page 4-82)
createconfigscript() (on page 14-52)
smu.measure.configlist.create() (on page 14-128)

smu.measure.configlist.store()

This function stores the active measure settings into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.store("listName")
smu.measure.configlist.store("listName", index)
```

listName	A string that represents the name of a measure configuration list
index	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active measure settings to a configuration index in a configuration list. If the index parameter is not provided, the new settings are appended to the end of the list. The index only stores the active settings for a single active measure function.

Configuration lists are not saved when the instrument is turned off or reset. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Refer to Settings stored in a measure configuration index (on page 4-83) for a complete list of measure settings that the instrument stores.

Example

`smu.measure.configlist.store("MyConfigList")`	Stores the active settings of the instrument to the end of the configuration list `MyConfigList`.
`smu.measure.configlist.store("MyConfigList", 5)`	Stores the active settings of the instrument to configuration index 5 in the measure configuration list `MyConfigList`.

Also see

Configuration lists (on page 4-82)
createconfigscript() (on page 14-52)
smu.measure.configlist.create() (on page 14-128)

smu.measure.configlist.storefunc()

This function allows you to store the settings for a measure function into a measure configuration list whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
smu.measure.configlist.storefunc("listName", function)
smu.measure.configlist.storefunc("listName", function, index)
```

listName	Name of the configuration list in which to store the function settings
function	The measure function settings to save into the configuration list: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`
index	The number of the configuration list index in which to store the settings

Details

You must create the configuration list before using this command.

If *index* is not specified, the settings are stored to the next available index in the configuration list.

Example

```
smu.measure.configlist.create("MyMeasList")
smu.measure.configlist.storefunc("MyMeasList", smu.FUNC_DC_VOLTAGE)
```

Create a measure configuration list named `MyMeasList`.
Store the attributes for the DC Voltage settings in index 1.

Also see

smu.measure.configlist.create() (on page 14-128)
smu.measure.setattribute() (on page 14-165)

smu.measure.count

This attribute sets the number of measurements to make when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
count = smu.measure.count
smu.measure.count = count
count = smu.measure.getattribute(function, smu.ATTR_MEAS_COUNT)
smu.measure.setattribute(function, smu.ATTR_MEAS_COUNT, count)
```

count	Number of measurements (1 to 300,000 or buffer capacity)
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command sets the number of measurements that are made when a measurement is requested. This command does not affect the trigger model.

When `smu.measure.count` or if the function for `smu.measure.setattribute` is the active function, this command sets the count for all measure functions. When you send `smu.measure.setattribute` for a function that is not active, only the count for the specified function is changed.

If you set the count to a value that is larger than the capacity of the reading buffer and the buffer fill mode is set to continuous, the buffer wraps until the number of readings specified have occurred. The earliest readings in the count are overwritten. If the buffer is set to fill once, readings stop when the buffer is filled, even if the count is not complete.

NOTE

To get better performance from the instrument, use the SimpleLoop trigger-model template instead of using the count command.

Example 1

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1
smu.measure.count = 200

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

-- Turn on output and initiate readings
smu.source.output = smu.ON
smu.measure.read(defbuffer1)

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
      print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

-- Discharge the capacitor to 0 V and turn off the output
smu.source.level = 0
delay(2)
smu.source.output = smu.OFF
```

This example uses `smu.measure.count` to do a capacitor test. This outputs 200 readings that are similar to the following output:

```
Rdg #   Time (s)   Current (A)
1   0   8.5718931952528e-11
2   0.151875   1.6215984111057e-10
3   0.303727   1.5521139928865e-10
. . .
198   29.91579194   1.5521250951167e-10
199   30.067648716   1.4131290582142e-10
200   30.219497716   1.5521067764368e-10
```

Example 2

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

-- Turn on output and initiate readings
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
      print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

-- Discharge the capacitor to 0 V and turn off the output
smu.source.level = 0
delay(2)
smu.source.output = smu.OFF
```

This example demonstrates how to use the SimpleLoop trigger-model template instead of `smu.measure.count` to do a capacitor test. Similar to Example 1, this also outputs 200 readings that are similar to the following:

```
Rdg #   Time (s)   Current (A)
1   0   8.5718931952528e-11
2   0.151875   1.6215984111057e-10
3   0.303727   1.5521139928865e-10
. . .
198   29.91579194   1.5521250951167e-10
199   30.067648716   1.4131290582142e-10
200   30.219497716   1.5521067764368e-10
```

Also see

smu.measure.read() (on page 14-159)
trigger.model.load() — SimpleLoop (on page 14-248)

smu.measure.displaydigits

This attribute determines the number of digits that are displayed for measurements on the front panel for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.DIGITS_5_5

Usage

```
digits = smu.measure.displaydigits
smu.measure.displaydigits = digits
digits = smu.measure.getattribute(function, smu.ATTR_MEAS_DIGITS)
smu.measure.setattribute(function, smu.ATTR_MEAS_DIGITS, digits)
```

digits	6½ display digits: `smu.DIGITS_6_5` 5½ display digits: `smu.DIGITS_5_5` 4½ display digits: `smu.DIGITS_4_5` 3½ display digits: `smu.DIGITS_3_5`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

The change in digits occurs the next time a measurement is made.

To change the number of digits returned in a remote command reading, use `format.asciiprecision`.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.displaydigits = smu.DIGITS_6_5
```

Set the measurement function to voltage with a front-panel display resolution of 6½.

Also see

format.asciiprecision (on page 14-88)

smu.measure.filter.count

This attribute sets the number of measurements that are averaged when filtering is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	10

Usage

```
filterCount = smu.measure.filter.count
smu.measure.filter.count = filterCount
filterCount = smu.measure.getattribute(function, smu.ATTR_MEAS_FILTER_COUNT)
smu.measure.setattribute(function, smu.ATTR_MEAS_FILTER_COUNT, filterCount)
```

filterCount	The number of readings required for each filtered measurement (1 to 100)
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. When the filter count is larger, more filtering is done, and the data is less noisy.

This command is set for the selected function.

Example

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.filter.count = 10
smu.measure.filter.type = smu.FILTER_MOVING_AVG
smu.measure.filter.enable = smu.ON
```

Set the measurement function to current.
Set the averaging filter type to moving average, with a filter count of 10.
Enable the averaging filter.

Also see

Filtering measurement data (on page 5-24)
smu.measure.filter.enable (on page 14-139)
smu.measure.filter.type (on page 14-140)

smu.measure.filter.enable

This attribute enables or disables the averaging filter for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
filterState = smu.measure.filter.enable
smu.measure.filter.enable = filterState
filterState = smu.measure.getattribute(function, smu.ATTR_MEAS_FILTER_ENABLE)
smu.measure.setattribute(function, smu.ATTR_MEAS_FILTER_ENABLE, filterState)
```

filterState	The filter status: ■ Disable the filter: `smu.OFF` ■ Enable the filter: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command enables or disables the averaging filter. When this is enabled, the reading returned by the instrument is an averaged value, taken from multiple measurements. The settings of the filter count and filter type for the selected measure function determines how the reading is averaged.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.filter.count = 10`
`smu.measure.filter.type = smu.FILTER_MOVING_AVG`
`smu.measure.filter.enable = smu.ON`
	Set the measurement function to current.
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 5-24)
smu.measure.filter.count (on page 14-139)
smu.measure.filter.type (on page 14-140)

smu.measure.filter.type

This attribute sets the type of averaging filter that is used for the selected measure function when the measurement filter is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.FILTER_REPEAT_AVG

Usage

```
filterType = smu.measure.filter.type
smu.measure.filter.type = filterType
filterType = smu.measure.getattribute(function, smu.ATTR_MEAS_FILTER_TYPE)
smu.measure.setattribute(function, smu.ATTR_MEAS_FILTER_TYPE, filterType)
```

filterType	The filter type to use when filtering is enabled; set to one of the following values: ■ Moving average filter: `smu.FILTER_MOVING_AVG` ■ Repeat filter: `smu.FILTER_REPEAT_AVG`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

You can select one of two types of averaging filters: repeating average or moving average.

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed, and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

NOTE

When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.filter.count = 10`
`smu.measure.filter.type = smu.FILTER_MOVING_AVG`
`smu.measure.filter.enable = smu.ON`
	Set the measurement function to current.
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 5-24)
smu.measure.filter.count (on page 14-139)
smu.measure.filter.enable (on page 14-139)

smu.measure.func

This attribute selects the active measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.FUNC_DC_CURRENT

Usage

```
mFunction = smu.measure.func
smu.measure.func = mFunction
```

mFunction	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

Set this command to the type of measurement you want to make.

Reading this command returns the measure function that is presently active.

When you select a function, settings for other commands that are related to the function become active. For example, assume that:

- You selected the current function and set the math function to reciprocal.
- You changed to the voltage function and set the math function to percent.

If you return to the current function, the math function returns to reciprocal. If you then switch from the current function to the voltage function, the math function returns to percent. All attributes that begin with `smu.measure.` are saved with the active measure function unless otherwise indicated in the command description.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.math.format = smu.MATH_PERCENT
smu.measure.math.enable = smu.ON
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.math.format = smu.MATH_RECIPROCAL
smu.measure.math.enable = smu.ON
print(smu.measure.math.format)
smu.measure.func = smu.FUNC_DC_VOLTAGE
print(smu.measure.math.format)
```

Sets the instrument to measure voltage and set the math format to percent and enable the math functions.
Set the instrument to measure resistance and set the math format to reciprocal and enable the math functions.
Print the math format while the resistance measurement function is selected. The output is:
`smu.MATH_RECIPROCAL`
Change the function to voltage. Print the math format. The output is:
`smu.MATH_PERCENT`

Also see

Making resistance measurements (on page 4-26)
Source and measure using TSP commands (on page 4-34)

smu.measure.getattribute()

This function returns the setting for a function attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = smu.measure.getattribute(function, setting)
```

value	The attribute value
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`
setting	The attribute for the function; refer to smu.measure.setattribute() (on page 14-165) for available settings

Details

You can retrieve one attribute at a time.

Example

```
print(smu.measure.getattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_RANGE))
print(smu.measure.getattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_NPLC))
print(smu.measure.getattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_DIGITS))
```

Retrieve the range, NPLC, and digits settings for the DC voltage function.
Example return:

```
0.02
1
smu.DIGITS_4_5
```

Also see

smu.measure.setattribute() (on page 14-165)

smu.measure.limit[Y].audible

This attribute determines if the instrument beeper sounds when a limit test passes or fails.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.AUDIBLE_NONE

Usage

```
state = smu.measure.limit[Y].audible
smu.measure.limit[Y].audible = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_AUDIBLE_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_AUDIBLE_Y, state)
```

state	When the beeper sounds: ■ Never: smu.AUDIBLE_NONE ■ On test failure: smu.AUDIBLE_FAIL ■ On test pass: smu.AUDIBLE_PASS
Y	Limit number: 1 or 2
function	The measurement function: ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

The tone and length of beeper cannot be adjusted.

Example

See smu.measure.limit[Y].fail (on page 14-147) for an example of how to use this command.

Also see

smu.measure.limit[Y].enable (on page 14-146)

smu.measure.limit[Y].autoclear

This attribute indicates if the test result for limit *Y* should be cleared automatically or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
value = smu.measure.limit[Y].autoclear
smu.measure.limit[Y].autoclear = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y, value)
```

value	The auto clear setting: ■ Disable: smu.OFF ■ Enable: smu.ON
Y	Limit number: 1 or 2
function	The measurement function: ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

When auto clear is set to on, limit conditions are cleared automatically after each measurement. If you are making a series of measurements, the instrument shows the limit test result of the last measurement for the pass or fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the auto clear setting to off. When this is set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The auto clear setting affects both the high and low limits.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT` `smu.measure.limit[1].autoclear = smu.ON`	Turns on autoclear for limit 1 when measuring DC current.

Also see

smu.measure.limit[Y].clear() (on page 14-145)

smu.measure.limit[Y].clear()

This function clears the results of the limit test defined by *Y* for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu.measure.limit[Y].clear()`

Y	Limit number: 1 or 2

Details

Use this command to clear the test results of limit *Y* when the limit auto clear option is turned off. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, turn the auto clear option on.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT` `smu.measure.limit[2].clear()`	Clears the test result for the high and low limit 2 for current measurements.

Also see

smu.measure.limit[Y].autoclear (on page 14-144)

smu.measure.limit[Y].enable

This attribute enables or disables a limit test on the measurement from the selected measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.limit[Y].enable
smu.measure.limit[Y].enable = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_ENABLE_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_ENABLE_Y, state)
```

state	Disable the test: `smu.OFF` Enable the test: `smu.ON`
Y	Limit number: 1 or 2
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command enables or disables a limit test for the selected measurement function. When this attribute is enabled, the limit *Y* testing occurs on each measurement made by the instrument. Limit *Y* testing compares the measurements to the high-limit and low-limit values. If a measurement falls outside these limits, the test fails.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.limit[1].enable = smu.ON`	Enable testing for limit 1 when measuring voltage.

Also see

smu.measure.limit[Y].autoclear (on page 14-144)
smu.measure.limit[Y].clear() (on page 14-145)
smu.measure.limit[Y].fail (on page 14-147)
smu.measure.limit[Y].high.value (on page 14-149)
smu.measure.limit[Y].low.value (on page 14-150)

smu.measure.limit[Y].fail

This attribute queries the results of a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Function change	Not applicable	Not applicable

Usage

```
result = smu.measure.limit[Y].fail
result = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_FAIL_Y)
```

result	The results of the limit test for limit *Y*: ■ `smu.FAIL_NONE`: Test passed; measurement under or equal to the high limit ■ `smu.FAIL_HIGH`: Test failed; measurement exceeded high limit ■ `smu.FAIL_LOW`: Test failed; measurement exceeded low limit ■ `smu.FAIL_BOTH`: Test failed; measurement exceeded both limits
Y	Limit number: 1 or 2
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command queries the result of a limit test for the selected measurement function.

The response message indicates if the limit test passed or how it failed (on the high or low limit).

If autoclear is set to off, reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command. To automatically clear the results, set auto clear on.

If auto clear is set to on and you are making a series of measurements, the last measurement limit determines the fail indication for the limit. If auto clear is turned off, the results return a test fail if any of one of the readings failed.

To use this attribute, you must set the limit state to on.

If the readings are stored in a reading buffer, you can use the *bufferVar*`.statuses` command to see the results.

Example

This example enables limits 1 and 2 for voltage, measurements. Limit 1 is checking for readings to be between 3 and 5 V, while limit 2 is checking for the readings to be between 1 and 7 V. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
reset()
-- set the instrument source current
smu.source.func = smu.FUNC_DC_CURRENT
-- set the instrument to measure voltage
smu.measure.func = smu.FUNC_DC_VOLTAGE
-- set the range to 10 V
smu.measure.range = 10
-- set the nplc to 0.1
smu.measure.nplc = 0.1
-- disable auto clearing for limit 1
smu.measure.limit[1].autoclear = smu.OFF
-- set high limit on 1 to fail if reading exceeds 5 V
smu.measure.limit[1].high.value = 5
-- set low limit on 1 to fail if reading is less than 3 V
smu.measure.limit[1].low.value = 3
--- set the beeper to sound if the reading exceeds the limits for limit 1
smu.measure.limit[1].audible = smu.AUDIBLE_FAIL
-- enable limit 1 checking for voltage measurements
smu.measure.limit[1].enable = smu.ON
-- disable auto clearing for limit 2
smu.measure.limit[2].autoclear = smu.OFF
-- set high limit on 2 to fail if reading exceeds 7 V
smu.measure.limit[2].high.value = 7
-- set low limit on 2 to fail if reading is less than 1 V
smu.measure.limit[2].low.value = 1
-- enable limit 2 checking for voltage measurements
smu.measure.limit[2].enable = smu.ON
-- set the measure count to 50
smu.measure.count = 50
-- create a reading buffer that can store 100 readings
LimitBuffer = buffer.make(100)
-- make 50 readings and store them in LimitBuffer
smu.measure.read(LimitBuffer)
-- Check if any of the 50 readings were outside of the limits
print("limit 1 results = " .. smu.measure.limit[1].fail)
print("limit 2 results = " .. smu.measure.limit[2].fail)
-- clear limit 1 conditions
smu.measure.limit[1].clear()
-- clear limit 2 conditions
smu.measure.limit[2].clear()
```

Example output that shows all readings are within limit values (all readings between 3 V and 5 V):

```
limit 1 results = smu.FAIL_NONE
limit 2 results = smu.FAIL_NONE
```

Example output showing at least one reading failed limit 1 high values (a 6 V reading would cause this condition or a reading greater than 5 V but less than 7 V):

```
limit 1 results = smu.FAIL_HIGH
limit 2 results = smu.FAIL_NONE
```

Example output showing at least one reading failed limit 1 and 2 low values (a 0.5 V reading would cause this condition or a reading less than 1 V):

```
limit 1 results = smu.FAIL_LOW
limit 2 results = smu.FAIL_LOW
```

Also see

bufferVar.statuses (on page 14-43)
Limit testing and binning (on page 4-66)
smu.measure.limit[Y].enable (on page 14-146)

smu.measure.limit[Y].high.value

This attribute specifies the upper limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
highLimit = smu.measure.limit[Y].high.value
smu.measure.limit[Y].high.value = highLimit
highLimit = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_HIGH_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_HIGH_Y, highLimit)
```

highLimit	The value of the high limit (–9.99999e+11 to +9.99999e+11)
Y	Limit number: 1 or 2
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command sets the high limit for the limit *Y* test for the selected measurement function. When limit *Y* testing is enabled, the instrument generates a fail indication when the measurement value is more than this value.

Example

See the example in smu.measure.limit[Y].fail (on page 14-147).

Also see

smu.measure.limit[Y].enable (on page 14-146)

smu.measure.limit[Y].low.value

This attribute specifies the lower limit for limit tests.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	–1

Usage

```
lowLimit = smu.measure.limit[Y].low.value
smu.measure.limit[Y].low.value = lowLimit
lowLimit = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_LOW_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_LOW_Y, lowLimit)
```

lowLimit	The low limit value of limit *Y* (-9.99999E+11 to 9.99999E+11)
Y	Limit number: 1 or 2
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command sets the lower limit for the limit *Y* test for the selected measure function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

Example

See the example in smu.measure.limit[Y].fail (on page 14-147).

Also see

smu.measure.limit[Y].enable (on page 14-146)

smu.measure.math.enable

This attribute enables or disables math operations on measurements for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
value = smu.measure.math.enable
smu.measure.math.enable = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_ENABLE)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_ENABLE, value)
```

value	The math enable setting: ■ Disable: `smu.OFF` ■ Enable: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format = smu.MATH_PERCENT`
`smu.measure.math.enable = smu.ON`
	When voltage measurements are made, the math format is enabled and set to percent.

Also see

Calculations that you can apply to measurements (on page 4-50)
smu.measure.math.format (on page 14-152)

smu.measure.math.format

This attribute specifies which math operation is performed on measurements when math operations are enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.MATH_PERCENT

Usage

```
operation = smu.measure.math.format
smu.measure.math.format = operation
operation = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_FORMAT)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_FORMAT, operation)
```

operation	Math operation to be performed on measurements: ■ y = mx+b: `smu.MATH_MXB` ■ Percent: `smu.MATH_PERCENT` ■ Reciprocal: `smu.MATH_RECIPROCAL`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This specifies which math operation is performed on measurements for the selected measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Displays measurements as the percentage of deviation from a specified reference constant.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is `1/X`, where `X` is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format = smu.MATH_RECIPROCAL`
`smu.measure.math.enable = smu.ON`
	Enables the reciprocal math operation on voltage measurements.

Also see

Calculations that you can apply to measurements (on page 4-50)
smu.measure.math.enable (on page 14-151)
smu.measure.math.mxb.bfactor (on page 14-153)
smu.measure.math.mxb.mfactor (on page 14-154)
smu.measure.math.percent (on page 14-155)

smu.measure.math.mxb.bfactor

This attribute specifies the offset, b, for the y = mx + b operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
value = smu.measure.math.mxb.bfactor
smu.measure.math.mxb.bfactor = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_MXB_BF)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_MXB_BF, value)
```

value	The offset for the y = mx + b operation; the valid range is –1e12 to +1e12
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This attribute specifies the offset (b) for an mx + b operation.

The mx + b math operation lets you manipulate normal display readings (x) mathematically based on the calculation:

$$y = mx + b$$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format = smu.MATH_MXB`
`smu.measure.math.mxb.mfactor = 0.80`
`smu.measure.math.mxb.bfactor = 50`
`smu.measure.math.enable = smu.ON`
	Set the measurement function to voltage.
	Set the math operation to mx+b.
	Set the scale factor for the mx +b operation to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 4-50)
smu.measure.math.enable (on page 14-151)
smu.measure.math.mxb.mfactor (on page 14-154)

smu.measure.math.mxb.mfactor

This attribute specifies the scale factor, m, for the y = mx + b math operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
value = smu.measure.math.mxb.mfactor
smu.measure.math.mxb.mfactor = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_MXB_MF)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_MXB_MF, value)
```

value	The scale factor; the valid range is –1e12 to +1e12
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command sets the scale factor (m) for an mx + b operation for the selected measurement function.

The mx + b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$$y = mx + b$$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`	Set the measurement function to voltage.
`smu.measure.math.format = smu.MATH_MXB`	Set the math operation to mx+b.
`smu.measure.math.mxb.mfactor = 0.80`	Set the scale factor for the mx+b operation to 0.80.
`smu.measure.math.mxb.bfactor = 50`	Set the offset factor to 50.
`smu.measure.math.enable = smu.ON`	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 4-50)
smu.measure.math.enable (on page 14-151)
smu.measure.math.mxb.bfactor (on page 14-153)

smu.measure.math.percent

This attribute specifies the reference constant that is used when math operations are set to percent.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
value = smu.measure.math.percent
smu.measure.math.percent = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_PERCENT)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_PERCENT, value)
```

value	The reference used when the math operation is set to percent; the range is –1e12 to +1e12
function	The measurement function: ▪ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ▪ Current measurement: `smu.FUNC_DC_CURRENT` ▪ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This is the constant that is used when the math operation is set to percent for the selected measurement function.

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

$$\text{Percent} = \left(\frac{\text{input - reference}}{\text{reference}}\right) \times 100\%$$

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format = smu.MATH_PERCENT`
`smu.measure.math.percent = 50`
`smu.measure.math.enable = smu.ON`
	Set the measurement function to voltage.
	Set the math operations to percent.
	Set the percentage value to 50 for voltage measurements.
	Enable math operations.

Also see

Calculations that you can apply to measurements (on page 4-50)
smu.measure.math.enable (on page 14-151)
smu.measure.math.format (on page 14-152)

smu.measure.nplc

This command sets the time that the input signal is measured for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
nplc = smu.measure.nplc
smu.measure.nplc = nplc
nplc = smu.measure.getattribute(function, smu.ATTR_MEAS_NPLC)
smu.measure.setattribute(function, smu.ATTR_MEAS_NPLC, nplc)
```

nplc	The number of power line cycles: 0.01 to 10
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified as the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50). For 60 Hz, if you set the NPLC to 0.1, the measure time is 1.667 ms.

This command is set for the measurement of specific functions (current, resistance, or voltage).

The shortest amount of time results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.nplc = 0.5`	Set the measurement function to DC Voltage. Set the NPLC value to 0.5, which is 0.0083 seconds (0.5/60).

Also see

Using NPLCs to adjust speed and accuracy (on page 5-9)

smu.measure.offsetcompensation

This attribute determines if offset compensation is used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.offsetcompensation
smu.measure.offsetcompensation = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_OFFSET_COMP)
smu.measure.setattribute(function, smu.ATTR_MEAS_OFFSET_COMP, state)
```

state	Disable offset compensation: `smu.OFF` Enable offset compensation: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

The voltage offsets caused by the presence of thermoelectric EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

This feature is only available for resistance measurements or when the `smu.measure.unit` is set to `smu.UNIT_OHM`.

Example

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.sense = smu.SENSE_4WIRE
smu.measure.offsetcompensation = smu.ON
smu.source.output = smu.ON
print(smu.measure.read())
smu.source.output = smu.OFF
```

Sets the measurement function to resistance. Set the instrument for 4-wire measurements and turn offset compensation on. Turn on the source, make a measurement, and turn the source off.

Example output:
`81592000`

Also see

None

smu.measure.range

This attribute determines the positive full-scale measure range.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: 10 nA Resistance: 200,000 Ω Voltage: 0.2 V

Usage

```
rangeValue = smu.measure.range
smu.measure.range = rangeValue
rangeValue = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE, rangeValue)
```

rangeValue	Set to the maximum expected value to be measured: ■ Current: 10 nA to 1 A ■ Resistance: 2 Ω to 200 MΩ ■ Voltage: 0.20 V to 1000 V
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

You can assign any real number using this command. The instrument selects the closest fixed range that is large enough to measure the entered number. For example, for current measurements, if you expect a reading of approximately 9 mA, set the range to 9 mA to select the 10 mA range. When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using.

This command is primarily intended to eliminate the time that is required by the instrument to automatically search for a range.

When a range is fixed, any signal greater than the entered range generates an overrange condition. When an overrange condition occurs, the front panel displays "Overflow" and the remote interface returns `9.9e+37`.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range, regardless of measurement range setting. However, the setting for the measure range is retained, and when the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

If you change the range while the output is off, the instrument does not update the hardware settings, but if you read the range setting, the return is the setting that will be used when the output is turned on. If you set a range while the output is on, the new setting takes effect immediately.

NOTE

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the selected measurement function (if supported by that function).

The range for measure functions defaults to autorange for all measure functions. If you switch from a fixed range to autorange, autorange is set to off. The range remains at the fixed range until a measurement is made, at which time the range is set to accommodate the new measurement.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 0.5
```

Set the source function to current.
Set the measurement function to voltage.
Instrument selects the 2 V measurement range.

Also see

Ranges (on page 4-39)
smu.measure.autorange (on page 14-122)

smu.measure.read()

This function makes measurements, places them in a reading buffer, and returns the last reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = smu.measure.read()
reading = smu.measure.read(bufferName)
```

reading	The last reading of the measurement process
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; if no buffer is defined, it defaults to `defbuffer1`

Details

This function makes a measurement using the present function setting, stores the reading in a reading buffer, and returns the last reading.

The `smu.measure.count` attribute determines how many measurements are performed. You can also use the trigger model Simple Loop.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

NOTE

To make a power reading, use the `smu.measure.unit` command and set the units to `smu.UNIT_WATT` for the voltage or current measurement function.

Example

`voltMeasBuffer = buffer.make(10000)`
`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`print(smu.measure.read(voltMeasBuffer))`
	Create a buffer named `voltMeasBuffer`.
	Set the instrument to measure voltage.
	Make a measurement that is stored in the `voltMeasBuffer` and is also printed.

Also see

buffer.make() (on page 14-13)
Reading buffers (on page 6-1)
smu.measure.count (on page 14-135)
smu.measure.unit (on page 14-167)
trigger.model.load() — SimpleLoop (on page 14-248)

smu.measure.readwithtime()

This function initiates measurements and returns the last actual measurement and time information in UTC format without using the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading, seconds, fractional = smu.measure.readwithtime()
smu.measure.readwithtime(bufferName)
```

reading	The last reading of the measurement process
seconds	Seconds in UTC format
fractional	Fractional seconds
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command initiates measurements using the present function setting, stores the readings in a reading buffer, and returns the last reading.

The `smu.measure.count` attribute determines how many measurements are performed.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

Example

```
print(smu.measure.readwithtime(defbuffer1))
```

Print the last measurement and time information from `defbuffer1` in UTC format, which will look similar to:
`-1.405293589829e-11  1400904629  0.1950935`

Also see

smu.measure.count (on page 14-135)
trigger.model.load() — SimpleLoop (on page 14-248)

smu.measure.rel.acquire()

This function acquires a measurement and stores it as the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
relativeValue = smu.measure.rel.acquire()
```

relativeValue	The internal measurement acquired for the relative offset value

Details

This command triggers the instrument to make a new measurement for the selected function. This measurement is then stored as the new relative offset level.

When you send this command, the instrument does not apply any math, limit test, or filter settings to the measurement, even if they are set. It is a measurement that is made as if these settings are disabled.

If an error event occurs during the measurement, `nil` is returned and the relative offset level remains at the last valid setting.

You must change to the function for which you want to acquire a value before sending this command.

The instrument must have relative offset enabled to use the acquired relative offset value.

After executing this command, you can use the `smu.measure.rel.level` attribute to see the last relative level value that was acquired or that was set.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
rel_value = smu.measure.rel.acquire()
smu.measure.rel.enable = smu.ON
```

Acquires a relative offset level value for voltage measurements and turns the relative offset feature on.

Also see

smu.measure.rel.enable (on page 14-162)
smu.measure.rel.level (on page 14-163)

smu.measure.rel.enable

This attribute enables or disables the application of a relative offset value to the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
relEnable = smu.measure.rel.enable
smu.measure.rel.enable = relEnable
relEnable = smu.measure.getattribute(function, smu.ATTR_MEAS_REL_ENABLE)
smu.measure.setattribute(function, smu.ATTR_MEAS_REL_ENABLE, relEnable)
```

relEnable	Relative measurement control: ■ Disable relative offset: `smu.OFF` ■ Enable relative offset: `smu.ON`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value calculated when you acquire the relative offset value.

Each returned measured relative reading is the result of the following calculation:

Displayed reading = Actual measured reading - Relative offset value

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`rel_value = smu.measure.rel.acquire()`
`smu.measure.rel.enable = smu.ON`
	Acquires a relative offset level value for voltage measurements and turns the relative offset feature on.

Also see

Relative offset (on page 4-47)
smu.measure.rel.acquire() (on page 14-161)
smu.measure.rel.level (on page 14-163)

smu.measure.rel.level

This attribute contains the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
relValue = smu.measure.rel.level
smu.measure.rel.level = relValue
relValue = smu.measure.getattribute(function, smu.ATTR_MEAS_REL_LEVEL)
smu.measure.setattribute(function, smu.ATTR_MEAS_REL_LEVEL, relValue)
```

relValue	Relative offset value for measurements: ■ Current (A): –1.05 to 1.05 ■ Resistance (Ω): –2.10e6 to 2.10e6 ■ Voltage (V): –1100 to 1100
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command specifies the relative offset value that can be applied to new measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

You can set this value, or have the instrument acquire a value. If the instrument acquires the value, read this setting to return the value that was measured internally.

NOTE

If you have math, limits, or filter operations selected, you can set the relative offset value to include the adjustments made by these operations. To include these operations, set `smu.measure.rel.level` to `smu.measure.read()`. The adjustments from these operations are not used if you use the `smu.measure.rel.acquire()` function to set the relative offset level.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.rel.level = smu.measure.read()`
`smu.measure.rel.enable = smu.ON`
	Sets the measurement function to current, performs a current measurement, uses it as the relative offset value, and enables the relative offset for current measurements.

Also see

Relative offset (on page 4-47)
smu.measure.rel.acquire() (on page 14-161)
smu.measure.rel.enable (on page 14-162)

smu.measure.sense

This attribute selects local (2-wire) or remote (4-wire) sensing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.SENSE_2WIRE

Usage

```
sensing = smu.measure.sense
smu.measure.sense = sensing
sensing = smu.measure.getattribute(function, smu.ATTR_MEAS_SENSE)
smu.measure.setattribute(function, smu.ATTR_MEAS_SENSE, sensing)
```

sensing	Two-wire sensing: `smu.SENSE_2WIRE` Four-wire sensing: `smu.SENSE_4WIRE`
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

This command determines if 2-wire (local) or 4-wire (remote) sensing is used.

When you use 4-wire sensing, voltages are measured at the device under test (DUT). For the source voltage, if the sensed voltage is lower than the programmed amplitude, the voltage source increases the voltage until the sensed voltage is the same as the programmed amplitude. This compensates for IR drop in the output test leads.

Using 4-wire sensing with voltage measurements eliminates any voltage drops that may be in the test leads between the 2470 and the DUT.

When you are using 2-wire sensing, voltage is measured at the output connectors.

When you are measuring resistance, you can enable 4-wire sensing to make 4-wire resistance measurements.

When the output is off, 4-wire sensing is disabled and the instrument uses 2-wire sense, regardless of the sense setting. When the output is on, the selected sense setting is used.

If the output is on when you change the sense setting, the output is turned off.

Example

`smu.measure.func = smu.FUNC_RESISTANCE` `smu.measure.sense = smu.SENSE_4WIRE`	Set the measurement function to resistance. Set the sense to 4-wire remote.

Also see

Two-wire local sense connections (on page 4-8)
Four-wire remote sense connections (on page 4-10)

smu.measure.setattribute()

This function allows you to set up a measure function whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
smu.measure.setattribute(function, setting, value)
```

function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`
setting	The attribute for the function; refer to **Details** and the tables following the examples
value	The attribute value

Details

The lists following the Example and Also See information show the attributes that are available for each function, with links to the descriptions of the corresponding TSP command descriptions. The options for each attribute are the same as the settings for the TSP commands.

Example

```
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_REL_LEVEL, 0.55)
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_LIMIT_HIGH_1, 0.64)
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_LIMIT_LOW_1, 0.32)
smu.measure.configlist.create("MyMeasList")
smu.measure.configlist.storefunc("MyMeasList", smu.FUNC_DC_VOLTAGE)
```

Configure the DC Voltage function settings for the relative offset level, limit 1 high value, and limit 1 low value whether or not DC Voltage is the active function.
Create a configuration list named `MyMeasList`.
Store the settings for the DC Voltage function in `MyMeasList` at index 1.

Also see

smu.measure.getattribute() (on page 14-143)

Measure options

Autorange (on page 14-122): `smu.ATTR_MEAS_RANGE_AUTO`

Autorange high limit (on page 14-123): `smu.ATTR_MEAS_RANGE_HIGH`

Autorange low limit (on page 14-124): `smu.ATTR_MEAS_RANGE_LOW`

Autorange rebound (on page 14-125): `smu.ATTR_MEAS_RANGE_REBOUND`

Autozero enable (on page 14-126): `smu.ATTR_MEAS_AUTO_ZERO`

Count (on page 14-135): `smu.ATTR_MEAS_COUNT`

Display digits (on page 14-138): `smu.ATTR_MEAS_DIGITS`

NPLC (on page 14-156): `smu.ATTR_MEAS_NPLC`

Offset compensation (on page 14-157): `smu.ATTR_MEAS_OFFSET_COMP`

Range (on page 14-158): `smu.ATTR_MEAS_RANGE`

Relative offset enable (on page 14-162): `smu.ATTR_MEAS_REL_ENABLE`

Relative offset level (on page 14-163): `smu.ATTR_MEAS_REL_LEVEL`

Sense (2-wire or 4-wire) (on page 14-164): `smu.ATTR_MEAS_SENSE`

Unit (on page 14-167): `smu.ATTR_MEAS_UNIT`

User delays (on page 14-168): `smu.ATTR_MEAS_USER_DELAY_`*N*, where *N* is 1 to 5

Filter options

Filter count (on page 14-139): `smu.ATTR_MEAS_FILTER_COUNT`

Filter enable (on page 14-139): `smu.ATTR_MEAS_FILTER_ENABLE`

Filter type (on page 14-140): `smu.ATTR_MEAS_FILTER_TYPE`

Math options (measure)

Enable math (on page 14-151): `smu.ATTR_MEAS_MATH_ENABLE`

b (offset) value (on page 14-153): `smu.ATTR_MEAS_MATH_MXB_BF`

m (scalar) value (on page 14-154): `smu.ATTR_MEAS_MATH_MXB_MF`

Math format (on page 14-152): `smu.ATTR_MEAS_MATH_FORMAT`

Percent (on page 14-155): `smu.ATTR_MEAS_MATH_PERCENT`

Limit options (measure)

Limit 1 audible (on page 14-143): `smu.ATTR_MEAS_LIMIT_AUDIBLE_1`

Limit 1 auto clear (on page 14-144): `smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_1`

Limit 1 enable (on page 14-146): `smu.ATTR_MEAS_LIMIT_ENABLE_1`

Limit 1 fail (on page 14-147): `smu.ATTR_MEAS_LIMIT_FAIL_1`

Limit 1 high value (on page 14-149): `smu.ATTR_MEAS_LIMIT_HIGH_1`

Limit 1 low value (on page 14-150): `smu.ATTR_MEAS_LIMIT_LOW_1`

Limit 2 audible (on page 14-143): `smu.ATTR_MEAS_LIMIT_AUDIBLE_2`

Limit 2 auto clear (on page 14-144): `smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_2`

Limit 2 enable (on page 14-146): `smu.ATTR_MEAS_LIMIT_ENABLE_2`

Limit 2 fail (on page 14-147): `smu.ATTR_MEAS_LIMIT_FAIL_2`

Limit 2 high value (on page 14-149): `smu.ATTR_MEAS_LIMIT_HIGH_2`

Limit 2 low value (on page 14-150): `smu.ATTR_MEAS_LIMIT_LOW_2`

smu.measure.unit

This attribute sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: smu.UNIT_AMP Resistance: smu.UNIT_OHM Voltage: smu.UNIT_VOLT

Usage

```
unitOfMeasure = smu.measure.unit
smu.measure.unit = unitOfMeasure
unitOfMeasure = smu.measure.getattribute(function, smu.ATTR_MEAS_UNIT)
smu.measure.setattribute(function, smu.ATTR_MEAS_UNIT, unitOfMeasure)
```

unitOfMeasure	The units of measure to be displayed for the measurement: ■ Current: `smu.UNIT_AMP` (only available for current measurements) ■ Resistance: `smu.UNIT_OHM` (available for voltage, current, or resistance measurements) ■ Volts: `smu.UNIT_VOLT` (only available for voltage measurements) ■ Power: `smu.UNIT_WATT` (only available for voltage or current measurements)
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

The change in measurement units is displayed when the next measurement is made.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.unit = smu.UNIT_WATT`	Changes the front-panel display and buffer readings for voltage measurements to be displayed as power readings in watts.

Also see

smu.measure.func (on page 14-142)

smu.measure.userdelay[N]

This attribute sets a user-defined delay that you can use in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0 (0 s)

Usage

```
delayTime = smu.measure.userdelay[N]
smu.measure.userdelay[N] = delayTime
delayTime = smu.measure.getattribute(function, smu.ATTR_MEAS_USER_DELAY_N)
smu.measure.setattribute(function, smu.ATTR_MEAS_USER_DELAY_N, delayTime)
```

delayTime	The delay (0 for no delay, or 167 ns to 10 ks)
N	The user delay to which this time applies (1 to 5)
function	The measurement function; see **Functions**
function	The measurement function: ■ Voltage measurement: `smu.FUNC_DC_VOLTAGE` ■ Current measurement: `smu.FUNC_DC_CURRENT` ■ Ohms measurement: `smu.FUNC_RESISTANCE`

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.

The delay is specific to the selected function.

Example

```
smu.measure.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay for measure 1 to 5 s.
Set trigger block 1 to turn the source output on.
Set trigger block 2 to a dynamic delay that calls measure user delay 1.
Set trigger block 3 to make a measurement.
Set trigger block 4 to turn the source output off.
Set trigger block 5 to branch to block 1 ten times.
Start the trigger model.

Also see

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 14-267)

smu.reset()

This function turns off the output and resets the commands that begin with `smu.` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.reset()
```

Details

This function turns off the output and resets the commands that begin with `smu.` to their default settings. It also deletes source and measure configuration lists.

Example

`smu.reset()`	Turns off the output and resets the SMU commands to their default settings.

Also see

reset() (on page 14-114)

smu.source.autorange

This attribute determines if the range is selected manually or automatically for the selected source function or voltage source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
sourceAutorange = smu.source.autorange
smu.source.autorange = sourceAutorange
sourceAutorange = smu.source.getattribute(function, smu.ATTR_SRC_RANGE_AUTO)
smu.source.setattribute(function, smu.ATTR_SRC_RANGE_AUTO, sourceAutorange)
```

sourceAutorange	Disable automatic source range: `smu.OFF` Enable automatic source range: `smu.ON`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

This command indicates the state of the range for the selected source. When automatic source range is disabled, the source range is set manually.

When automatic source range is enabled, the instrument selects the range that is most appropriate for the value that is being sourced. The output level controls the range. If you read the range after the output level is set, the instrument returns the range that the instrument chose as appropriate for that source level.

If the source range is set to a specific value from the front panel or a remote command, the setting for automatic range is set to disabled.

Only available for the current and voltage functions.

Example

`smu.source.func = smu.FUNC_DC_CURRENT` `smu.source.autorange = smu.ON`	Set the source function to current. Set the instrument to select the source range automatically.

Also see

smu.source.range (on page 14-187)

smu.source.autodelay

This attribute enables or disables the automatic delay that occurs when the source is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
state = smu.source.autodelay
smu.source.autodelay = state
state = smu.source.getattribute(function, smu.ATTR_SRC_DELAY_AUTO)
smu.source.setattribute(function, smu.ATTR_SRC_DELAY_AUTO, state)
```

state	Disable the source auto delay: `smu.OFF` Enable the source auto delay: `smu.ON`
sourceAutorange	Disable automatic source range: `smu.OFF` Enable automatic source range: `smu.ON`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

When autodelay is turned on, the actual delay that is set depends on the range.

When source autodelay is on, if you set a source delay, the autodelay is turned off.

Example

`smu.source.autodelay = smu.OFF`	Turn off auto delay when current is being sourced.

Also see

smu.source.delay (on page 14-179)

smu.source.configlist.catalog()

This function returns the name of one source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.catalog()
```

Details

You can use this command to retrieve the names of source configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no source configuration lists have been created for the instrument.

Example

`print(smu.source.configlist.catalog())`	Request the name of one source configuration list that is stored in the instrument. Send the command again until it returns `nil` to get all stored lists.

Also see

Configuration lists (on page 4-82)
smu.source.configlist.create() (on page 14-172)

smu.source.configlist.create()

This function creates an empty source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list	Configuration script	

Usage

```
smu.source.configlist.create("listName")
```

listName	A string that represents the name of a source configuration list

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list through a power cycle, create a configuration script to save instrument settings, including any defined configuration lists.

Configuration list names must be unique. For example, the name of a source configuration list cannot be the same as the name of a measure configuration list.

Example

```
reset()

smu.source.configlist.create("MyScrList")

print(smu.source.configlist.catalog())

print(smu.source.configlist.catalog())



smu.source.configlist.store("MyScrList")
smu.source.configlist.store("MyScrList")
print(smu.source.configlist.size("MyScrList"))
```

Create a source configuration list named `MyScrLst`.

Print the name of one configuration list stored in volatile memory.

Output:
`MyScrList`

Print the name of one configuration list.

Output:
`nil`

Nil indicates that no more configuration lists are stored.

Store a configuration index in `MyScrList`.

Store a configuration index in `MyScrList`.

Print the number of configuration indexes in `MyScrList`.

Output:
2

Also see

Configuration lists (on page 4-82)
smu.source.configlist.store() (on page 14-177)

smu.source.configlist.delete()

This function deletes a source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.delete("listName")
smu.source.configlist.delete("listName", index)
```

listName	A string that represents the name of a source configuration list
index	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

`smu.source.configlist.delete("mySourceList")`	Deletes a configuration list named `mySourceList`.
`smu.source.configlist.delete("mySourceList", 14)`	Deletes delete configuration index 14 in the source configuration list named `mySourceList`

Also see

Configuration lists (on page 4-82)
smu.source.configlist.create() (on page 14-172)

smu.source.configlist.query()

This function returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.query("listName", index)
smu.source.configlist.query("listName", index, "fieldSeparator")
```

listName	A string that represents the name of a source configuration list
index	A number that defines a specific configuration index in the configuration list; the default is the first index in the configuration list
fieldSeparator	String that represents the separator for the data; use one of the following: ■ Comma (default): `,` ■ Semicolon: `;` ■ New line: `\n`

Details

This command can only return data for one configuration index. To get data for additional configuration indexes, resend the command and specify different configuration indexes.

Refer to Settings stored in a source configuration list (on page 4-83) for a complete list of source settings that the instrument stores in a source configuration list.

Example

```
print(smu.source.configlist.query("MyScrList", 2))
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Also see

Configuration lists (on page 4-82)
smu.source.configlist.create() (on page 14-172)
Settings stored in a source configuration index (on page 4-83)

smu.source.configlist.recall()

This function recalls a specific configuration index in a specific source configuration list and an optional measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.recall("listName", index)
smu.source.configlist.recall("listName", index, "measureListName")
smu.source.configlist.recall("listName", index, "measureListName", measureIndex)
```

listName	A string that represents the name of a source configuration list
index	A number that defines a specific configuration index in the source configuration list
measureListName	A string that represents the name of a measure configuration list
measureIndex	A number that defines a specific configuration index in the measure configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a specific source configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified source configuration list of that index.

You can optionally specify a measure configuration list and index to recall with the source settings. If you do not specify a measure index, the measure index defaults to match the source index. Specify a source and measure list together with this command to allow the instrument to coordinate the application of the settings in the two lists appropriately. If you do not need have the application of the source and measure configuration lists coordinated, you can specify just the source configuration list with this command and use the smu.measure.configlist.recall() (on page 14-131) command to recall measure settings separately in your application.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, event code 2790, "Configuration list, error, does not exist" is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. To see the settings that are recalled with an index, use the smu.source.configlist.query() (on page 14-174) command.

NOTE

To recall a source configuration list separately (not with this command), recall the source configuration list before the measure configuration list. This order ensures that dependencies between source and measure settings will be properly handled.

Example

`smu.source.configlist.recall("MySourceList")`	Because an index was not specified, this command recalls configuration index 1 from a configuration list named `MySourceList`.
`smu.source.configlist.recall("MySourceList", 5)`	Recalls configuration index 5 in a configuration list named `MySourceList`.
`smu.source.configlist.recall("MySourceList", 3, "MyMeasList")`	Recalls index 3 from a source configuration list named `MySourceList`, then recalls index 3 from a measure configuration list named `MyMeasureList` (because the measure list index was not specified, the instrument automatically recalled the same index that was called in the source list).
`smu.source.configlist.recall("MySourceList", 3, "MyMeasList", 5)`	Recalls index 3 from a source configuration list named `MySourceList`, then recalls index 5 from a measure configuration list named `MyMeasList`.

Also see

Configuration lists (on page 4-82)
smu.source.configlist.create() (on page 14-172)

smu.source.configlist.size()

This function returns the number of configuration indexes in a source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

indexCount = smu.source.configlist.size("*listName*")

indexCount	A number that represents the total quantity of indexes stored in the specified source configuration list
listName	A string that represents the name of a source configuration list

Details

The size of the list is equal to the number of configuration indexes in a configuration list.

Example

`print(smu.source.configlist.size("MyScrList"))`	Determine the number of configuration indexes in a source configuration list named `MyScrList`. Example output: 2

Also see

Configuration lists (on page 4-82)
smu.source.configlist.create() (on page 14-172)

smu.source.configlist.store()

This function stores the active source settings into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list		

Usage

```
smu.source.configlist.store("listName")
smu.source.configlist.store("listName", index)
```

listName	A string that represents the name of a source configuration list
index	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active source settings to a configuration index in a configuration list. If the index is defined, the configuration list is stored in that index. If the index is not defined, the configuration index is appended to the end of the list. If a configuration index already exists for the specified index, the new configuration overwrites the existing configuration index.

Refer to Settings stored in a source configuration index (on page 4-83) for information about the settings this command stores.

Example

`smu.source.configlist.store("MyConfigList")`	Store the active settings of the instrument to the source configuration list `MyConfigList`. Settings are saved at the end of the list since no index parameter is specified.
`smu.source.configlist.store("MyConfigList", 5)`	Store the active settings of the instrument to configuration index 5 on the source configuration list `MyConfigList`.

Also see

None

smu.source.configlist.storefunc()

This function allows you to store the settings for a source function into a source configuration list whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
smu.source.configlist.storefunc("ConfigListName", function)
smu.source.configlist.storefunc("ConfigListName", function, index)
```

ConfigListName	Name of the configuration list in which to store the function settings
function	The function to save into the configuration list: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`
index	The number of the configuration list index in which to store the settings

Details

The configuration list must be created before you send this function.

Example

```
smu.source.configlist.create("sourcelist")
smu.source.configlist.storefunc("sourcelist", smu.FUNC_DC_CURRENT)
```

Create a configuration list named `sourcelist`.
Store the settings for the DC current source function into the configuration list in index 1.

Also see

smu.measure.configlist.create() (on page 14-128)

smu.source.delay

This attribute contains the source delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```
sDelay = smu.source.delay
smu.source.delay = sDelay
sDelay = smu.source.getattribute(function, smu.ATTR_SRC_DELAY)
smu.source.setattribute(function, smu.ATTR_SRC_DELAY, sDelay)
```

sDelay	The length of the delay (0 to 10 ks)
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

This command sets a delay for the selected source function. This delay is in addition to normal settling times.

After the programmed source is turned on, this delay allows the source level to settle before a measurement is made.

If you set a specific source delay (`smu.source.delay`), source autodelay is turned off.

When source autodelay is turned on, the manual source delay setting is overwritten with the autodelay setting.

When either a source delay or autodelay is set, the delay is applied to the first source output and then only when the magnitude of the source changes.

Example

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.delay = 3
```

Set the function to voltage. Set a 3 s delay after the source is turned on before a measurement is made.

Also see

smu.source.autodelay (on page 14-171)
Source delay (on page 4-46)

smu.source.func

This attribute contains the source function, which can be voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list	Configuration script Source configuration list	smu.FUNC_DC_VOLTAGE

Usage

```
sFunction = smu.source.func
smu.source.func = sFunction
```

sFunction	The source function; set to one of the following values: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

When you set this command, it configures the instrument as either a voltage source or a current source.

When you read this command, it returns the output setting of the source.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
```

Sets the source function of the instrument to be a current source.

Also see

smu.source.level (on page 14-182)
smu.source.output (on page 14-185)

smu.source.getattribute()

This function returns the setting for a function attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = smu.source.getattribute(function, setting)
```

value	The attribute value
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`
setting	The setting of the attribute; refer to smu.source.setattribute() (on page 14-190) for the list of attributes

Details

You can retrieve one attribute at a time.

Example

```
print(smu.source.getattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_DELAY))
```

Retrieve the source delay setting for DC current.
Example return:
`1.05e-07`

Also see

smu.source.setattribute() (on page 14-190)

smu.source.highc

This attribute enables or disables high-capacitance mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFF

Usage

```
state = smu.source.highc
smu.source.highc = state
state = smu.source.getattribute(function, smu.ATTR_SRC_HIGHC)
smu.source.setattribute(function, smu.ATTR_SRC_HIGHC, state)
```

state	Turn high-capacitance mode off: `smu.OFF` Turn high-capacitance mode on: `smu.ON`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

When the instrument is measuring low current and is driving a capacitive load, you may see overshoot, ringing, and instability. You can enable the high capacitance mode to minimize these problems.

Example

`smu.source.highc = smu.ON`	Turn the high capacitance mode on.

Also see

High-capacitance operation (on page 5-23)

smu.source.level

This attribute immediately selects a fixed amplitude for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```
sourceLevel = smu.source.level
smu.source.level = sourceLevel
sourceLevel = smu.source.getattribute(function, smu.ATTR_SRC_LEVEL)
smu.source.setattribute(function, smu.ATTR_SRC_LEVEL, sourceLevel)
```

sourceLevel	Current: –1.05 A to 1.05 A Voltage: –1100 V to 1100 V
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

This command sets the output level of the voltage or current source. If the output is on, the new level is sourced immediately.

The sign of the source level dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

If a manual source range is selected, the level cannot exceed the specified range. For example, if the voltage source is on the 2 V range, you cannot set the voltage source level to 3 V. When autorange is selected, the amplitude can be set to any level supported by the instrument.

Example

`smu.source.func = smu.FUNC_DC_VOLTAGE` `smu.source.level = 1`	Set the instrument to source voltage and set it to source 1 V.

Also see

smu.source.func (on page 14-180)
smu.source.output (on page 14-185)
smu.source.protect.level (on page 14-186)

smu.source.offmode

This attribute defines the state of the source when the output is turned off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFFMODE_NORMAL

Usage

```
sourceOffMode = smu.source.offmode
smu.source.offmode = sourceOffMode
sourceOffMode = smu.source.getattribute(function, smu.ATTR_SRC_OFFMODE)
smu.source.setattribute(function, smu.ATTR_SRC_OFFMODE, sourceOffMode)
```

sourceOffMode	The output-off setting; set to one of the following values (see the **Details** below for specifics regarding each option): ■ `smu.OFFMODE_NORMAL` ■ `smu.OFFMODE_ZERO` ■ `smu.OFFMODE_HIGHZ` ■ `smu.OFFMODE_GUARD`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

When the 2470 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10% of the full scale of the present measurement function autorange value
- If source readback is off, Output Off is displayed in the home screen Source area
- If source readback is on, the actual measurement is displayed in the home screen Source area
- If measurement is set to resistance, dashes (--.----) are shown in the home screen Source area
- The Source button on the home screen shows the value that will be sourced when the output is turned on again

When the high-impedance output-off state is selected and the output is turned off:

- The measurement sense is set to 2-wire
- The output relay opens, disconnecting the instrument as a load

Opening the relay disconnects external circuitry from the inputs and outputs of the instrument. To prevent excessive wear on the output relay, do not use this output-off state for tests that turn the output off and on frequently.

The high-impedance output-off state should be used when the instrument is connected to a power source or another source-measure instrument. In some cases, it may also be appropriate for devices such as capacitors.

When the zero output-off state is selected and you turn off the output:

- The measurement sense is changed to 2-wire
- The voltage source is selected and set to 0 V
- The range is set to the presently selected range (turn off autorange)
- If the source is voltage, the current limit is not changed
- If the source is current, the current limit is set to the programmed source current value or to 10% full scale of the present current range, whichever is greater

When the zero output-off state is selected, you can use the instrument as an ammeter because it is outputting 0 V.

When the guard output-off state is selected and the output is turned off, the following actions occur:

- The measurement sense is changed to 2-wire
- The current source is selected and set to 0 A if the source is set to current (amps); otherwise, the output remains a voltage source when the output is turned off
- The voltage limit is set to 10% full scale of the present voltage range

Note that the front-panel display does not reflect all of the changes. For example, the 4-wire display indicator continues to display when the output is off, even though the sense is changed to 2-wire.

Example

`smu.source.offmode = smu.OFFMODE_HIGHZ`	Sets the output-off state so that the instrument opens the output relay when the output is turned off.

Also see

Output-off state (on page 4-16)
smu.source.output (on page 14-185)

smu.source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFF

Usage

```
sourceOutput = smu.source.output
smu.source.output = sourceOutput
```

sourceOutput	Switch the source output off: `smu.OFF` Switch the source output on: `smu.ON`

Details

When the output is switched on, the instrument sources either voltage or current, as set by `smu.source.func`.

Example

```
smu.source.output = smu.ON
```

Switch the source output of the instrument to on.

Also see

Turn the 2470 output on or off (on page 3-4)
smu.source.func (on page 14-180)
smu.source.offmode (on page 14-183)

smu.source.protect.level

This attribute sets the overvoltage protection setting of the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.PROTECT_NONE

Usage

```
smu.PROTECT_x = smu.source.protect.level
smu.source.protect.level = smu.PROTECT_x
smu.PROTECT_x = smu.source.getattribute(function, smu.ATTR_SRC_PROTECT_LEVEL)
smu.source.setattribute(function, smu.ATTR_SRC_PROTECT_LEVEL, smu.PROTECT_x)
```

x	The overvoltage protection level: `20V`, `40V`, `100V`, `200V`, `300V`, `400V`, `500V`, or `NONE`
sourceOffMode	The output-off setting; set to one of the following values (see the **Details** below for specifics regarding each option): ■ `smu.OFFMODE_NORMAL` ■ `smu.OFFMODE_ZERO` ■ `smu.OFFMODE_HIGHZ` ■ `smu.OFFMODE_GUARD`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced.

This protection is in effect for both positive and negative output voltages.

When this attribute is used in a test sequence, it should be set before turning the source on.

⚠ WARNING

Even with the overvoltage protection set to the lowest value (20 V), never touch anything connected to the terminals of the 2470 when the instrument is on. Always assume that a hazardous voltage (greater than 30 V_{RMS}) is present when the instrument is on. To prevent damage to the device under test or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

Example

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.protect.level = smu.PROTECT_40V
```

Sets the maximum voltage limit of the instrument to 40 V.

Also see

Overvoltage protection (on page 4-35)

smu.source.protect.tripped

This attribute indicates if the overvoltage source protection feature is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = smu.source.protect.tripped
value = smu.source.getattribute(function, smu.ATTR_SRC_PROTECT_TRIPPED)
```

value	Overvoltage protection not active: `smu.OFF` Overvoltage protection active: `smu.ON`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

When overvoltage protection is active, the instrument restricts the maximum voltage level that the instrument can source.

Example

```
print(smu.source.protect.tripped)
```

If overvoltage protection is active, the output is: `smu.ON`

Also see

Overvoltage protection (on page 4-35)
smu.source.protect.level (on page 14-186)

smu.source.range

This attribute selects the range for the source for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	Current: 1e-8 A Voltage: 0.2 V

Usage

```
rangeValue = smu.source.range
smu.source.range = rangeValue
rangeValue = smu.source.getattribute(function, smu.ATTR_SRC_RANGE)
smu.source.setattribute(function, smu.ATTR_SRC_RANGE, rangeValue)
```

rangeValue	Set to the maximum expected voltage or current to be sourced; see **Details** for values; the ranges are: ■ Current: –1.05 A to 1.05 A ■ Voltage: –1050 V to 1050 V

function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

This command manually selects the measurement range for the specified source.

To select the range, specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 3 V, set the source level to 3. The 2470 selects the 20 V range.

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 200 mV, 2 V, 20 V, 200 V, and 1000 V.

When you read this value, the instrument returns the positive full-scale value that the instrument is presently using.

This command is intended to eliminate the time required by the automatic range selection.

To select the range, you can specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 50 mV, send 0.05 (or 50e-3) to select the 200 mV range.

NOTE

If automatic range selection is set to on, when you select a specific range, automatic is set to off. To set the range to automatic selection, use the source autorange command.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.autorange = smu.OFF
smu.source.range = 1
```

Set the instrument to source current.
Turn autorange off.
Set the source range to 1 A.

Also see

Ranges (on page 4-39)
smu.source.autorange (on page 14-170)

smu.source.readback

This attribute determines if the instrument records the measured source value or the configured source value when making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
state = smu.source.readback
smu.source.readback = state
state = smu.source.getattribute(function, smu.ATTR_SRC_READBACK)
smu.source.setattribute(function, smu.ATTR_SRC_READBACK, state)
```

state	Disable readback: `smu.OFF` Enable readback: `smu.ON`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

When source readback is off, the instrument records and displays the source value you set. When you use the actual source value (source readback on), the instrument measures the actual source value immediately before making the measurement of the device under test.

Using source readback results in more accurate measurements, but also a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the device-under-test measurement. When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the device-under-test measurement.

Example

`reset()`	Reset the instrument to default settings.
`testDataBuffer = buffer.make(100)`	Make a buffer named `testDataBuffer` that can hold 100 readings.
`smu.source.func = smu.FUNC_DC_VOLTAGE`	Set source function to voltage.
`smu.measure.func = smu.FUNC_DC_CURRENT`	Set the measurement function to current.
`smu.source.readback = smu.ON`	Set read back on.
`smu.source.level = 10`	Set the instrument to take 100 readings.
`smu.measure.count = 100`	Turn the output on.
`smu.source.output = smu.ON`	Take the measurements.
`smu.measure.read(testDataBuffer)`	Turn the output off.
`smu.source.output = smu.OFF`	Get the source values and measurements from the buffer.
`printbuffer(1, 100, testDataBuffer.sourcevalues, testDataBuffer)`	

Also see

smu.measure.func (on page 14-142)
smu.source.func (on page 14-180)

smu.source.setattribute()

This function allows you to set up a source function whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Source configuration list	Not applicable

Usage

```
smu.source.setattribute(function, setting, value)
```

function	The source function; set to one of the following values: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`
setting	The parameter to be set; see **Details**
value	The function or setting value

Details

You can set the following parameters. The links in the list below link to the descriptions of the corresponding TSP command descriptions. The settings available for each parameter are the same as the settings for the TSP commands.

The parameters you can set are:

- Autorange (on page 14-170): `smu.ATTR_SRC_RANGE_AUTO`
- Autodelay (on page 14-171): `smu.ATTR_SRC_DELAY_AUTO`
- Delay (on page 14-179): `smu.ATTR_SRC_DELAY`
- Function (on page 14-180): `smu.ATTR_SRC_FUNCTION`
- High capacitance (on page 14-181): `smu.ATTR_SRC_HIGHC`
- Level (on page 14-182): `smu.ATTR_SRC_LEVEL`
- Limit level (on page 14-201): `smu.ATTR_SRC_LIMIT_LEVEL`
- Output Off (on page 14-183): `smu.ATTR_SRC_OFFMODE`
- Overvoltage protection limit (on page 14-186): `smu.ATTR_SRC_PROTECT_LEVEL`
- Overvoltage protection limit active (on page 14-187): `smu.ATTR_SRC_PROTECT_TRIPPED`
- Range (on page 14-187): `smu.ATTR_SRC_RANGE`
- Readback (on page 14-189): `smu.ATTR_SRC_READBACK`
- Source limit exceeded (on page 14-202): `smu.ATTR_SRC_LIMIT_TRIPPED`
- User delay *N* (on page 14-200): `smu.ATTR_SRC_USER_DELAY_`*N*, where *N* is the user delay, 1 to 5

Example

```
smu.source.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_RANGE, 0.007)
smu.source.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_DELAY, 0.35)
smu.source.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_LEVEL, 0.035)
```

Sets the range, delay, and level settings for DC current, whether or not DC current is the active function.

Also see

smu.source.configlist.storefunc() (on page 14-178)
smu.source.getattribute() (on page 14-180)

smu.source.sweeplinear()

This function sets up a linear sweep for a fixed number of measurement points.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplinear("configListName", start, stop, points)
smu.source.sweeplinear("configListName", start, stop, points, sDelay)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
   rangeType)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
   rangeType, failAbort)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
   rangeType, failAbort, dual)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
   rangeType, failAbort, dual, bufferName)
```

configListName	A string that contains the name of the configuration list that the instrument will create for this sweep
start	The voltage or current source level at which the sweep starts: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
stop	The voltage or current at which the sweep stops: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
points	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
sDelay	The delay between measurement points; default is `smu.DELAY_AUTO`, which enables autodelay, or a specific delay value from 50 µs to 10 ks, or 0 for no delay
count	The number of times to run the sweep; default is 1: ■ Infinite loop: `smu.INFINITE` ■ Finite loop: 1 to 268,435,455

rangeType	The source range that is used for the sweep: ■ Most sensitive source range for each source level in the sweep: `smu.RANGE_AUTO` ■ Best fixed range: `smu.RANGE_BEST` (default) ■ Present source range for the entire sweep: `smu.RANGE_FIXED`
failAbort	Complete the sweep if the source limit is exceeded: `smu.OFF` Abort the sweep if the source limit is exceeded: `smu.ON` (default)
dual	Determines if the sweep runs from start to stop and then from stop to start: ■ Sweep from start to stop only: `smu.OFF` (default) ■ Sweep from start to stop, then stop to start: `smu.ON`
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it clears any existing trigger models, creates a source configuration list, and populates the trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the sweep delay, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```
reset()
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.source.sweeplinear("VoltLinSweep", 0, 10, 20, 1e-3, 1, smu.RANGE_FIXED)
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 100e-6
trigger.model.initiate()
```

Reset the instrument to its defaults.
Set the source function to voltage.
Set the source range to 20 V.
Set up a linear sweep that sweeps from 0 to 10 V in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep `VoltLinSweep`.
Set the measure function to current.
Set the current range to 100 µA.
Start the sweep.

Also see

Sweep operation (on page 4-54)
trigger.model.initiate() (on page 14-241)

smu.source.sweeplinearstep()

This function sets up a linear source sweep configuration list and trigger model with a fixed number of steps.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplinearstep("configListName", start, stop, step)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
   rangeType)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
   rangeType, failAbort)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
   rangeType, failAbort, dual)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
   rangeType, failAbort, dual, bufferName)
```

configListName	A string that contains the name of the configuration list that the instrument will create for this sweep
start	The voltage or current source level at which the sweep starts: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V
stop	The voltage or current at which the sweep stops: ■ Current: -1.05 A to 1.05 A ■ Voltage: -1100 V to 1100 V

step	The step size at which the source level will change; must be more than 0
sDelay	The delay between measurement points; default is `smu.DELAY_AUTO`, which enables autodelay, a specific delay value from 50 µs to 10 ks, or 0 for no delay
count	The number of times to run the sweep; default is `1`: ▪ Infinite loop: `smu.INFINITE` ▪ Finite loop: `1` to `268,435,455`
rangeType	The source range that is used for the sweep: ▪ Most sensitive source range for each source level in the sweep: `smu.RANGE_AUTO` ▪ Best fixed range: `smu.RANGE_BEST` (default) ▪ Present source range for the entire sweep: `smu.RANGE_FIXED`
failAbort	Complete the sweep if the source limit is exceeded: `smu.OFF` Abort the sweep if the source limit is exceeded: `smu.ON` (default)
dual	Determines if the sweep runs from start to stop and then from stop to start: ▪ Sweep from start to stop only: `smu.OFF` (default) ▪ Sweep from start to stop, then stop to start: `smu.ON`
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

When the sweep is started, the instrument sources a specific voltage or current voltage to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it deletes the existing trigger model and creates a trigger model with a uniform series of ascending or descending voltage or current changes, called steps. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the stop level, which is calculated from the number of steps. A measurement is performed at each source step (including the start and stop levels). At this level, the instrument performs another measurement and then stops the sweep.

The instrument uses the step size parameter to determine the number of source level changes. The source level changes in equal steps from the start level to the stop level. To avoid a setting conflicts error, make sure the step size is greater than the start value and less than the stop value. To calculate the number of source-measure points in a sweep, use the following formula:

Points = [(Stop - Start) / Step] + 1

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```
reset()
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 1
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
smu.source.sweeplinearstep("CurrLogSweep", -1.05, 1.05, .25, 10e-3, 1,
    smu.RANGE_FIXED)
trigger.model.initiate()
```

Reset the instrument to its defaults.
Set the source function to current.
Set the source range to 1 A. Set the measure function to voltage with a range of 20 V.
Set up a linear step sweep that sweeps from -1.05 A to 1.05 A in 0.25 A increments with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep `CurrLogSweep`.
Start the sweep.

Also see

Sweep operation (on page 4-54)

smu.source.sweeplist()

This function sets up a sweep based on a configuration list, which allows you to customize the sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplist("configListName")
smu.source.sweeplist("configListName", index)
smu.source.sweeplist("configListName", index, sDelay)
smu.source.sweeplist("configListName", index, sDelay, count)
smu.source.sweeplist("configListName", index, sDelay, count, failAbort)
smu.source.sweeplist("configListName", index, sDelay, count, failAbort, bufferName)
```

configListName	The name of the source configuration list that the sweep uses; this must be defined before sending this command
index	The index in the configuration list where the sweep starts; default is `1`
sDelay	The delay between measurement points; default is `0` for no delay or you can set a specific delay value from 50 µs to 10 ks
count	The number of times to run the sweep; default is `1`: ■ Infinite loop: `smu.INFINITE` ■ Finite loop: `1` to `268,435,455`
failAbort	Complete the sweep if the source limit is exceeded: `smu.OFF` Abort the sweep if the source limit is exceeded: `smu.ON` (default)
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command allows you to set up a custom sweep, using a configuration list to specify the source levels.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

To run the sweep, initiate the trigger model.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```
reset()
smu.source.configlist.create("CurrListSweep")
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 100e-3
smu.source.level = 1e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 10e-6
smu.source.configlist.store("CurrListSweep")
smu.source.level = 7e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 11e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 9e-3
smu.source.configlist.store("CurrListSweep")
smu.source.sweeplist("CurrListSweep", 1, 0.001)
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
trigger.model.initiate()
```

Reset the instrument to its defaults
Create a source configuration list called `CurrListSweep`.
Set the source function to current.
Set the source current range to 100 mA.
Set the source current level to 1 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 10 μA.
Save the source settings to `CurrListSweep`.
Set the source current level to 7 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 11 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 9 mA.
Save the source settings to `CurrListSweep`.
Set up a list sweep that uses the entries from the `CurrListSweep` configuration list and starts at index 1 of the list.
Set a source delay of 1 ms.
Start the sweep.

Also see

Configuration lists (on page 4-82)
Sweep operation (on page 4-54)
trigger.model.initiate() (on page 14-241)

smu.source.sweeplog()

This function sets up a logarithmic sweep for a set number of measurement points.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplog("configListName", start, stop, points)
smu.source.sweeplog("configListName", start, stop, points, sDelay)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort, dual)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort, dual, bufferName)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort, dual, bufferName, asymptote)
```

configListName	A string that contains the name of the configuration list that the instrument will create for this sweep
start	The voltage or current source level at which the sweep starts: ■ Current: 1 pA to 1.05 A ■ Voltage: 1 pV to 1100 V
stop	The voltage or current at which the sweep stops: ■ Current: 1 pA to 1.05 A ■ Voltage: 1 pV to 1100 V
points	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
sDelay	The delay between measurement points; default is `smu.DELAY_AUTO`, which enables autodelay, or a specific delay value from 50 µs to 10 ks, or 0 for no delay
count	The number of times to run the sweep; default is `1`: ■ Infinite loop: `smu.INFINITE` ■ Finite loop: `1` to `268,435,455`
rangeType	The source range that is used for the sweep: ■ Most sensitive source range for each source level in the sweep: `smu.RANGE_AUTO` ■ Best fixed range: `smu.RANGE_BEST` (default) ■ Present source range for the entire sweep: `smu.RANGE_FIXED`
failAbort	Complete the sweep if the source limit is exceeded: `smu.OFF` Abort the sweep if the source limit is exceeded: `smu.ON` (default)
dual	Determines if the sweep runs from start to stop and then from stop to start: ■ Sweep from start to stop only: `smu.OFF` (default) ■ Sweep from start to stop, then stop to start: `smu.ON`

bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
asymptote	Default is 0; see **Details**

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it clears the existing trigger model and creates a new trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

The asymptote changes the inflection of the sweep curve and allows it to sweep through zero. You can use the asymptote parameter to customize the inflection and offset of the source value curve. Setting this parameter to zero provides a conventional logarithmic sweep. The asymptote value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep. The asymptote value must not be equal to or between the starting and ending values. It must be outside the range defined by the starting and ending values.

Example

```
reset()
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 100e-6
smu.source.sweeplog("VoltLogSweep", 1, 10, 20, 1e-3, 1, smu.RANGE_FIXED)
trigger.model.initiate()
```

Reset the instrument to its defaults.
Set the source function to voltage.
Set the source range to 20 V.
Set the measure function to current.
Set the current range to 100 μA.
Set up a log sweep that sweeps from 1 to 10 V in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep `VoltLogSweep`.
Start the sweep.

Also see

Sweep operation (on page 4-54)
trigger.model.initiate() (on page 14-241)

smu.source.userdelay[N]

This attribute sets a user-defined delay that you can use in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```
delayTime = smu.source.userdelay[N]
smu.source.userdelay[N] = delayTime
delayTime = smu.source.getattribute(function, smu.ATTR_SRC_USER_DELAY_N)
smu.source.setattribute(function, smu.ATTR_SRC_USER_DELAY_N, delayTime)
```

delayTime	The delay in seconds (0 to 10 ks)
N	The number that identifies this user delay (1 to 5)
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.

The delay is specific to the selected function.

Example

```
smu.source.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_S1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay for source 1 to 5 s.
Set trigger block 1 to turn the source output on.
Set trigger block 2 to a dynamic delay that calls source user delay 1.
Set trigger block 3 to make a measurement.
Set trigger block 4 to turn the source output off.
Set trigger block 5 to branch to block 1 ten times.
Start the trigger model.

Also see

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 14-267)

smu.source.xlimit.level

This attribute selects the source limit for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	Voltage: 105 µA Current: 21 V

Usage

```
value = smu.source.xlimit.level
smu.source.xlimit.level = value
value = smu.source.getattribute(function, smu.ATTR_SRC_LIMIT_LEVEL)
smu.source.setattribute(function, smu.ATTR_SRC_LIMIT_LEVEL, value)
```

value	The limit: ■ Current source function: -1.05 A to 1.05 A ■ Voltage source function: -1100 V to 1100 V
x	The function for which to set the limit: ■ Voltage: `v` ■ Current: `i`
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

This command sets the source limit for measurements. The values that can be set for this command are limited by the setting for the overvoltage protection limit.

The 2470 cannot source levels that exceed this limit.

If you change the measure range to a range that is not appropriate for this limit, the instrument changes the source limit to a limit that is appropriate to the range and a warning is generated. Depending on the source range, your actual maximum limit value could be lower. The instrument makes adjustments to stay in the operating boundaries.

This value can also be limited by the measurement range. If a specific measurement range is set, the limit must be 10.6% or higher of the measurement range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

Limits are absolute values.

You can use `smu.source.xlimit.tripped` to check the limit state of the source output.

Example

`smu.source.func = smu.FUNC_DC_VOLTAGE` `smu.source.ilimit.level = 1`	Set the source function to voltage with the current limit set to 1 A.

Also see

smu.source.protect.level (on page 14-186)
smu.source.xlimit.tripped (on page 14-202)

smu.source.xlimit.tripped

This attribute indicates if the source exceeded the limits that were set for the selected measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Function change	Not saved	Not applicable

Usage

```
state = smu.source.xlimit.tripped
state = smu.source.getattribute(function, smu.ATTR_SRC_LIMIT_TRIPPED)
```

state	Indicates if the limit has been tripped: ■ Not tripped = `0` ■ Tripped = `1`
x	The function whose limit was tripped: ■ `v`: voltage ■ `i`: current
function	The source function: ■ Current source: `smu.FUNC_DC_CURRENT` ■ Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

You can use this command to check the limit state of the source.

If the limits were exceeded, the instrument clamps the source to keep the source within the set limits.

If you check the limit for the source that is not presently selected, `nil` is returned.

Example

`print(smu.source.vlimit.tripped)`	Check the state of the source limit for voltage. If the limit was exceeded, the output is: `smu.ON`

Also see

smu.source.xlimit.level (on page 14-201)

smu.terminals

This attribute describes which set of input and output terminals the instrument is using.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.TERMINALS_FRONT

Usage

```
terminals = smu.terminals
smu.terminals = terminals
```

terminals	Use the front-panel input and output terminals: `smu.TERMINALS_FRONT` Use the rear-panel input and output terminals: `smu.TERMINALS_REAR`

Details

This command selects which set of input and output terminals the instrument uses. You can select front panel or rear panel terminals.

If the output is on when you change from one set of terminals to the other, the output is turned off.

This command replaces the `smu.measure.terminals` command, which is deprecated.

Example

```
smu.terminals = smu.TERMINALS_FRONT
```

Use the front-panel terminals for measurements.

Also see

None

status.clear()

This function clears event registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.clear()
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It does not affect the Questionable Event Enable or Operation Event Enable registers.

Example

```
status.clear()
```

Clear the bits in the registers

Also see

*CLS (on page 15-2)

status.condition

This attribute stores the status byte condition register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
statusByte = status.condition
```

statusByte	The status byte

Details

You can use this command to read the status byte, which is returned as a numeric value.

When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence. The returned value can indicate that one or more status events occurred. If more than one bit of the register is set, *statusByte* equals the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128). See Understanding bit settings (on page 16-15) for additional information about reading bit values.

NOTE

If you are using the GPIB, USB, or VXI-11 serial poll sequence of the 2470 to get the status byte (also called a serial poll byte), B6 is the Request for Service (RQS) bit. If the bit is set, it indicates that a serial poll (SRQ) has occurred. For additional detail, see Serial polling and SRQ (on page 16-13).

The meanings of the individual bits of this register are shown in the following table.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.MSB	An enabled measurement event
1	2	Not used	
2	4	status.EAV	An error or status message is present in the Error Queue
3	8	status.QSB	An enabled questionable event
4	16	status.MAV	A response message is present in the Output Queue
5	32	status.ESB	An enabled standard event
6	64	status.MSS	An enabled summary bit of the status byte register is set
7	128	status.OSB	An enabled operation event

Example

```
statusByte = status.condition
print(statusByte)
```

Returns `statusByte`.
Example output:
`1.29000e+02`
Converting this output (129) to its binary equivalent yields 1000 0001
Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSS) and B7 (OSB).

Also see

None

status.operation.condition

This attribute reads the Operation Event Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
operationRegister = status.operation.condition
```

operationRegister	The status of the operation status register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page 16-15).

Example

```
print(status.operation.condition)
```

Returns the contents of the register.

Also see

Operation Event Register (on page 16-7)

status.operation.enable

This attribute sets or reads the contents of the Operation Event Enable Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
operationRegister = status.operation.enable
status.operation.enable = operationRegister
```

operationRegister	The status of the operation status register

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

Example

```
-- decimal 20480 = binary 0101 0000 0000 0000
operationRegister = 20480
status.operation.enable = operationRegister
```

Sets the 12 and 14 bits of the operation status enable register using a decimal value.

Also see

Operation Event Register (on page 16-7)
Understanding bit settings (on page 16-15)

status.operation.event

This attribute reads the Operation Event Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
operationRegister = status.operation.event
```

operationRegister	The status of the operation status register

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example

```
status.request_enable = status.OSB
status.operation.setmap(0, 4918, 4917)
status.operation.enable = 1
defbuffer1.clear()
defbuffer1.fillmode = buffer.FILL_ONCE
defbuffer1.capacity = 10
smu.measure.count = 10
smu.measure.read()
print(status.operation.event)
```

Set bits in the Status Request Enable Register to record an enabled event in the Operation Status Register.
Map event number 4918 (a default buffer is full) to set bit 0 in the Operation Event Register and event number 4917 (a default buffer is empty) to clear bit 0.
Clear `defbuffer1`.
Set `defbuffer1` to fill once.
Resizes `defbuffer1` to 10 readings.
Sets the measure count to 10 readings and makes a measurement.
Reads the operation event register.
Output:
1

Also see

Operation Event Register (on page 16-7)

status.operation.getmap()

This function requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setEvent, clearEvent = status.operation.getmap(bitNumber)
```

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`print(status.operation.getmap(0))`	Query bit 0 of the Operation Event Register. Example output: `4918 4917`

Also see

Operation Event Register (on page 16-7)
Programmable status register sets (on page 16-4)
status.operation.setmap() (on page 14-207)

status.operation.setmap()

This function allows you to map events to bits in the Operation Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.operation.setmap(bitNumber, setEvent)
status.operation.setmap(bitNumber, setEvent, clearEvent)
```

bitNumber	The bit number that is mapped to an event (0 to 14)
setEvent	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
clearEvent	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to `0` when the event is detected.

If the event is set to zero (`0`), the bit is never set.

See Event numbers (on page 16-9) for information about event numbers.

Example

```
status.operation.setmap(0, 2731, 2732)
```

When event 2731 (trigger model initiated) occurs, bit 0 in the condition and event registers of the Operation Event Register are set. When event 2732 (trigger model idled) occurs, bit 0 in the condition register is cleared.

Also see

Event numbers (on page 16-9)
Operation Event Register (on page 16-7)
Programmable status register sets (on page 16-4)
status.operation.getmap() (on page 14-207)

status.preset()

This function resets all bits in the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.preset()
```

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the Service Request Enable Register (`*SRE`) to Standard Request Enable Register (`*ESE`).

Preset does not affect the event queue.

The Standard Event Status Register is not affected by this command.

Example

```
status.preset()
```

Resets the instrument status model.

Also see

Status model (on page 16-1)

status.questionable.condition

This attribute reads the Questionable Condition Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
questionableRegister = status.questionable.condition
```

questionableRegister	The value of the register (0 to 65535)

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page 16-15).

Example

```
print(status.questionable.condition)
```

Reads the Questionable Condition Register.

Also see

Questionable Event Register (on page 16-6)
Understanding bit settings (on page 16-15)

status.questionable.enable

This attribute sets or reads the contents of the questionable event enable register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
questionableRegister = status.questionable.enable
status.questionable.enable = questionableRegister
```

questionableRegister	The value of the register (0 to 65535)

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register are set.

For detail on interpreting the value of a register, see Understanding bit settings (on page 16-15).

Example

```
status.questionable.enable = 17
print(status.questionable.enable)
```

Set bits 0 and 4 of the Questionable Event Enable Register.
Returns 17, which indicates the register was set correctly.

Also see

Questionable Event Register (on page 16-6)

status.questionable.event

This attribute reads the Questionable Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
questionableRegister = status.questionable.event
```

questionableRegister	The value of the questionable status register (0 to 65535)

Details

This query reads the contents of the questionable status event register. After sending this command and addressing the instrument to talk, a value is sent to the computer. This value indicates which bits in the appropriate register are set.

The Questionable Register can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set the Questionable Register to the sum of their decimal weights. For example, to set bits B12 and B13, set the Questionable Register to 12,288 (which is the sum of 4,096 + 8,192).

Example 1

```
-- decimal 66 = binary 0100 0010
questionableRegister = 66
status.questionable.enable = questionableRegister
```

Uses a decimal value to set bits B1 and B6 of the status questionable enable register.

Example 2

```
-- decimal 2560 = binary 00001010 0000 0000
questionableRegister = 2560
status.questionable.enable = questionableRegister
```

Uses a decimal value to set bits B9 and B11 of the status questionable enable register.

Also see

Questionable Event Register (on page 16-6)

status.questionable.getmap()

This function requests the mapped set event and mapped clear event status for a bit in the Questionable Event Registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setEvent, clearEvent = status.questionable.getmap(bitNumber)
```

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check (0 to 14)

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`print(status.questionable.getmap(9))`	Returns the events that were mapped to set and clear bit 9.

Also see

Questionable Event Register (on page 16-6)
status.questionable.setmap() (on page 14-211)

status.questionable.setmap()

This function maps events to bits in the questionable event registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.questionable.setmap(bitNumber, setEvent)
status.questionable.setmap(bitNumber, setEvent, clearEvent)
```

bitNumber	The bit number that is mapped to an event (0 to 14)
setEvent	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
clearEvent	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to `0` when the event is detected.

If the event is set to zero (`0`), the bit is never set.

See Event numbers (on page 16-9) for information about event numbers.

Example

```
status.questionable.setmap(0, 4917, 4918)
```

When event 4917 (a default buffer is 0% filled) occurs, bit 0 is set in the condition register and the event register of the Questionable Event Register. When event 4918 (a default buffer is 100% filled) occurs, bit 0 in the condition register is cleared.

Also see

Event numbers (on page 16-9)
status.questionable.getmap() (on page 14-211)

status.request_enable

This attribute stores the settings of the Service Request (SRQ) Enable Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
SRQEnableRegister = status.request_enable
status.request_enable = SRQEnableRegister
```

SRQEnableRegister	The status of the service request (SRQ) enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings (0 to 255)

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	`status.MSB`	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	`status.EAV`	An error or status message is present in the Error Queue.
3	8	`status.QSB`	An enabled event in the Questionable Status Register has occurred.
4	16	`status.MAV`	A response message is present in the Output Queue.
5	32	`status.ESB`	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	`status.OSB`	An enabled event in the Operation Status Register has occurred.

Example 1

```
requestSRQEnableRegister = status.MSB + status.OSB
status.request_enable = requestSRQEnableRegister
```

Uses constants to set the MSB and OSB bits of the service request (SRQ) enable register and clear all other bits.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the MSB and OSB bits and clear all other bits of the service request (SRQ) enable register.

Example 3

```
status.request_enable = 0
```

Clear the register.

Also see

Status model (on page 16-1)
Understanding bit settings (on page 16-15)

status.standard.enable

This attribute reads or sets the bits in the Status Enable register of the Standard Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
standardRegister = status.standard.enable
status.standard.enable = standardRegister
```

standardRegister	The value of the Status Enable register of the Standard Event Register (0 to 255)

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or value of the bit as the *standardRegister* parameter.

You can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B2, set *standardRegister* to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-108) function.
1	2	Not used	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example 1

```
standardRegister = status.standard.OPC + status.standard.QYE
status.standard.enable = standardRegister
```

Uses constants to set the OPC and QYE bits of the standard event status enable register.

Example 2

```
-- decimal 5 = binary 0000 0101
standardRegister = 5
status.standard.enable = standardRegister
```

Uses a decimal value to set the OPC and QYE bits of the standard event status enable register.

Also see

Standard Event Register (on page 16-3)
Understanding bit settings (on page 16-15)

status.standard.event

This attribute returns the contents of the Standard Event Status Register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	status.preset()	Not applicable	0

Usage

```
standardRegister = status.standard.event
```

standardRegister	The status of the standard event status register

Details

When this command returns zero (0), no bits are set. You can send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-108) function.
1	2	Not used	Not used.
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

```
print(status.standard.event)
```

May return the value 129, showing that the Standard Event Status Register contains binary 10000001

Also see

Standard Event Register (on page 16-3)
Understanding bit settings (on page 16-15)

timer.cleartime()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
timer.cleartime()
```

Example

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.
Output:

```
Delay time was 0.500099
35
```

Also see

timer.gettime() (on page 14-216)

timer.gettime()

This function measures the elapsed time since the timer was last cleared.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
time = timer.gettime()
```

time	The elapsed time in seconds (1 µs resolution)

Example

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.
Output:

```
Delay time was 0.500099
35
```

Also see

timer.cleartime() (on page 14-216)

trigger.blender[N].clear()

This function clears the blender event detector and resets the overrun indicator of blender *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].clear()
```

N	The blender number (up to two)

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

```
trigger.blender[2].clear()
```

Clears the event detector for blender 2.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender performs OR operations or AND operations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	false (AND)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The type of operation: ■ `true`: OR operation ■ `false`: AND operation
N	The blender number (up to two)

Details

This command selects whether the blender waits for any one event (OR) or waits for all selected events (AND) before signaling an output event.

Example

```
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 14-219)

trigger.blender[N].overrun

This attribute indicates whether or not an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Trigger blender *N* clear Trigger blender *N* reset	Not applicable	Not applicable

Usage

overrun = trigger.blender[*N*].overrun

overrun	Trigger blender overrun state (`true` or `false`)
N	The blender number (up to two)

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

```
print(trigger.blender[1].overrun)
```

If an event was ignored, the output is `true`.
If an event was not ignored, the output is `false`.

Also see

trigger.blender[N].reset() (on page 14-219)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].reset()
```

N	The trigger event blender (up to two)

Details

The `trigger.blender[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.blender[N].orenable`
- `trigger.blender[N].stimulus[M]`

It also clears `trigger.blender[N].overrun`.

Example

```
trigger.blender[1].reset()
```

Resets the trigger blender 1 settings to factory defaults.

Also see

trigger.blender[N].orenable (on page 14-217)
trigger.blender[N].overrun (on page 14-218)
trigger.blender[N].stimulus[M] (on page 14-219)

trigger.blender[N].stimulus[M]

This attribute specifies the events that trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = event
```

event	The event that triggers the blender action; see **Details**
N	An integer that represents the trigger event blender (up to two)
M	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event.

Use none to disable the blender input.

The *event* parameter may be any of the trigger events shown in the following table.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMER`*N*
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINK`*N*

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 14-219)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.blender[N].wait(timeout)
```

triggered	Trigger detection indication for blender
N	The trigger blender (up to two) on which to wait
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
print(trigger.blender[1].wait(3))
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.
Wait 3 s while checking if trigger blender 1 event has occurred.

Also see

trigger.blender[N].clear() (on page 14-217)

trigger.clear()

This function clears any pending command triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

A command trigger indicates if a trigger event has been detected over a command interface since the last `trigger.wait()` command was sent. Command triggers are generated by:

- Sending `*TRG` over a remote interface
- GPIB `GET` bus commands
- USBTMC trigger messages
- VXI-11 device trigger commands

`trigger.clear()` clears the command triggers and discards the history of trigger events.

Example

`*TRG`
`print(trigger.wait(1))`
`trigger.clear()`
`print(trigger.wait(1))`
	Generate a trigger event.
	Check if there are any pending trigger events.
	Output: `true`
	Clear any pending command triggers.
	Check if there are any pending trigger events.
	Output: `false`

Also see

trigger.wait() (on page 14-295)

trigger.continuous

This attribute determines the trigger mode setting after bootup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Never	Nonvolatile memory	trigger.CONT_AUTO

Usage

```
setting = trigger.continuous
trigger.continuous = setting
```

setting	Do not start continuous measurements after boot: `trigger.CONT_OFF` Start continuous measurements after boot: `trigger.CONT_AUTO` Place the instrument into local control and start continuous measurements after boot: `trigger.CONT_RESTART`

Details

Conditions must be valid before continuous measurements can start.

When the restart parameter is selected, the instrument is placed in local mode, aborts any running scripts, and aborts any trigger models that are running. If the command is in a script, it is the last command that runs before the script is aborted. The restart parameter is not stored in nonvolatile memory, so it does not affect start up behavior.

The off and automatic parameters are stored in nonvolatile memory, so they affect start up behavior.

Example

```
trigger.continuous = trigger.CONT_OFF
```
When the instrument starts up, the measurement method is set to idle.

Also see

None

trigger.digin[N].clear()

This function clears the trigger event on a digital input line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digin[N].clear()
```

N	Digital I/O trigger line (1 to 6)

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to `false`).

Example

```
trigger.digin[2].clear()
```
Clears the trigger event detector on I/O line 2.

Also see

digio.line[N].mode (on page 14-58)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].overrun (on page 14-225)
trigger.digin[N].wait() (on page 14-225)

trigger.digin[N].edge

This attribute sets the edge used by the trigger event detector on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.digin[N].edge
trigger.digin[N].edge = detectedEdge
```

detectedEdge	The trigger logic value: ■ Detect falling-edge triggers as inputs: `trigger.EDGE_FALLING` ■ Detect rising-edge triggers as inputs: `trigger.EDGE_RISING` ■ Detect either falling or rising-edge triggers as inputs: `trigger.EDGE_EITHER` ■ See **Details** for descriptions of values
N	Digital I/O trigger line (1 to 6)

Details

This command sets the logic on which the trigger event detector and the output trigger generator operate on the specified trigger line.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Trigger mode values

Value	Description
`trigger.EDGE_FALLING`	Detects falling-edge triggers as input when the line is configured as an input or open drain
`trigger.EDGE_RISING`	Detects rising-edge triggers as input when the line is configured as an open drain
`trigger.EDGE_EITHER`	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_IN
trigger.digin[4].edge = trigger.EDGE_RISING
```

Sets the trigger mode for digital I/O line 4 to detect a rising-edge trigger as an input.

Also see

digio.line[N].mode (on page 14-58)
digio.line[N].reset() (on page 14-59)
digio.writeport() (on page 14-62)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].clear() (on page 14-223)

trigger.digin[N].overrun

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Digital I/O trigger *N* clear Digital I/O trigger *N* reset	Not applicable	Not applicable

Usage

```
overrun = trigger.digin[N].overrun
```

overrun	Trigger overrun state (`true` or `false`)
N	Digital I/O trigger line (1 to 6)

Details

If this is `true`, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = trigger.digin[1].overrun
print(overrun)
```

If there is no trigger overrun on digital input 1, the output is:
`false`

Also see

digio.line[N].mode (on page 14-58)
digio.line[N].reset() (on page 14-59)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].clear() (on page 14-223)

trigger.digin[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.digin[N].wait(timeout)
```

triggered	Trigger detected: `true` No triggers detected during the timeout period: `false`
N	Digital I/O trigger line (1 to 6)
timeout	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `trigger.digin[N].wait()` or `trigger.digin[N].clear()` was called, this function returns a value immediately. After waiting for a trigger with this function, the event detector is automatically reset and is ready to detect the next trigger. This is true regardless of the number of events detected.

Example

`digio.line[4].mode = digio.MODE_TRIGGER_IN`
`triggered = trigger.digin[4].wait(3)`
`print(triggered)`
	Waits up to 3 s for a trigger to be detected on trigger line 4, then outputs the results.
	Output if no trigger is detected:
	`false`
	Output if a trigger is detected:
	`true`

Also see

digio.line[N].mode (on page 14-58)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].clear() (on page 14-223)

trigger.digout[N].assert()

This function asserts a trigger pulse on one of the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`trigger.digout[N].assert()`

N	Digital I/O trigger line (1 to 6)

Details

Initiates a trigger event and does not wait for completion. The pulse width that is set determines how long the instrument asserts the trigger.

Example

`digio.line[2].mode = digio.MODE_TRIGGER_OUT`
`trigger.digout[2].pulsewidth = 20e-6`
`trigger.digout[2].assert()`
	Asserts a trigger on digital I/O line 2 with a pulse width of 20 μs.

Also see

digio.line[N].mode (on page 14-58)
Digital I/O port configuration (on page 8-14)
trigger.digout[N].pulsewidth (on page 14-228)

trigger.digout[N].logic

This attribute sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger *N* reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.digout[N].logic
trigger.digout[N].logic = logicType
```

logicType	The output logic of the trigger generator: ■ Assert a TTL-high pulse for output: `trigger.LOGIC_POSITIVE` ■ Assert a TTL-low pulse for output: `trigger.LOGIC_NEGATIVE`
N	Digital I/O trigger line (1 to 6)

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

The output state of the digital I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_OUT
trigger.digout[4].logic = trigger.LOGIC_NEGATIVE
```

Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

digio.line[N].mode (on page 14-58)
digio.line[N].reset() (on page 14-59)
Digital I/O port configuration (on page 8-14)

trigger.digout[N].pulsewidth

This attribute describes the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
width = trigger.digout[N].pulsewidth
trigger.digout[N].pulsewidth = width
```

width	The pulse width (0 to 100 ks)
N	Digital I/O trigger line (1 to 6)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `trigger.digout[N].release()`.

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_OUT
trigger.digout[4].pulsewidth = 20e-6
```

Sets the pulse width for trigger line 4 to 20 µs.

Also see

digio.line[N].mode (on page 14-58)
digio.line[N].reset() (on page 14-59)
Digital I/O port configuration (on page 8-14)
trigger.digout[N].assert() (on page 14-226)
trigger.digout[N].release() (on page 14-228)

trigger.digout[N].release()

This function releases an indefinite length or latched trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digout[N].release()
```

N	Digital I/O trigger line (1 to 6)

Details

Releases a trigger that was asserted with an indefinite pulsewidth time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_OUT
trigger.digout[4].release()
```

Releases digital I/O trigger line 4.

Also see

digio.line[N].mode (on page 14-58)
Digital I/O port configuration (on page 8-14)
trigger.digout[N].assert() (on page 14-226)
trigger.digout[N].pulsewidth (on page 14-228)

trigger.digout[N].stimulus

This attribute selects the event that causes a trigger to be asserted on the digital output line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger *N* reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.digout[N].stimulus
trigger.digout[N].stimulus = event
```

event	The event to use as a stimulus; see **Details**
N	Digital I/O trigger line (1 to 6)

Details

The digital trigger pulsewidth command determines how long the trigger is asserted.

The trigger stimulus for a digital I/O line can be set to one of the trigger events that are described in the following table.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMER`*N*
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINK`*N*

Example

```
digio.line[2].mode = digio.MODE_TRIGGER_OUT
trigger.digout[2].stimulus = trigger.EVENT_TIMER3
```

Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.

Also see

digio.line[N].mode (on page 14-58)
digio.line[N].reset() (on page 14-59)
Digital I/O port configuration (on page 8-14)
trigger.digin[N].clear() (on page 14-223)
trigger.digout[N].assert() (on page 14-226)

trigger.lanin[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanin[N].clear()
```

N	The LAN event number (1 to 8) to clear

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

```
trigger.lanin[5].clear()
```

Clears the event detector with LAN event trigger 5.

Also see

trigger.lanin[N].overrun (on page 14-231)

trigger.lanin[N].edge

This attribute sets the trigger operation and detection mode of the specified LAN event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EDGE_EITHER

Usage

```
edgeMode = trigger.lanin[N].edge
trigger.lanin[N].edge = edgeMode
```

edgeMode	The trigger mode; see the **Details** for more information
N	The LAN event number (1 to 8)

Details

This command controls how the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values		
Mode	**Trigger packets detected as input**	**LAN trigger packet generated for output with a…**
`trigger.EDGE_EITHER`	Rising or falling edge (positive or negative state)	negative state
`trigger.EDGE_FALLING`	Falling edge (negative state)	negative state
`trigger.EDGE_RISING`	Rising edge (positive state)	positive state

Example

```
trigger.lanin[1].edge = trigger.EDGE_FALLING
```

Set the edge state of LAN event 1 to falling.

Also see

Digital I/O (on page 8-12)
TSP-Link system expansion interface (on page 9-1)

trigger.lanin[N].overrun

This attribute contains the overrun status of the LAN event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger *N* clear	Not applicable	Not applicable

Usage

```
overrun = trigger.lanin[N].overrun
```

overrun	The trigger overrun state for the specified LAN packet (`true` or `false`)
N	The LAN event number (1 to 8)

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

Example

`overrun = trigger.lanin[5].overrun` `print(overrun)`	Checks the overrun status of a trigger on LAN5 and outputs the value, such as: `false`

Also see

trigger.lanin[N].clear() (on page 14-230)
trigger.lanin[N].wait() (on page 14-232)
trigger.lanout[N].assert() (on page 14-233)
trigger.lanout[N].stimulus (on page 14-238)

trigger.lanin[N].wait()

This function waits for an input trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.lanin[*N*].wait(*timeout*)

triggered	Trigger detection indication (`true` or `false`)
N	The trigger packet over LAN to wait for (1 to 8)
timeout	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `trigger.lanin[N].wait()` or `trigger.lanin[N].clear()` was called, this function returns immediately.

After waiting for a LAN trigger event with this function, the event detector is automatically reset and rearmed regardless of the number of events detected.

Example

`triggered = trigger.lanin[5].wait(3)`	Wait for a trigger event with LAN trigger 5 with a timeout of 3 s.

Also see

trigger.lanin[N].clear() (on page 14-230)
trigger.lanin[N].overrun (on page 14-231)
trigger.lanout[N].assert() (on page 14-233)
trigger.lanout[N].stimulus (on page 14-238)

trigger.lanout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].assert()
```

N	The LAN event number (1 to 8)

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudo line state to the appropriate state.

The following indexes provide the listed LXI events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- …
- 8:LAN7

Example

```
trigger.lanout[5].assert()
```

Creates a trigger with LAN trigger 5.

Also see

lan.lxidomain (on page 14-98)
trigger.lanin[N].clear() (on page 14-230)
trigger.lanin[N].overrun (on page 14-231)
trigger.lanin[N].wait() (on page 14-232)
trigger.lanout[N].assert() (on page 14-233)
trigger.lanout[N].ipaddress (on page 14-236)
trigger.lanout[N].protocol (on page 14-237)
trigger.lanout[N].stimulus (on page 14-238)

trigger.lanout[N].connect()

This function prepares the event generator for outgoing trigger events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].connect()
```

N	The LAN event number (1 to 8)

Details

This command prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event is changed.

Example

```
trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST
trigger.lanout[1].connect()
trigger.lanout[1].assert()
```

Set the protocol for LAN trigger 1 to be multicast when sending LAN triggers. Then, after connecting the LAN trigger, send a message on LAN trigger 1 by asserting it.

Also see

trigger.lanin[N].overrun (on page 14-231)
trigger.lanin[N].wait() (on page 14-232)
trigger.lanout[N].assert() (on page 14-233)
trigger.lanout[N].ipaddress (on page 14-236)
trigger.lanout[N].protocol (on page 14-237)
trigger.lanout[N].stimulus (on page 14-238)

trigger.lanout[N].connected

This attribute contains the LAN event connection state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
connected = trigger.lanout[N].connected
```

connected	The LAN event connection state: ■ `true`: Connected ■ `false`: Not connected
N	The LAN event number (1 to 8)

Details

This is set to `true` when the LAN trigger is connected and ready to send trigger events after a successful `trigger.lanout[N].connect()` command. If the LAN trigger is not ready to send trigger events, this value is `false`.

This attribute is also `false` when the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attribute is changed or when the remote connection closes the connection.

Example

```
trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST
print(trigger.lanout[1].connected)
```

Outputs `true` if connected, or `false` if not connected.
Example output:
`false`

Also see

trigger.lanout[N].connect() (on page 14-234)
trigger.lanout[N].ipaddress (on page 14-236)
trigger.lanout[N].protocol (on page 14-237)

trigger.lanout[N].disconnect()

This function disconnects the LAN trigger event generator.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].disconnect()
```

N	The LAN event number (1 to 8)

Details

When this command is set for TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attributes for this event are changed.

Also see

trigger.lanout[N].ipaddress (on page 14-236)
trigger.lanout[N].protocol (on page 14-237)

trigger.lanout[N].ipaddress

This attribute specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	"0.0.0.0"

Usage

```
ipAddress = trigger.lanout[N].ipaddress
trigger.lanout[N].ipaddress = "ipAddress"
```

ipAddress	The LAN address for this attribute as a string in dotted decimal notation
N	The LAN event number (1 to 8)

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

```
trigger.lanout[3].protocol = lan.PROTOCOL_TCP
trigger.lanout[3].ipaddress = "192.0.32.10"
trigger.lanout[3].connect()
```

Set the protocol for LAN trigger 3 to be TCP when sending LAN triggers.
Use IP address `"192.0.32.10"` to connect the LAN trigger.

Also see

trigger.lanout[N].connect() (on page 14-234)

trigger.lanout[N].logic

This attribute sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.lanout[N].logic
trigger.lanout[N].logic = logicType
```

logicType	The type of logic: ■ Positive: `trigger.LOGIC_POSITIVE` ■ Negative: `trigger.LOGIC_NEGATIVE`
N	The LAN event number (1 to 8)

Example

```
trigger.lanout[2].logic = trigger.LOGIC_POSITIVE
```

Set the logic for LAN trigger line 2 to positive.

Also see

None

trigger.lanout[N].protocol

This attribute sets the LAN protocol to use for sending trigger messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	lan.PROTOCOL_TCP

Usage

```
protocol = trigger.lanout[N].protocol
trigger.lanout[N].protocol = protocol
```

protocol	The protocol to use for messages from the trigger: ▪ `lan.PROTOCOL_TCP` ▪ `lan.PROTOCOL_UDP` ▪ `lan.PROTOCOL_MULTICAST`
N	The LAN event number (1 to 8)

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored, and event messages are sent to the multicast address 224.0.23.159.

Example

`print(trigger.lanout[1].protocol)`	Get LAN protocol that is being used for sending trigger messages for LAN event 1.

Also see

trigger.lanout[N].connect() (on page 14-234)
trigger.lanout[N].ipaddress (on page 14-236)

trigger.lanout[N].stimulus

This attribute specifies events that cause this trigger to assert.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EVENT_NONE

Usage

```
LANevent = trigger.lanout[N].stimulus
trigger.lanout[N].stimulus = LANevent
```

LANevent	The LAN event that causes this trigger to assert; see **Details** for values
N	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored, and the action overrun is set.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMER`*N*
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINK`*N*

Example

```
trigger.lanout[5].stimulus = trigger.EVENT_TIMER1
```

Use the timer 1 trigger event as the source for LAN trigger 5 stimulus.

Also see

trigger.lanout[N].connect() (on page 14-234)
trigger.lanout[N].ipaddress (on page 14-236)

trigger.model.abort()

This function stops all trigger model commands on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.abort()
```

Details

When this command is received, the instrument stops the trigger model.

Example

`trigger.model.abort()`	Terminates all commands related to the trigger model on the instrument.

Also see

Effect of GPIB line events on 2470 (on page 2-12)
Aborting the trigger model (on page 8-50)
Trigger model (on page 8-25)

trigger.model.getblocklist()

This function returns the settings for all trigger model blocks.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.getblocklist()
```

Details

This returns the settings for the trigger model.

Example

```
print(trigger.model.getblocklist())
```

Returns the settings for the trigger model. Example output is:

```
 1) BUFFER_CLEAR          BUFFER: defbuffer1
 2) MEASURE               BUFFER: defbuffer1 COUNT: 1
 3) BRANCH_COUNTER        VALUE: 5  BRANCH_BLOCK: 2
 4) DELAY_CONSTANT        DELAY: 1.000000000
 5) BRANCH_COUNTER        VALUE: 3  BRANCH_BLOCK: 2
```

Also see

trigger.model.getbranchcount() (on page 14-240)

trigger.model.getbranchcount()

This function returns the count value of the trigger model counter block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.getbranchcount(blockNumber)
```

blockNumber	The sequence of the block in the trigger model

Details

This command returns the counter value. When the counter is active, this returns the present count. If the trigger model has started or is running but has not yet reached the counter block, this value is 0.

Example

```
reset()
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 0.1)
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
trigger.model.initiate()
delay(1)
print(trigger.model.getbranchcount(4))
waitcomplete()
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and make five readings.
Delay 0.1 s.
Loop ten more times back to block 2.
Send the count command to check which count has been completed for block 4.
At end of execution, 10 readings are stored in `defbuffer1`.

Also see

trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 14-253)

trigger.model.initiate()

This function starts the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.initiate()
```

Also see

Trigger model (on page 8-25)
trigger.model.abort() (on page 14-239)

trigger.model.load() — ConfigList

This function loads a trigger-model template configuration that uses source and measure configuration lists.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("ConfigList", "measureConfigList", "sourceConfigList")
trigger.model.load("ConfigList", "measureConfigList", "sourceConfigList", delay)
trigger.model.load("ConfigList", "measureConfigList", "sourceConfigList", delay,
   bufferName)
```

measureConfigList	A string that contains the name of the measurement configuration list to use
sourceConfigList	A string that contains the name of the source configuration list to use
delay	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`.

Details

This trigger-model template incorporates a source configuration list and measure configuration list. You must set up the configuration lists before loading the trigger model. If the configuration lists change, you must resend this command.

You can also set a delay and change the reading buffer.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
reset()
smu.source.configlist.create("SOURCE_LIST")
smu.measure.configlist.create("MEASURE_LIST")
smu.source.level = 1
smu.source.configlist.store("SOURCE_LIST")
smu.measure.range = 1e-3
smu.measure.configlist.store("MEASURE_LIST")
smu.source.level = 5
smu.source.configlist.store("SOURCE_LIST")
smu.measure.range = 10e-3
smu.measure.configlist.store("MEASURE_LIST")
smu.source.level = 10
smu.source.configlist.store("SOURCE_LIST")
smu.measure.range = 100e-3
smu.measure.configlist.store("MEASURE_LIST")
trigger.model.load("ConfigList", "MEASURE_LIST", "SOURCE_LIST")
trigger.model.initiate()
```

Set up a source configuration list named `SOURCE_LIST` and a measurement configuration list named `MEASURE_LIST`. Load the configuration list trigger model, using these two configuration lists. Start the trigger model.

Also see

None

trigger.model.load() — DurationLoop

This function loads a trigger-model template configuration that makes continuous measurements for a specified amount of time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("DurationLoop", duration)
trigger.model.load("DurationLoop", duration, delay)
trigger.model.load("DurationLoop", duration, delay, bufferName)
```

duration	The amount of time for which to make measurements (167 ns to 100 ks)
delay	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`

Details

When you load this trigger-model template, you can specify amount of time to make a measurement and the length of the delay before the measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.level = 5

-- Turn on output, initiate readings, and store them in defbuffer1
trigger.model.load("DurationLoop", 10, 0.01, defbuffer1)
trigger.model.initiate()
```

Reset the instrument. Set the instrument to source voltage at 5 V. Set to measure current.
Load the duration loop trigger model to take measurements for 10 s with a 10 ms delay before each measurement.
Start the trigger model.

Also see

None

trigger.model.load() — Empty

This function clears the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.load("Empty")
```

Details

When you load this trigger-model template, any blocks that have been defined in the trigger model are cleared so the trigger model has no blocks defined.

Example

```
trigger.model.load("Empty")
print(trigger.model.getblocklist())
```

Clear the trigger model to have no blocks defined.
Output:
EMPTY

Also see

None

trigger.model.load() — GradeBinning

This function loads a trigger-model template configuration that sets up a grading operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern, bufferName)
```

components	The number of components to measure (1 to 268,435,455)
startInLine	The input line that starts the test; 5 for digital line 5, 6 for digital line 6; default is 5
startDelay	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
endDelay	The delay time after the measurement (167 ns to 10 ks); default is 0 for no delay
limitxHigh	*x* is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
limitxLow	*x* is 1, 2, 3, or 4; the lower limit that the measurement is compared against
limit1Pattern	The bit pattern that is sent when the measurement fails limit 1; range 1 to 15; default is 1

limit2Pattern	The bit pattern that is sent when the measurement fails limit 2; range 1 to 15; default is 2
limit3Pattern	The bit pattern that is sent when the measurement fails limit 3; range 1 to 15; default is 4
limit4Pattern	The bit pattern that is sent when the measurement fails limit 4; range 1 to 15; default is 8
allPattern	The bit pattern that is sent when all limits have passed; 1 to 15; default is 15
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`

Details

This trigger-model template allows you to grade components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the pass pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Also see

None

trigger.model.load() — LogicTrigger

This function loads a trigger-model template configuration that sets up a logic trigger through the digital I/O.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear)
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear, delay)
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear, delay,
   bufferName)
```

digInLine	The digital input line (1 to 6); also, the event that the trigger model will wait on in block 1
digOutLine	The digital output line (1 to 6)
count	The number of measurements the instrument will make
clear	To clear previously detected trigger events when entering the wait block: `trigger.CLEAR_ENTER` To immediately act on any previously detected triggers and not clear them (default): `trigger.CLEAR_NEVER`
delay	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`

Details

This trigger model waits for a digital input event to occur, makes a measurement, and issues a notify event. The notify event asserts a digital output line.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
trigger.model.load("LogicTrigger", 1, 2, 10, 0.001, defbuffer1)
```

Set up the template to use the digital in line and wait for a pulse from digital in line 1 to trigger measurements.
Pulse digital out line 2 when the measurement is complete.
Make 10 measurements, with a delay of 1 ms before each measurement.
Store the measurements in `defbuffer1`.

Also see

trigger.digout[N].logic (on page 14-227)

trigger.model.load() — LoopUntilEvent

This function loads a trigger-model template configuration that makes continuous measurements until the specified event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear)
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear, delay)
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear, delay,
    bufferName)
```

triggerEvent	The event that ends infinite triggering or readings set to occur before the trigger; see **Details**
position	The number of readings to make in relation to the size of the reading buffer; enter as a percentage (0% to 100%)
clear	To clear previously detected trigger events when entering the wait block (default): `trigger.CLEAR_ENTER` To immediately act on any previously detected triggers and not clear them: `trigger.CLEAR_NEVER`
delay	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
bufferName	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`

Details

The event constant is the event that ends infinite triggering or ends readings set to occur before the trigger and start post-trigger readings. The trigger model makes readings until it detects the event constant. After the event, it makes a finite number of readings, based on the setting of the trigger position.

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled. For example, if this is set to 75 for a reading buffer that holds 10,000 readings, the trigger model makes 2500 readings after it detects the source event. There are 7500 pre-trigger readings and 2500 post-trigger readings.

The instrument makes two sets of readings. The first set is made until the trigger event occurs. The second set is made after the trigger event occurs, up to the number of readings calculated by the position parameter.

You cannot have the event constant set at none when you run this trigger-model template.

The following table lists the options that are available for *triggerEvent*.

Trigger events	
Event description	**Event constant**
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINK`*N*
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMER`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT

-- Initiate readings
trigger.model.load("LoopUntilEvent", trigger.EVENT_DISPLAY, 50)
trigger.model.initiate()
```

Reset the instrument.
Set the instrument to measure current.
Load the LoopUntilEvent trigger model to make measurements until the front panel trigger key is pressed, then continue to make measurements equal to 50% of the reading buffer size.
Start the trigger model.

Also see

None

trigger.model.load() — SimpleLoop

This function loads a trigger-model template configuration that makes a specific number of measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("SimpleLoop", count)
trigger.model.load("SimpleLoop", count, delay)
trigger.model.load("SimpleLoop", count, delay, bufferName)
```

count	The number of measurements the instrument will make
delay	The delay time before each measurement (167 ns to 10 ks); default is `0` for no delay
bufferName	Indicates the reading buffer to use; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, `defbuffer1` is used

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Example

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

-- Turn on output and initiate readings
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
      print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

-- Discharge the capacitor to 0 V and turn off the output
smu.source.level = 0
delay(2)
smu.source.output = smu.OFF
```

This example uses the SimpleLoop trigger-model template to do a capacitor test. This example produces 200 readings that have output similar to the following example:

```
Rdg #   Time (s)   Current (A)
1   0             8.5718931952528e-11
2   0.151875      1.6215984111057e-10
3   0.303727      1.5521139928865e-10
. . .
198   29.91579194   1.5521250951167e-10
199   30.067648716   1.4131290582142e-10
200   30.219497716   1.5521067764368e-10
```

Also see

None

trigger.model.load() — SortBinning

This function loads a trigger-model template configuration that sets up a sorting operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern, bufferName)
```

components	The number of components to measure (1 to 268,435,455)
limitxHigh	*x* is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
limitxLow	*x* is 1, 2, 3, or 4; the lower limit that the measurement is compared against
limit1Pattern	The bit pattern that is sent when the measurement passes limit 1; range 1 to 15; default is 1
limit2Pattern	The bit pattern that is sent when the measurement passes limit 2; range 1 to 15; default is 2
limit3Pattern	The bit pattern that is sent when the measurement passes limit 3; range 1 to 15; default is 4

limit4Pattern	The bit pattern that is sent when the measurement passes limit 4; range 1 to 15; default is 8
allPattern	The bit pattern that is sent when all limits have failed; 1 to 15; default is 15
startInLine	The input line that starts the test; 5 for digital line 5, 6 for digital line 6; default is 5
startDelay	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
endDelay	The delay time after the measurement (167 ns to 10 ks); default is 0 for no delay
bufferName	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1

Details

This trigger-model template allows you to sort components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the all fail pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Also see

None

trigger.model.pause()

This function pauses a running trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.pause()
```

Details

This command pauses the trigger model.

To continue the trigger model, send the resume command.

Example

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
```

```
trigger.model.pause()
delay(10)
trigger.model.resume()
waitcomplete()
print(defbuffer1.n)
```

Set up a trigger model, then run it, pause for delay of 10 seconds, then resume it.

Also see

trigger.model.initiate() (on page 14-241)
trigger.model.resume() (on page 14-252)

trigger.model.resume()

This function continues a paused trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.resume()
```

Details

This command continues running the trigger model operation if the trigger model was paused.

Example

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
trigger.model.pause()
delay(10)
trigger.model.resume()
waitcomplete()
print(defbuffer1.n)
```

Set up a trigger model, then run it, pause for delay of 10 seconds, then resume it.

Also see

trigger.model.initiate() (on page 14-241)
trigger.model.pause() (on page 14-251)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS

This function defines a trigger model block that always goes to a specific block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ALWAYS, branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number to execute when the trigger model reaches the Branch Always block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by *branchToBlock*.

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ALWAYS, 20)
```

When the trigger model reaches block 6, always branch to block 20.

Also see

None

trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER

This function defines a trigger model block that branches to a specified block a specified number of times.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_COUNTER, targetCount,
   branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
targetCount	The number of times to repeat
branchToBlock	The block number of the trigger model block to execute when the counter is less than the *targetCount* value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value. At that point, the trigger model continues to the next building block in the sequence.

The counter is reset to 0 when the trigger model starts. It is incremented each time trigger model execution reaches the counter block.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting. Use `trigger.model.getbranchcount()` to query the counter.

Example

```
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
print(trigger.model.getbranchcount(4))
```

When the trigger model reaches this block, the trigger model returns to block 2. This repeats 10 times.
An example of the return if the trigger model has reached this block 5 times is:
5

Also see

trigger.model.getbranchcount() (on page 14-240)
trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT (on page 14-274)

trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA

This function defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
    branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
    branchToBlock, measureBlock)
```

blockNumber	The sequence of the block in the trigger model
targetDifference	The value against which the block compares the difference between the measurements
branchToBlock	The block number of the trigger model block to execute when the difference between the measurements is less than or equal to the *targetDifference*
measureBlock	The block number of the measure/digitize block that makes the measurements to be compared; if this is `0` or undefined, the trigger model uses the previous measure/digitize block

Details

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_DELTA, 0.35, 8, 3)
```

Configure trigger block 5 to branch to block 8 when the measurement difference from block 3 is less than 0.35.

Also see

Delta block (on page 8-40)

trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT

This function defines a trigger model block that goes to a specified block if a measurement meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
    limitA, limitB, branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
    limitA, limitB, branchToBlock, measureBlock)
```

blockNumber	The sequence of the block in the trigger model
limitType	The type of limit, which can be one of the following types: ■ `trigger.LIMIT_ABOVE` ■ `trigger.LIMIT_BELOW` ■ `trigger.LIMIT_INSIDE` ■ `trigger.LIMIT_OUTSIDE`

limitA	The lower limit that the measurement is tested against; if *limitType* is set to: ■ trigger.LIMIT_ABOVE: This value is ignored ■ trigger.LIMIT_BELOW: The measurement must be below this value ■ trigger.LIMIT_INSIDE: The low limit that the measurement is compared against ■ trigger.LIMIT_OUTSIDE: The low limit that the measurement is compared against
limitB	The upper limit that the measurement is tested against; if *limitType* is set to: ■ trigger.LIMIT_ABOVE: The measurement must be above this value ■ trigger.LIMIT_BELOW: This value is ignored ■ trigger.LIMIT_INSIDE: The high limit that the measurement is compared against ■ trigger.LIMIT_OUTSIDE: The high limit that the measurement is compared against
branchToBlock	The block number of the trigger model block to execute when the measurement meets the defined criteria
measureBlock	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected
- Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected
- Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
- Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value

The measurement block must be a measure/digitize block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measure/digitize block is used.

If the limit A is more than the limit B, the values are automatically swapped so that the lesser value is used as the lower limit.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_ABOVE,
   0.1, 1, 2)
```

Sets trigger block 5 to be a constant limit that branches to block 2 when the measurement is above the value set for limit B (which is set to 1). Note that limit A must be set but is ignored.

Also see

Constant Limit block (on page 8-38)

trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC

This function defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    limitNumber, branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    limitNumber, branchToBlock, measureBlock)
```

blockNumber	The sequence of the block in the trigger model
limitType	The type of limit, which can be one of the following types: ■ `trigger.LIMIT_ABOVE` ■ `trigger.LIMIT_BELOW` ■ `trigger.LIMIT_INSIDE` ■ `trigger.LIMIT_OUTSIDE`
limitNumber	The limit number (`1` or `2`)
branchToBlock	The block number of the trigger model block to execute when the measurement meets the criteria set in the configuration list
measureBlock	The block number of the measure/digitize block that makes the measurements to be compared; if this is `0` or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-dynamic-limits block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

Example

```
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC,
   trigger.LIMIT_OUTSIDE, 2, 10, 5)
```

Configure block 7 to check if limit 2 is outside its limit values, based on the measurements made in block 5. If values are outside the measurements, branch to block 10. If the values are not outside the measurements, trigger model execution continues to block 8.

Also see

Dynamic Limit block (on page 8-39)
smu.measure.limit[Y].low.value (on page 14-150)
smu.measure.limit[Y].high.value (on page 14-149)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT

This function branches to a specified block when a specified trigger event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ON_EVENT, event,
   branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
event	The event that must occur before the trigger model branches the specified block
branchToBlock	The block number of the trigger model block to execute when the specified event occurs

Details

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The following table shows the constants for the events.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMER`*N*
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINK`*N*

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ON_EVENT, trigger.EVENT_DISPLAY, 2)
```

When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the next block in the trigger model).

Also see

On event block (on page 8-37)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE

This function causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE, branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time trigger model execution encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when trigger model execution reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model execution encounters this block.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE, 4)
```

When the trigger model reaches block 2, the trigger model goes to block 4 instead of going in the default sequence of block 3.

Also see

Once block (on page 8-41)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED

This function defines a trigger model block that causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE_EXCLUDED,
    branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE_EXCLUDED, 4)
```

When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

Once excluded block (on page 8-41)

trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR

This function defines a trigger model block that clears the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR, bufferName)
```

blockNumber	The sequence of the block in the trigger model
bufferName	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used

Details

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

Example

```
trigger.model.setblock(3, trigger.BLOCK_BUFFER_CLEAR, capTest2)
```

Assign trigger block 3 to buffer clear; when the trigger model reaches block 3, it clears the reading buffer named `capTest2`.

Also see

buffer.make() (on page 14-13)
Buffer clear block (on page 8-27)

trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT

This function recalls the settings at the next index of a source or measure configuration list, or both a source and measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_NEXT, "configurationList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_NEXT, "configurationList",
   "optionalConfigList")
```

blockNumber	The sequence of the block in the trigger model
configurationList	A string that defines the source or measure configuration list to recall
optionalConfigList	The name of the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list

Details

If two configuration lists are specified with this command, they must not be of the same type. For example, if the first configuration list is a measure configuration list, the second configuration list must be a source configuration list. The order of the configuration lists does not matter with this command, as long as they are of the opposite type.

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger model block. If the configuration list has recalled an index with the recall block, the next index in the list is recalled instead of the first. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the next block is reached after that recall, index 2 is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Next block.

Example 1

```
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "measTrigList")
```

Configure trigger block 5 to load the next index in the configuration list named `measTrigList`.

Example 2

```
trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "measTrigList")
trigger.model.setblock(2, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(3, trigger.BLOCK_CONFIG_NEXT, "measTrigList")
print(trigger.model.getblocklist())
```

Clear the trigger model.
Recall index 1 of a configuration list named `measTrigList`.
Clear reading buffer named `defbuffer1`.
Recall the second index of a configuration list named `measTrigList`.
Output:

```
1) CONFIG_RECALL          CONFIG_LIST: measTrigList  INDEX: 1
2) BUFFER_CLEAR           BUFFER: defbuffer1
3) CONFIG_NEXT            CONFIG_LIST: measTrigList
```

Example 3

```
trigger.model.setblock(7, trigger.BLOCK_CONFIG_NEXT, "measTrigList",
   "sourTrigList")
```

Configure trigger block 7 to load the next index in both the configuration list named `measTrigList` and the configuration list named `sourTrigList`.

Also see

Configuration lists (on page 4-82)

trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV

This function defines a trigger model block that recalls the settings stored at the previous index in a source or measure configuration list, or both a source and measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_PREV, "configurationList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_PREV, "configurationList",
   "optionalConfigList")
```

blockNumber	The sequence of the block in the trigger model
configurationList	A string that defines the source or measure configuration list to recall
optionalConfigList	A string that defines the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list

Details

If two configuration lists are specified with this command, they must not be of the same type. For example, if the first configuration list is a measure configuration list, the second configuration list must be a source configuration list. The order of the configuration lists does not matter with this command, as long as they are of the opposite type.

The Config List Prev block defines a trigger model block that recalls the settings stored at the previous index in a source or measure configuration list if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are each recalled from the previous index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger model block. If the configuration list has recalled an index with the recall block, the previous index in the list is called instead of the last. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the previous block is reached after that recall, the last index is recalled. However, if the recall block recalled index 3, the previous block would recall index 2. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Prev block.

Example 1

```
trigger.model.setblock(8, trigger.BLOCK_CONFIG_PREV, "measTrigList")
```

Configure trigger block 8 to load the previous index in the configuration list named `measTrigList`.

Example 2

```
trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 3)
trigger.model.setblock(2, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(3, trigger.BLOCK_CONFIG_PREV, "measTrigList")
print(trigger.model.getblocklist())
```

Clear the trigger model.
Recall index 3 of a configuration list named `measTrigList`.
Clear reading buffer named `defbuffer1`.
Then, recall the second index of a configuration list named `measTrigList`.
Output:

```
1) CONFIG_RECALL         CONFIG_LIST: measTrigList  INDEX: 3
2) BUFFER_CLEAR          BUFFER: defbuffer1
3) CONFIG_PREV           CONFIG_LIST: measTrigList
```

Example 3

```
trigger.model.setblock(7, trigger.BLOCK_CONFIG_PREV, "measTrigList",
   "sourTrigList")
```

Configure trigger block 7 to load the previous index in both the configuration list named `measTrigList` and the configuration list named `sourTrigList`.

Also see

Configuration lists (on page 4-82)

trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL

This function recalls the system settings that are stored in a source or measure configuration list, or both a source and measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
   "configurationList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
   "configurationList", index)
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
   "configurationList", index, "optionalConfigList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
   "configurationList", index, "optionalConfigList", optionalIndex)
```

blockNumber	The sequence of the block in the trigger model
configurationList	A string that defines the source or measure configuration list to recall
index	The index in the configuration list to recall; default is `1`
optionalConfigList	A string that defines the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list
optionalIndex	The index to recall from the second configuration list; defaults to `1`

Details

If two configuration lists are specified with this command, they must not be of the same type. For example, if the first configuration list is a measure configuration list, the second configuration list must be a source configuration list. The order of the configuration lists does not matter with this command, as long as they are of the opposite type.

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration lists must be defined before you can use this block. If one of the indices for the configuration list changes, verify that the trigger model count is still accurate.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Recall block.

Example 1

```
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 5)
```

Configure trigger block 3 to load index 5 from the configuration list named `measTrigList`.

Example 2

```
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 5,
   "sourTrigList")
print(trigger.model.getblocklist())
```

Configure trigger block 3 to load index 5 from the configuration list named `measTrigList` and load index 1 from the configuration list name `sourTrigList`.
Query the configuration of the block.
Output:

```
5) CONFIG_RECALL   CONFIG_LIST: measTrigList and sourTrigList  INDEX: 5 and 1
```

Also see

Configuration lists (on page 4-82)

trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT

This function adds a constant delay to the execution of a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

`trigger.model.setblock(`*`blockNumber`*`, trigger.BLOCK_DELAY_CONSTANT, `*`time`*`)`

`blockNumber`	The sequence of the block in the trigger model
`time`	The amount of time to delay in seconds (167 ns to 10 ks, or 0 for no delay)

Details

When trigger model execution reaches a delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

Example

```
trigger.model.setblock(7, trigger.BLOCK_DELAY_CONSTANT, 30e-3)
```

Configure trigger block 7 to delay the trigger model before the next block until a delay of 30 ms elapses.

Also see

None

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC

This function adds a user delay to the execution of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DELAY_DYNAMIC,
   trigger.USER_DELAY_Mn)
```

blockNumber	The sequence of the block in the trigger model
userDelay	The number of the user delay: ■ `trigger.USER_DELAY_M`*n*, where *n* is the number of the user delay (1 to 5) set by `smu.measure.userdelay[`*N*`]` ■ `trigger.USER_DELAY_S`*n*, where *n* is the number of the user delay (1 to 5) set by `smu.source.userdelay[`*N*`]`

Details

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Each source function can also have up to five unique user delay times (S1 to S5). The delay time is set by the user-delay command, which is only available over a remote interface.

Though the trigger model can be used with any function, the user delay is set per function. Make sure you are setting the delay for the function you intend to use with the trigger model.

Example

```
smu.measure.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay for measure 1 to 5 s.
Set trigger block 1 to turn the source output on.
Set trigger block 2 to a dynamic delay that calls measure user delay 1.
Set trigger block 3 to make a measurement.
Set trigger block 4 to turn the source output off.
Set trigger block 5 to branch to block 1 ten times.
Start the trigger model.

Also see

smu.measure.userdelay[N] (on page 14-168)
smu.source.userdelay[N] (on page 14-200)

trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO

This function defines a trigger model block that sets the lines on the digital I/O port high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DIGITAL_IO, bitPattern, bitMask)
```

blockNumber	The sequence of the block in the trigger model
bitPattern	Sets the value that specifies the output line bit pattern (0 to 63)
bitMask	Specifies the bit mask; if omitted, all lines are driven (0 to 63)

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern. The pattern can be specified as a six-bit binary, hexadecimal, or integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63, 0x3F, 0b111111) or omit this parameter. If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

```
for x = 3, 6 do digio.line[x].mode = digio.MODE_DIGITAL_OUT end
trigger.model.setblock(4, trigger.BLOCK_DIGITAL_IO, 20, 60)
```

The for loop configures digital I./O lines 3 through 6 as digital outputs. Trigger block `4` is then configured with a bit pattern of `20` (digital I/O lines 3 and 5 high). The optional bit mask is specified as `60` (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

digio.line[N].mode (on page 14-58)

trigger.model.setblock() — trigger.BLOCK_LOG_EVENT

This function allows you to log an event in the event log when the trigger model is running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_LOG_EVENT, eventNumber, "message")
```

blockNumber	The sequence of the block in the trigger model
eventNumber	The event number: ▪ `trigger.LOG_INFO`*N* ▪ `trigger.LOG_WARN`*N* ▪ `trigger.LOG_ERROR`*N* Where *N* is 1 to 4; you can define up to four of each type You can also set `trigger.LOG_WARN_ABORT`, which aborts the trigger model immediately and posts a warning event log message
message	A string up to 31 characters

Details

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

Example

```
trigger.model.setblock(9, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Trigger model
    complete.")
```

Set trigger model block 9 to log an event when the trigger model completes. In the event log, the message is:
`TM #1 block #9 logged: Trigger model complete.`

Also see

None

trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE

This function defines a trigger block that makes or digitizes a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE, bufferName)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE, bufferName,
    count)
```

blockNumber	The sequence of the block in the trigger model
bufferName	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used
count	The number of measure or digitize readings to make before moving to the next block in the trigger model; set to: ■ A specific value ■ Infinite (run continuously until stopped): `trigger.COUNT_INFINITE` ■ Stop infinite to stop the block: `trigger.COUNT_STOP` ■ Use most recent count value: `trigger.COUNT_AUTO`

Details

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

Digitized measurements are not a feature on the 2470. However, you can use this command to communicate with other Keithley instruments that do offer the digitized measurements feature and to share code with other Keithley instruments.

Firmware versions of the 2470 before version 1.7.0 had a separate measure block. If you have code that is using that block, it works in this version of the 2470 firmware.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1. For example, `trigger.model.setblock(1, trigger.BLOCK_MEASURE)` changes to `trigger.model.setblock(1, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1, 1)`.

Example 1

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.
Set the function to measure DC voltage.
Set block 1 to clear `defbuffer1`.
Set block 2 to set a delay of `0`.
Set block 3 to make measurements infinitely.
Set block 4 to wait until the front-panel TRIGGER key is pressed.
Set block 5 to stop making measurements.
Set block 6 to send a notification.
Start the trigger model.
You must press the front-panel TRIGGER key to stop measurements.
Output the number of readings.

Example 2

```
reset()
smu.measure.configlist.create("countactive")
smu.measure.count = 2
smu.measure.configlist.store("countactive")  -- index1
smu.measure.count = 10
smu.measure.configlist.store("countactive")  -- index2
smu.measure.count = 3
smu.measure.configlist.store("countactive")  -- index3

trigger.model.setblock(1, trigger.BLOCK_CONFIG_NEXT, "countactive")
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
   trigger.COUNT_AUTO)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 1)
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 3, 1)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.
Set up a configuration list named `countactive`.

```
Set the measure count to 2 (replace smu.measure.count with smu.digitize.count if using a digitize
function.)
Store the count in index 1.
Set the measure count to 10.
Store the count in index 2.
Set the measure count to 3.
Store the count in index 3.
Set up trigger model block 1 to call the next index from the countactive configuration list.
Set block 2 to measure or digitize and store the readings in defbuffer1, using the most recent count value.
Set block 3 to add a delay of 1 s.
Set block 4 to iterate through the trigger model 3 times, returning to block 1.
Start the trigger model.
Output the number of readings. There should be 15 readings.
```

Also see

buffer.make() (on page 14-13)
Measure/Digitize block (on page 8-27)

trigger.model.setblock() — trigger.BLOCK_NOP

This function creates a placeholder that performs no action in the trigger model; available only using remote commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOP)
```

blockNumber	The sequence of the block in the trigger model

Details

If you remove a trigger model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

```
trigger.model.setblock(4, trigger.BLOCK_NOP)
```

Set block number 4 to be a no operation block.

Also see

None

trigger.model.setblock() — trigger.BLOCK_NOTIFY

This function defines a trigger model block that generates a trigger event and immediately continues to the next block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFYN)
```

blockNumber	The sequence of the block in the trigger model
N	The identification number of the notification; 1 to 8

Details

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

Define trigger model block 5 to be the notify 2 event. Assign the notify 2 event to be the stimulus for digital output line 3.

Also see

Notify block (on page 8-31)

trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT

This function creates a block in the trigger model that resets a branch counter to 0.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_RESET_BRANCH_COUNT, counter)
```

blockNumber	The sequence of the block in the trigger model
counter	The block number of the counter that is to be reset

Details

When the trigger model reaches the Counter Reset block, it resets the count of the specified Branch on Counter block to zero.

Example

```
trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_BRANCH_COUNTER, 5, 2)
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 3, 2)
trigger.model.setblock(6, trigger.BLOCK_RESET_BRANCH_COUNT, 3)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and take 5 readings.
Delay a second.
Loop three more times back to block 2.
Reset block 3 to 0.
Start the trigger model and wait for measurements to complete.
Print the number of readings in the buffer.
Output:
15

Also see

trigger.model.getbranchcount() (on page 14-240)
trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 14-253)

trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT

This function defines a trigger block that turns the output source on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_SOURCE_OUTPUT, state)
```

blockNumber	The sequence of the block in the trigger model
state	Turn the source off: `smu.OFF` Turn the source on: `smu.ON`

Details

The source output block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as `SOURCE_OUTPUT`.

Example

```
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
```

Set trigger model to turn the source on when it reaches block 2.

Also see

Wait block (on page 8-29)

trigger.model.setblock() — trigger.BLOCK_WAIT

This function defines a trigger model block that waits for an event before allowing the trigger model to continue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear, logic, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear, logic, event,
    event)
```

blockNumber	The sequence of the block in the trigger model
event	The event that must occur before the trigger block allows trigger execution to continue (see **Details**)
clear	To clear previously detected trigger events when entering the wait block: `trigger.CLEAR_ENTER` To immediately act on any previously detected triggers and not clear them (default): `trigger.CLEAR_NEVER`
logic	If each event must occur before the trigger model continues: `trigger.WAIT_AND` If at least one of the events must occur before the trigger model continues: `trigger.WAIT_OR`

Details

You can use the wait block to synchronize measurements with other instruments and devices.

You can set the instrument to wait for the following events:

- Front-panel TRIGGER key press
- Notify (only available when using remote commands)
- Command interface trigger
- Digital input/output signals, such as DIGIO and TSP-Link
- LAN
- Blender
- Timer
- Source limit condition

The event can occur before trigger model execution reaches the wait block. If the event occurs after trigger model execution starts but before the trigger model execution reaches the wait block, the trigger model records the event. By default, when trigger model execution reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when trigger model execution is at the start block and when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action; you cannot combine `AND` and `OR` logic in a single block.

You cannot leave the first event as no trigger. If the first event is not defined, the trigger model errors when you attempt to initiate it.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a notify block before the wait block. Set the notify block to use the same timer as the wait block.

The following table shows the constants for the events.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMER`*N*
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINK`*N*

Example

```
trigger.model.setblock(9, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
```

Set trigger model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing.

Also see

Wait block (on page 8-29)

trigger.model.state()

This function returns the present state of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status, status, n = trigger.model.state()
```

status	The status of the trigger model: ■ `trigger.STATE_ABORTED` ■ `trigger.STATE_ABORTING` ■ `trigger.STATE_BUILDING` ■ `trigger.STATE_EMPTY` ■ `trigger.STATE_FAILED` ■ `trigger.STATE_IDLE` ■ `trigger.STATE_RUNNING` ■ `trigger.STATE_WAITING`
n	The last trigger model block that was executed

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model last executed.

The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped

Example

`print(trigger.model.state())`	An example output if the trigger model is waiting and is at block 9 would be: `trigger.STATE_WAITING trigger.STATE_EMPTY 9`

Also see

None

trigger.timer[N].clear()

This function clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].clear()
```

N	Trigger timer number (1 to 4)

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

```
trigger.timer[1].clear()
```
Clears trigger timer 1.

Also see

trigger.timer[N].count (on page 14-279)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event or is enabled as a timer or alarm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

count	Number of times to repeat the trigger (0 to 1,048,575)
N	Trigger timer number (1 to 4)

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example 1

```
print(trigger.timer[1].count)
```

Read trigger count for timer number 1.

Example 2

```
reset()
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY8
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON

trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY8)
trigger.model.setblock(3, trigger.BLOCK_WAIT, trigger.EVENT_TIMER4)
trigger.model.setblock(4, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 20, 3)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.
Reset trigger timer 4.
Set trigger timer 4 to have a 0.5 s delay.
Set the stimulus for trigger timer 4 to be the notify 8 event.
Set the trigger timer 4 stimulus to off.
Set the timer event to occur when the timer delay elapses.
Set the trigger timer 4 count to 20.
Enable trigger timer 4.

Clear the trigger model.
Set trigger model block 1 to clear the buffer.
Set trigger model block 2 to generate the notify 8 event.
Set trigger model block 3 to wait for the trigger timer 4 to occur.
Set trigger model block 4 to make a measurement and store it in default buffer 1.
Set trigger model block 5 to repeat the trigger model 20 times, starting at block 3.
Start the trigger model.
Wait until all commands are complete.
Print the number of entries in default buffer 1.

Output:
`20`

Also see

trigger.timer[N].clear() (on page 14-279)
trigger.timer[N].delay (on page 14-281)
trigger.timer[N].reset() (on page 14-283)

trigger.timer[N].delay

This attribute sets and reads the timer delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

interval	Delay interval in seconds (8 µs to 100 ks)
N	Trigger timer number (1 to 4)

Details

Once the timer is enabled, each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

```
trigger.timer[N].delaylist = {interval}
```

This creates a delay list of one value.

Reading this attribute returns the delay interval that will be used the next time the timer is triggered.

If you use the trigger timer with a trigger model, make sure the trigger timer delay is set so that the readings are paced correctly.

Example

`trigger.timer[1].delay = 50e-6`	Set the trigger timer 1 to delay for 50 µs.

Also see

trigger.timer[N].reset() (on page 14-283)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

intervals	Table of delay intervals in seconds
N	Trigger timer number (1 to 4)

Details

Each time the timer is triggered after it is enabled, it uses the next delay period from the array. The default value is an array with one value of 10 µs.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be ≥ 50 µs.

Example

```
trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}

DelayList = trigger.timer[3].delaylist
for x = 1, table.getn(DelayList) do
   print(DelayList[x])
end
```

Set a delay list on trigger timer 3 with three delays (50 µs, 100 µs, and 150 µs).
Read the delay list on trigger timer 3.
Output:

```
5e-05
0.0001
0.00015
```

Also see

trigger.timer[N].reset() (on page 14-283)

trigger.timer[N].enable

This attribute enables the trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].enable
trigger.timer[N].enable = state
```

state	Disable the trigger timer: `trigger.OFF` Enable the trigger timer: `trigger.ON`
N	Trigger timer number (1 to 4)

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

Example

`trigger.timer[3].enable = trigger.ON`	Enable the trigger timer for timer 3.

Also see

None

trigger.timer[N].reset()

This function resets trigger timer settings to their default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`trigger.timer[N].reset()`

N	Trigger timer number (1 to 4)

Details

The `trigger.timer[N].reset()` function resets the following attributes to their default values:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].enable`
- `trigger.timer[N].start.fractionalseconds`
- `trigger.timer[N].start.generate`
- `trigger.timer[N].start.seconds`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

`trigger.timer[1].reset()`	Resets the attributes associated with timer 1 to their default values.

Also see

trigger.timer[N].count (on page 14-279)
trigger.timer[N].delay (on page 14-281)
trigger.timer[N].delaylist (on page 14-281)
trigger.timer[N].enable (on page 14-282)
trigger.timer[N].start.fractionalseconds (on page 14-284)
trigger.timer[N].start.generate (on page 14-285)
trigger.timer[N].start.overrun (on page 14-286)
trigger.timer[N].start.seconds (on page 14-286)
trigger.timer[N].start.stimulus (on page 14-287)

trigger.timer[N].start.fractionalseconds

This attribute configures the fractional seconds of an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	0

Usage

```
time = trigger.timer[N].start.fractionalseconds
trigger.timer[N].start.fractionalseconds = time
```

time	The time in fractional seconds (0 to <1 s)
N	Trigger timer number (1 to 4)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

`trigger.timer[1].start.fractionalseconds = 0.4`	Set the trigger timer to start in 0.4 s.

Also see

trigger.timer[N].start.generate (on page 14-285)

trigger.timer[N].start.generate

This attribute specifies when timer events are generated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].start.generate
trigger.timer[N].start.generate = state
```

state	Generate a timer event when the timer delay elapses: `trigger.OFF` Generate a timer event when the timer starts and when the delay elapses: `trigger.ON`
N	Trigger timer number (1 to 4)

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event `trigger.EVENT_TIMER`*N*.

Example

```
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY8
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON
```

Reset trigger timer 4.
Set trigger timer 4 to have a 0.5 s delay.
Set the stimulus for trigger timer 4 to be the notify 8 event.
Set the timer event to occur when the timer delay elapses.
Set the trigger timer 4 count to 20.
Enable trigger timer 4.

Also see

trigger.timer[N].reset() (on page 14-283)

trigger.timer[N].start.overrun

This attribute indicates if an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Trigger timer *N* reset	Not applicable	Not applicable

Usage

```
state = trigger.timer[N].start.overrun
```

state	The trigger overrun state (`true` or `false`)
N	Trigger timer number (1 to 4)

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

`print(trigger.timer[1].start.overrun)`	If an event was ignored, the output is `true`. If the event was not ignored, the output is `false`.

Also see

trigger.timer[N].reset() (on page 14-283)

trigger.timer[N].start.seconds

This attribute configures the seconds of an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	0 (0 s)

Usage

```
time = trigger.timer[N].start.seconds
trigger.timer[N].start.seconds = time
```

time	The time: 0 s to 2,147,483,647 s
N	Trigger timer number (1 to 4)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

```
trigger.timer[1].start.seconds = localnode.gettime() + 30
trigger.timer[1].enable = trigger.ON
```

Set the trigger timer to start 30 s from the time when the timer is enabled.

Also see

None

trigger.timer[N].start.stimulus

This attribute describes the event that starts the trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.timer[N].start.stimulus
trigger.timer[N].start.stimulus = event
```

event	The event that starts the trigger timer; see **Details**
N	Trigger timer number (1 to 4)

Details

Set the stimulus to any trigger event to start the timer when that event occurs.

Set the stimulus to none to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDER`*N*
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGIO`*N*
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LAN`*N*
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFY`*N*

Trigger events	
Event description	**Event constant**
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_IN
trigger.timer[1].delay = 3e-3
trigger.timer[1].start.stimulus = trigger.EVENT_DIGIO3
```

Set digital I/O line 3 to be a trigger input.
Set timer 1 to delay for 3 ms.
Set timer 1 to start the timer when an event is detected on digital I/O line 3.

Also see

None

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.timer[*N*].wait(*timeout*)

triggered	Trigger detection indication
N	Trigger timer number (1 to 4)
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.timer[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on timer 3.
If `false` is returned, no trigger was detected during the 10 s timeout.
If `true` is returned, a trigger was detected.

Also see

trigger.timer[N].clear() (on page 14-279)

trigger.tsplinkin[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkin[N].clear()
```

N	The trigger line (1 to 3) to clear

Details

The trigger event detector enters the detected state when an event is detected. When this command is sent, the instrument:

- Clears the trigger event detector
- Discards the history of the trigger line
- Clears the `trigger.tsplinkin[N].overrun` attribute

Example

```
tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[2].clear()
```

Clears the trigger event on TSP-Link line 2.

Also see

trigger.tsplinkin[N].overrun (on page 14-290)
tsplink.line[N].mode (on page 14-298)

trigger.tsplinkin[N].edge

This attribute indicates which trigger edge controls the trigger event detector for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.tsplinkin[N].edge
trigger.tsplinkin[N].edge = detectedEdge
```

detectedEdge	The trigger mode: ■ Detect falling-edge triggers as inputs: `trigger.EDGE_FALLING` ■ Detect rising-edge triggers as inputs: `trigger.EDGE_RISING` ■ Detect either falling or rising-edge triggers as inputs: `trigger.EDGE_EITHER`
N	The trigger line (1 to 3)

Details

When the edge is detected, the instrument asserts a TTL-low pulse for the output.

The output state of the I/O line is controlled by the trigger logic. The user-specified output state of the line is ignored.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[3].edge = trigger.EDGE_RISING
```

Sets synchronization line 3 to detect rising edge triggers as input.

Also see

digio.writeport() (on page 14-62)
tsplink.line[N].mode (on page 14-298)
tsplink.line[N].reset() (on page 14-298)

trigger.tsplinkin[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup TSP-Link line *N* clear	Not applicable	Not applicable

Usage

```
overrun = trigger.tsplinkin[N].overrun
```

overrun	Trigger overrun state
N	The trigger line (1 to 3)

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself.

It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun.

Example

`print(trigger.tsplinkin[1].overrun)`	If an event on line 1 was ignored, displays `true`; if no additional event occurred, displays `false`.

Also see

None

trigger.tsplinkin[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.tsplinkin[N].wait(timeout)
```

triggered	Trigger detection indication; set to one of the following values: ■ `true`: A trigger is detected during the timeout period ■ `false`: A trigger is not detected during the timeout period
N	The trigger line (1 to 3)
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events are detected since the last time this command or `trigger.tsplinkin[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
triggered = trigger.tsplinkin[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on TSP-Link line 3.
If `false` is returned, no trigger was detected during the 10-s timeout.
If `true` is returned, a trigger was detected.

Also see

trigger.tsplinkin[N].clear() (on page 14-289)
tsplink.line[N].mode (on page 14-298)

trigger.tsplinkout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkout[N].assert()
```

N	The trigger line (1 to 3)

Details

Initiates a trigger event and does not wait for completion. The set pulse width determines how long the trigger is asserted.

Example

```
tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[2].assert()
```

Asserts trigger on trigger line 2.

Also see

tsplink.line[N].mode (on page 14-298)

trigger.tsplinkout[N].logic

This attribute defines the trigger output with output logic for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.tsplinkout[N].logic
trigger.tsplinkout[N].logic = logicType
```

logicType	The output logic of the trigger generator: ■ Assert a TTL-high pulse for output: `trigger.LOGIC_POSITIVE` ■ Assert a TTL-low pulse for output: `trigger.LOGIC_NEGATIVE`
N	The trigger line (1 to 3)

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

The output state of the digital I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].logic = trigger.LOGIC_POSITIVE
```

Sets the trigger logic for synchronization line 3 to output a positive pulse.

Also see

trigger.tsplinkout[N].assert() (on page 14-291)
tsplink.line[N].mode (on page 14-298)

trigger.tsplinkout[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
width = trigger.tsplinkout[N].pulsewidth
trigger.tsplinkout[N].pulsewidth = width
```

width	The pulse width (0.0 to 100 ks)
N	The trigger line (1 to 3)

Details

Setting the pulse width to 0 asserts the trigger indefinitely.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].pulsewidth = 20e-6
```

Sets pulse width for trigger line 3 to 20 µs.

Also see

trigger.tsplinkout[N].assert() (on page 14-291)
trigger.tsplinkout[N].release() (on page 14-293)
tsplink.line[N].mode (on page 14-298)

trigger.tsplinkout[N].release()

This function releases a latched trigger on the given TSP-Link trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkout[N].release()
```

N	The trigger line (1 to 3)

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].release()
```

Releases trigger line 3.

Also see

trigger.tsplinkout[N].assert() (on page 14-291)
tsplink.line[N].mode (on page 14-298)

trigger.tsplinkout[N].stimulus

This attribute specifies the event that causes the synchronization line to assert a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.tsplinkout[N].stimulus
trigger.tsplinkout[N].stimulus = event
```

event	The event identifier for the triggering event (see **Details**)
N	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to `trigger.EVENT_NONE`.

Do not use this attribute when triggering under script control. Use `trigger.tsplinkout[N].assert()` instead.

The *event* parameters that you can use are described in the table below.

Trigger events	
Event description	**Event constant**
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
A command interface trigger: ■ **Any remote interface:** `*TRG` ■ **GPIB only:** GET bus command ■ **USB only:** A USBTMC TRIGGER message ■ **VXI-11:** VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
No trigger event	`trigger.EVENT_NONE`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`

Example

`print(trigger.tsplinkout[3].stimulus)`	Outputs the event that will start action on TSP-Link trigger line 3.

Also see

trigger.tsplinkout[N].assert() (on page 14-291)
tsplink.line[N].reset() (on page 14-298)

trigger.wait()

This function waits for a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

triggered = trigger.wait(*timeout*)

triggered	A trigger was detected during the timeout period: `true` No triggers were detected during the timeout period: `false`
timeout	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A USBTMC trigger message is received (USB only)
- A *TRG message is received

If one or more of these trigger events were previously detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

`triggered = trigger.wait(10)` `print(triggered)`	Waits up to 10 s for a trigger. If `false` is returned, no trigger was detected during the 10 s timeout. If `true` is returned, a trigger was detected.

Also see

trigger.clear() (on page 14-221)

tsplink.group

This attribute contains the group number of a TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

groupNumber	The group number of the TSP-Link node (0 to 64)

Details

To remove the node from all groups, set the attribute value to 0.

When the node is turned off, the group number for that node changes to 0.

The master node can be assigned to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

Example

`tsplink.group = 3`	Assign the instrument to TSP-Link group number 3.

Also see

Using groups to manage nodes on a TSP-Link system (on page 9-7)

tsplink.initialize()

This function initializes all instruments and enclosures in the TSP-Link system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.initialize()
tsplink.initialize()
tsplink.initialize(expectedNodes)
```

nodesFound	The number of nodes found on the system, including the node on which the command is running
expectedNodes	The number of nodes expected on the system (1 to 32)

Details

This function regenerates the system configuration information regarding the nodes connected to the TSP-Link system. You must initialize the system after making configuration changes. You need to initialize the system after you:

- Turn off power or reboot any instrument in the system
- Change node numbers on any instrument in the system
- Rearrange or disconnect the TSP-Link cable connections between instruments

If the only node on the TSP-Link network is the one running the command and *expectedNodes* is not provided, this function generates an error event. If you set *expectedNodes* to 1, the node is initialized.

If you include *expectedNodes*, if *nodesFound* is less than *expectedNodes*, an error event is generated.

NOTE

If any TSP-Link cabled node is powered off, initialize will fail.

Example

```
nodesFound = tsplink.initialize(2)
print("Nodes found = " .. nodesFound)
```

Perform a TSP-Link initialization and indicate how many nodes are found.

Example output if two nodes are found:

```
Nodes found = 2
```

Example output if fewer nodes are found and if `localnode.showevents = 7`:

```
1219, TSP-Link found fewer nodes than expected
Nodes found = 1
```

Also see

Initializing the TSP-Link system (on page 9-4)
localnode.showevents (on page 14-105)
tsplink.node (on page 14-301)
tsplink.state (on page 14-302)

tsplink.line[N].mode

This attribute defines the trigger operation of a TSP-Link line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	tsplink.MODE_DIGITAL_OPEN_DRAIN

Usage

```
mode = tsplink.line[N].mode
tsplink.line[N].mode = mode
```

mode	The trigger mode; see **Details**
N	The trigger line (1 to 3)

Details

This command defines whether or not the line is used as a digital or trigger control line and if it is an input or output.

The line mode can be set to the following options:

- TSP-Link digital open drain line: `tsplink.MODE_DIGITAL_OPEN_DRAIN`
- TSP-Link trigger open drain line: `tsplink.MODE_TRIGGER_OPEN_DRAIN`
- TSP-Link trigger synchronous master: `tsplink.MODE_SYNCHRONOUS_MASTER`
- TSP-Link trigger synchronous acceptor: `tsplink.MODE_SYNCHRONOUS_ACCEPTOR`

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
```

Sets the trigger mode for synchronization line 3 as a trigger open drain line.

Also see

trigger.tsplinkin[N].edge (on page 14-289)
trigger.tsplinkout[N].logic (on page 14-292)

tsplink.line[N].reset()

This function resets some of the TSP-Link trigger attributes to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.line[N].reset()
```

N	The trigger line (1 to 3)

Details

The `tsplink.line[N].reset()` function resets the following attributes to their default values:

- `tsplink.line[N].mode`
- `trigger.tsplinkin[N].edge`
- `trigger.tsplinkout[N].logic`
- `trigger.tsplinkout[N].pulsewidth`
- `trigger.tsplinkout[N].stimulus`

This also clears `trigger.tsplinkin[N].overrun`.

Example

`tsplink.line[3].reset()`	Resets TSP-Link trigger line 3 attributes to default values.

Also see

trigger.tsplinkin[N].edge (on page 14-289)
trigger.tsplinkin[N].overrun (on page 14-290)
trigger.tsplinkout[N].logic (on page 14-292)
trigger.tsplinkout[N].pulsewidth (on page 14-293)
trigger.tsplinkout[N].stimulus (on page 14-294)
tsplink.line[N].mode (on page 14-298)

tsplink.line[N].state

This attribute reads or writes the digital state of a TSP-Link synchronization line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	tsplink.STATE_HIGH

Usage

```
lineState = tsplink.line[N].state
tsplink.line[N].state = lineState
```

lineState	The state of the synchronization line: ■ Low: `tsplink.STATE_LOW` ■ High: `tsplink.STATE_HIGH`
N	The trigger line (1 to 3)

Details

Use `tsplink.writeport()` to write to all TSP-Link synchronization lines.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

`lineState = tsplink.line[3].state` `print(lineState)`	Assume line 3 is set high, and then the state is read. Output: `tsplink.STATE_HIGH`

Also see

tsplink.line[N].mode (on page 14-298)
tsplink.writeport() (on page 14-303)

tsplink.master

This attribute reads the node number assigned to the master node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

masterNodeNumber = tsplink.master

masterNodeNumber	The node number of the master node (1 to 63)

Details

This attribute returns the node number of the master in a set of instruments connected using TSP-Link.

Example

`LinkMaster = tsplink.master`	Store the TSP-Link master node number in a variable called `LinkMaster`.

Also see

tsplink.initialize() (on page 14-296)

tsplink.node

This attribute defines the node number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	2

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

nodeNumber	The node number of the instrument or enclosure (1 to 63)

Details

This command sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.reset()` from an earlier TSP-Link instrument or `tsplink.initialize()` is executed on any node in the system.

Each node connected to the TSP-Link system must be assigned a different node number.

Example

```
tsplink.node = 3
```

Sets the TSP-Link node for this instrument to number 3.

Also see

tsplink.initialize() (on page 14-296)
tsplink.state (on page 14-302)

tsplink.readport()

This function reads the TSP-Link trigger lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readport()
```

data	Numeric value that indicates which lines are set

Details

The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and the value of bit 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. This indicates that line 2 is high (1), and that the other two lines are low (0).

Example

`data = tsplink.readport()` `print(data)`	Reads state of all three TSP-Link lines. Assuming line 2 is set high, the output is: `2.000000e+00` (binary 010) The format of the output may vary depending on the ASCII precision setting.

Also see

Triggering using TSP-Link trigger lines (on page 9-6)
tsplink.line[N].state (on page 14-299)
tsplink.writeport() (on page 14-303)

tsplink.state

This attribute describes the TSP-Link online state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

state = `tsplink.state`

state	TSP-Link state (`online` or `offline`)

Details

When the instrument power is first turned on, the state is `offline`. After `tsplink.initialize()` or `tsplink.reset()` is successful, the state is `online`.

Example

`state = tsplink.state` `print(state)`	Read the state of the TSP-Link system. If it is online, the output is: `online`

Also see

tsplink.initialize() (on page 14-296)
tsplink.node (on page 14-301)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writeport(data)
```

data	Value to write to the port (0 to 7)

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other two lines are set low (0).

The `reset()` function does not affect the present states of the trigger lines.

Example

```
tsplink.writeport(3)
```

Sets the synchronization lines 1 and 2 high (binary 011).

Also see

tsplink.line[N].state (on page 14-299)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.clear(connectionID)
```

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

```
tspnet.write(testdevice, "print([[hello]])")
print(tspnet.readavailable(testdevice))



tspnet.clear(testdevice)
print(tspnet.readavailable(testdevice))
```

Write data to a device, then print how much is available.
Output:
`6.00000e+00`

Clear data and print how much data is available again.
Output:
`0.00000e+00`

Also see

tspnet.connect() (on page 14-304)
tspnet.readavailable() (on page 14-309)
tspnet.write() (on page 14-314)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
connectionID = tspnet.connect("ipAddress")
connectionID = tspnet.connect("ipAddress", portNumber, "initString")
```

connectionID	The connection ID to be used as a handle in all other `tspnet` function calls
ipAddress	IP address to which to connect in a string
portNumber	Port number (default 5025)
initString	Initialization string to send to *ipAddress*

Details

This command connects a device to another device through the LAN interface. If the *portNumber* is 23, the interface uses the Telnet protocol and sets appropriate termination characters to communicate with the device.

If a *portNumber* and *initString* are provided, it is assumed that the remote device is not TSP-enabled. The 2470 does not perform any extra processing, prompt handling, error handling, or sending of commands. In addition, the `tspnet.tsp.*` commands cannot be used on devices that are not TSP-enabled.

If neither a *portNumber* nor an *initString* is provided, the remote device is assumed to be a Keithley Instruments TSP-enabled device. Depending on the state of the `tspnet.tsp.abortonconnect` attribute, the 2470 sends an `abort` command to the remote device on connection.

The 2470 also enables TSP prompts on the remote device and event management. The 2470 places remote errors and events from the TSP-enabled device in its own event queue and prefaces these events with `Remote Error`, followed by an event description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show events by changing `localnode.showerrors` or `localnode.showevents` on the remote TSP-enabled device. If you do this, subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

```
instrumentID = tspnet.connect("192.0.2.1")
if instrumentID then
   -- Use instrumentID as needed here
   tspnet.disconnect(instrumentID)
end
```

Connect to a TSP-enabled device.

Example 2

```
instrumentID = tspnet.connect("192.0.2.1", 1394, "*rst\r\n")
if instrumentID then
   -- Use instrumentID as needed here
   tspnet.disconnect(instrumentID)
end
```

Connect to a device that is not TSP-enabled.

Also see

localnode.prompts (on page 14-102)
localnode.showevents (on page 14-105)
tspnet.tsp.abortonconnect (on page 14-312)
tspnet.disconnect() (on page 14-305)

tspnet.disconnect()

This function disconnects a specified TSP-Net session.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.disconnect(`*`connectionID`*`)`

`connectionID`	The connection ID returned from `tspnet.connect()`

Details

This function disconnects the two devices by closing the connection. The *`connectionID`* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

```
testID = tspnet.connect("192.0.2.0")
-- Use the connection
tspnet.disconnect(testID)
```

Create a TSP-Net session.

Close the session.

Also see

tspnet.connect() (on page 14-304)

tspnet.execute()

This function sends a command string to the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.execute("connectionID", "commandString")
value1 = tspnet.execute("connectionID", "commandString", formatString)
value1, value2 = tspnet.execute("connectionID", "commandString", formatString)
value1, ..., valueN = tspnet.execute("connectionID", "commandString", formatString)
```

connectionID	The connection ID returned from `tspnet.connect()`
commandString	The command to send to the remote device
value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The *N*th value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
formatString	Format string for the output

Details

This command sends a command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (`tspnet.termination()`). You can also specify a format string, which causes the command to wait for a response from the remote instrument. The 2470 decodes the response message according to the format specified in the format string and returns the message as return values from the function (see `tspnet.read()` for format specifiers).

When this command is sent to a TSP-enabled instrument, the 2470 suspends operation until a timeout error is generated or until the instrument responds. The TSP prompt from the remote instrument is read and discarded. The 2470 places any remotely generated errors and events into its event queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

```
tspnet.execute(instrumentID, "runScript()")
```

Command the remote device to run a script named `runScript`.

Example 2

```
tspnet.timeout = 5
id_instr = tspnet.connect("192.0.2.23", 23, "*rst\r\n")
tspnet.termination(id_instr, tspnet.TERM_CRLF)
tspnet.execute(id_instr, "*idn?")
print("tspnet.execute returns:", tspnet.read(id_instr))
```

Print the `*idn?` string from the remote device.

Also see

tspnet.connect() (on page 14-304)
tspnet.read() (on page 14-308)
tspnet.termination() (on page 14-310)
tspnet.write() (on page 14-314)

tspnet.idn()

This function retrieves the response of the remote device to `*IDN?`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

idnString = tspnet.idn(*connectionID*)

idnString	The returned `*IDN?` string
connectionID	The connection ID returned from `tspnet.connect()`

Details

This function retrieves the response of the remote device to `*IDN?`.

Example

```
deviceID = tspnet.connect("192.0.2.1")
print(tspnet.idn(deviceID))
tspnet.disconnect(deviceID)
```

Assume the instrument is at IP address 192.0.2.1.
The output that is produced when you connect to the instrument and read the identification string may appear as:

```
KEITHLEY INSTRUMENTS,MODEL
    2470,00000170,1.1.0s
```

Also see

tspnet.connect() (on page 14-304)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valueN = tspnet.read(connectionID, formatString)
```

value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The nth value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
connectionID	The connection ID returned from `tspnet.connect()`
formatString	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following specifiers:

`%[width]s`	Read data until the specified length
`%[max width]t`	Read data until the specified length or until punctuation is found, whichever comes first
`%[max width]n`	Read data until a newline or carriage return
`%d`	Read a number (delimited by punctuation)

A maximum of 10 format specifiers can be used for a maximum of 10 return values.

If *formatString* is not provided, the command returns a string that contains the data until a new line is reached. If no data is available, the 2470 pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When the 2470 reads from a TSP-enabled remote instrument, the 2470 removes Test Script Processor (TSP®) prompts and places any errors or events it receives from the remote instrument into its own event queue. The 2470 prefaces events and errors from the remote device with `Remote Error`, followed by the event number and description.

Example

```
tspnet.write(deviceID, "*idn?\r\n")
print("write/read returns:", tspnet.read(deviceID))
```

Send the `"*idn?\r\n"` message to the instrument connected as `deviceID`.
Display the response that is read from `deviceID` (based on the `*idn?` message).

Also see

tspnet.connect() (on page 14-304)
tspnet.readavailable() (on page 14-309)
tspnet.timeout (on page 14-311)
tspnet.write() (on page 14-314)

tspnet.readavailable()

This function checks to see if data is available from the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
bytesAvailable = tspnet.readavailable(connectionID)
```

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from `tspnet.connect()`

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")

repeat bytes = tspnet.readavailable(ID) until bytes > 0

print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that will create data.
Wait for data to be available.

Also see

tspnet.connect() (on page 14-304)
tspnet.read() (on page 14-308)

tspnet.reset()

This function disconnects all TSP-Net sessions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.reset()
```

Details

This command disconnects all remote instruments connected through TSP-Net. For TSP-enabled devices, this causes any commands or scripts running remotely to be terminated.

Also see

None

tspnet.termination()

This function sets the device line termination sequence.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

type	The termination type: ■ `tspnet.TERM_LF` ■ `tspnet.TERM_CR` ■ `tspnet.TERM_CRLF` ■ `tspnet.TERM_LFCR`
connectionID	The connection ID returned from `tspnet.connect()`
termSequence	The termination sequence: ■ `tspnet.TERM_LF` ■ `tspnet.TERM_CR` ■ `tspnet.TERM_CRLF` ■ `tspnet.TERM_LFCR`

Details

This function sets and gets the termination character sequence that is used to indicate the end of a line for a TSP-Net connection.

Using the *termSequence* parameter sets the termination sequence. The present termination sequence is always returned.

For the *termSequence* parameter, use the same values listed in the table above for type. There are four possible combinations, all of which are made up of line feeds (LF or 0x10) and carriage returns (CR or 0x13). For TSP-enabled devices, the default is `tspnet.TERM_LF`. For devices that are not TSP-enabled, the default is `tspnet.TERM_CRLF`.

Example

`deviceID = tspnet.connect("192.0.2.1")`
`if deviceID then`
`  tspnet.termination(deviceID, tspnet.TERM_LF)`
`end`
	Sets termination type for IP address 192.0.2.1 to `TERM_LF`.

Also see

tspnet.connect() (on page 14-304)
tspnet.disconnect() (on page 14-305)

tspnet.timeout

This attribute sets the timeout value for the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	20.0 (20 s)

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

value	The timeout duration in seconds (1 ms to 30.0 s)

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands will wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution but is only accurate to the nearest 10 ms.

Example

```
tspnet.timeout = 2.0
```
Sets the timeout duration to 2 s.

Also see

tspnet.connect() (on page 14-304)
tspnet.execute() (on page 14-306)
tspnet.read() (on page 14-308)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.abort(connectionID)
```

connectionID	Integer value used as a handle for other `tspnet` commands

Details

This function is appropriate only for TSP-enabled instruments.

Sends an abort command to the remote instrument.

Example

`tspnet.tsp.abort(testConnection)`	Stops remote instrument execution on `testConnection`.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	■ Enable: `1` ■ Disable: `0`

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument will not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

`tspnet.tsp.abortonconnect = 0`	Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

tspnet.connect() (on page 14-304)

tspnet.tsp.rbtablecopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
table = tspnet.tsp.rbtablecopy(connectionID, "name")
table = tspnet.tsp.rbtablecopy(connectionID, "name", startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other `tspnet` commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The *startIndex* and *endIndex* parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

```
times =
      tspnet.tsp.rbtablecopy(testTspdevice,
   "testRemotebuffername.timestamps", 1, 3)
print(times)
```

Copy the specified timestamps table for items 1 through 3, then display the table.
Example output:

```
01/01/2015
    10:10:10.0000013,01/01/2015
    10:10:10.0000233,01/01/2015
    10:10:10.0000576
```

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.runscript(connectionID, "name", "script")
```

connectionID	Integer value used as an identifier for other `tspnet` commands
name	The name that is assigned to the script
script	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions later.

Example

```
tspnet.tsp.runscript(myConnection, "myTest",
"print([[start]]) for d = 1, 10 do print([[work]]) end print([[end]])")
```

Load and run a script entitled `myTest` on the TSP-enabled instrument connected with `myConnection`.

Also see

tspnet.execute() (on page 14-306)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.write(connectionID, "inputString")
```

connectionID	The connection ID returned from `tspnet.connect()`
inputString	The string to be written

Details

The `tspnet.write()` function sends *`inputString`* to the remote instrument. It does not wait for command completion on the remote instrument.

The 2470 sends *`inputString`* to the remote instrument exactly as indicated. The *`inputString`* must contain any necessary new lines, termination, or other syntax elements needed to complete properly.

Because `tspnet.write()` does not process output from the remote instrument, do not send commands that generate too much output without processing the output. This command can stop executing if there is too much unprocessed output from previous commands.

Example

`tspnet.write(myID, "runscript()\r\n")`	Commands the remote instrument to execute a command or script named `runscript()` on a remote device identified in the system as `myID`.

Also see

tspnet.connect() (on page 14-304)
tspnet.read() (on page 14-308)

upgrade.previous()

This function returns to a previous version of the 2470 firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
upgrade.previous()
```

Details

This function allows you to revert to an earlier version of the firmware.

When you send this function, the instrument searches the USB flash drive in the front-panel USB port for an upgrade file. If the file is found, the instrument performs the upgrade. An error is returned if an upgrade file is not found.

NOTE

Use this command with caution. Make sure your instrument can support the earlier version and that there are no compatibility issues. Check with Keithley Instruments before using this command if you have questions.

Also see

Upgrading the firmware (on page 10-3)
upgrade.unit() (on page 14-316)

upgrade.unit()

This function upgrades the 2470 firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
upgrade.unit()
```

Details

When `upgrade.unit()` is used, the firmware is only loaded if the version of the firmware is newer than the existing version. If the version is older or at the same revision level, it is not upgraded.

When you send this function, the instrument searches the USB flash drive in the front-panel USB port for an upgrade file. If the file is found, the instrument verifies that the file is a newer version. If the version is older or at the same revision level, it is not upgraded, although it does request a reboot. If it is a newer version, the instrument performs the upgrade. An error event message is returned if no upgrade file is found.

Also see

upgrade.previous() (on page 14-315)
Upgrading the firmware (on page 10-3)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.add("name", "value")
```

name	The name of the string; the key of the key-value pair
value	The string to associate with *name*; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

Example

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.catalog() do
   print(name .. " = " ..
      userstring.get(name))
end
```

Stores user-defined strings in nonvolatile memory and recalls them from the instrument using a for loop.
Example output:

```
assetnumber = 236
contact = John Doe
product = Widgets
```

Also see

userstring.catalog() (on page 14-317)
userstring.delete() (on page 14-318)
userstring.get() (on page 14-318)

userstring.catalog()

This function creates an iterator for the user-defined string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do body end
```

name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The catalog provides access for user-defined string pairs, allowing you to manipulate all the key-value pairs in nonvolatile memory. The entries are enumerated in no particular order.

Example 1

```
for name in userstring.catalog() do
   userstring.delete(name)
end
```

Deletes all user-defined strings in nonvolatile memory.

Example 2

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.catalog() do
   print(name .. " = " ..
      userstring.get(name))
end
```

Prints all `userstring` key-value pairs.
Output:

```
product = Widgets
assetnumber = 236
contact = John Doe
```

Notice the key-value pairs are not listed in the order they were added.

Also see

userstring.add() (on page 14-316)
userstring.delete() (on page 14-318)
userstring.get() (on page 14-318)

userstring.delete()

This function deletes a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.delete("name")
```

name	The name (key) of the key-value pair of the user-defined string to delete

Details

This function deletes the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.delete("assetnumber")
userstring.delete("product")
userstring.delete("contact")
```

Deletes the user-defined strings associated with the `assetnumber`, `product`, and `contact` names.

Also see

userstring.add() (on page 14-316)
userstring.catalog() (on page 14-317)
userstring.get() (on page 14-318)

userstring.get()

This function retrieves a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value = userstring.get("name")
```

value	The value of the user-defined string key-value pair
name	The name (key) of the user-defined string

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.add("assetnumber", "236")
value = userstring.get("assetnumber")
print(value)
```

Create the user-defined string `assetnumber`, set to a value of 236.
Read the value associated with the user-defined string named `assetnumber`.
Store it in a variable called `value`, then print the variable `value`.
Output:
236

Also see

userstring.add() (on page 14-316)
userstring.catalog() (on page 14-317)
userstring.delete() (on page 14-318)

waitcomplete()

This function waits for all previously started overlapped commands to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
waitcomplete()
waitcomplete(group)
```

group	Specifies which TSP-Link group on which to wait

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `waitcomplete()` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. This command is not needed for sequential commands.

A group number may only be specified when this node is the master node.

If no *group* is specified, the local group is used.

If zero (0) is specified for the *group*, this function waits for all nodes in the system.

NOTE

Any nodes that are not assigned to a group (group number is 0) are part of the master node's group.

Example 1

`waitcomplete()`	Waits for all nodes in the local group.

Example 2

`waitcomplete(G)`	Waits for all nodes in group G.

Example 3

`waitcomplete(0)`	Waits for all nodes on the TSP-Link network.

Also see

None

Section 15

Common commands

In this section:

Introduction 15-1
*CLS 15-2
*ESE 15-2
*ESR? 15-4
*IDN? 15-5
*LANG 15-5
*OPC 15-6
*RST 15-7
*SRE 15-7
*STB? 15-8
*TRG 15-9
*TST? 15-9
*WAI 15-10

Introduction

This section describes the general remote interface commands and common commands. Note that although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the 2470 does not strictly conform to that standard.

The general remote interface commands are commands that have the same general meaning, regardless of the instrument you use them with (for example, DCL always clears the GPIB interface and returns it to a known state).

The common commands perform operations such as reset, wait-to-continue, and status.

Common commands always begin with an asterisk (*) and may include one or more parameters. The command keyword is separated from the first parameter by a blank space.

If you are using a SCPI remote interface, the commands can be combined. Use a semicolon (;) to separate multiple commands, as shown below:

```
*RST; *CLS; *ESE 32; *OPC?
```

Although the commands in this section are shown in uppercase, they are not case sensitive (you can use either uppercase or lowercase).

If you are using the TSP remote interface, each command must be sent in a separate message.

NOTE

If you are using the TSP remote interface, note that the common commands and general bus commands cannot be used in scripts.

*CLS

This command clears the event registers and queues.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*CLS
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It also clears the event log. It does not affect the Questionable Event Enable or Operation Event Enable registers.

This is the equivalent of sending the SCPI commands `:STATus:CLEar` and `:SYStem:CLEar` or the TSP commands `status.clear()` and `eventlog.clear()`.

To reset all the bits of the Standard Event Enable Register, send the command:

```
*ESE 0
```

Also see

*ESE (on page 15-2)
:STATus:PRESet (on page 12-100)
status.preset() (on page 14-208)

*ESE

This command sets and queries bits in the Status Enable register of the Standard Event Register.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	See **Details**

Usage

```
*ESE <n>
*ESE?
```

<n>	The value of the Status Enable register of the Standard Event Register (0 to 255)

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or the value of the bit as the <n> parameter.

If you are using TSP, you can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B2, set *standardRegister* to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

If you are using SCPI, you can only set the bit as a numeric value. When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-108) function.
1	2	Not used	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

`*ESE 129`	`*ESE 129` sets the Status Enable register of the Standard Event Register to binary `10000001`, which enables the PON and OPC bits.

Also see

*CLS (on page 15-2)
Standard Event Register (on page 16-3)
Status model (on page 16-1)

*ESR?

This command reads and clears the contents of the Standard Event Status Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
*ESR?
```

Details

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-108) function.
1	2	Not used	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

`*ESR?`	Example output: `128` Shows that the Standard Event Status Register contains binary `10000000`, which indicates that the instrument was rebooted since the last time this register was read.

Also see

Status model (on page 16-1)

*IDN?

This command retrieves the identification string of the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
*IDN?
```

Details

The identification string includes the manufacturer, model number, serial number, and firmware revision of the instrument. The string is formatted as follows:

```
KEITHLEY INSTRUMENTS,MODEL nnnn,xxxxxxxx,yyyyyy
```

Where:

- nnnn is the model number
- xxxxxxxx is the serial number
- yyyyyy is the firmware revision level

Example

`*IDN?`	Output: `KEITHLEY INSTRUMENTS,MODEL 2470,01234567,1.0.0i`

Also see

System information (on page 2-37)

*LANG

This command determines which command set is used by the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	SCPI

Usage

```
*LANG <commandSet>
*LANG?
```

`<commandSet>`	The command set to be used: ■ `TSP` ■ `SCPI`

Details

The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

Example

`*LANG TSP` `*LANG?`	Set the command set to TSP. Verify setting by sending the command set query. Output: TSP The TSP command set is in use.

Also see

Status model (on page 16-1)

*OPC

This command sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
*OPC
*OPC?
```

Details

When `*OPC` is sent, the OPC bit (bit 0) in the Status Event Status Register is set after all pending command operations have been executed. After all programmed operations are complete, the instrument returns to idle, at which time all pending commands (including `*OPC` and `*OPC?`) are executed. After the last pending command is executed, the OPC bit is set or an ASCII “1” is placed in the Output Queue.

When the trigger model is executing, most sent commands are not executed. If a command cannot be processed, an error event message is generated in the event log.

Also see

:INITiate[:IMMediate] (on page 12-145)
opc() (on page 14-108)

*RST

This command resets the instrument settings to their default values and clears the reading buffers.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*RST
```

Details

Returns the instrument to default settings, cancels all pending commands, and cancels the response to any previously received `*OPC` and `*OPC?` commands.

Also see

reset() (on page 14-114)

*SRE

This command sets or clears the bits of the Service Request Enable Register.

Type	Affected by	Where saved	Default value
Command and query	:STATus:PRESet status.preset()	Not applicable	0

Usage

```
*SRE <n>
*SRE?
```

`<n>`	Clear the Status Request Enable Register: `0` Set the instrument for an SRQ interrupt: `32`

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	`status.MSB`	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	`status.EAV`	An error or status message is present in the Error Queue.
3	8	`status.QSB`	An enabled event in the Questionable Status Register has occurred.
4	16	`status.MAV`	A response message is present in the Output Queue.
5	32	`status.ESB`	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	`status.OSB`	An enabled event in the Operation Status Register has occurred.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

`*SRE 0`	Clear the bits of the Status Request Enable Register.

Also see

Understanding bit settings (on page 16-15)

*STB?

This command gets the status byte of the instrument without clearing the request service bit.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
*STB?
```

Details

This command is similar to a serial poll, but it is processed like any other instrument command.

The `*STB?` command returns the same result as a serial poll, but the master summary bit (MSB) is not cleared if a serial poll has occurred. The MSB is not cleared until all other bits feeding into the MSB are cleared.

Example

`*STB?`	Queries the status byte.

Also see

None

*TRG

This command generates a trigger event from a remote command interface.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*TRG
```

Details

Use the `*TRG` command to generate a trigger event.

If you are using the SCPI command set, this command generates the `COMMand` event. If you are using the TSP command set, this command generates the `trigger.EVENT_COMMAND` event. You can use this constant as the stimulus of any trigger object, which causes that trigger object to respond to the trigger events generated by `*TRG`. See Using trigger events to start actions in the trigger model (on page 8-52).

Also see

:INITiate[:IMMediate] (on page 12-145)

*TST?

This command is accepted and returns 0. A self-test is not actually performed.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	0

Usage

```
*TST?
```

Also see

None

*WAI

This command postpones the execution of subsequent commands until all previous overlapped commands are finished.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*WAI
```

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `*WAI` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. The `*WAI` command is not needed for sequential commands. Typically, this command is sent after the initiate trigger model command.

Also see

:INITiate[:IMMediate] (on page 12-145)
waitcomplete() (on page 14-319)

Section 16

Status model

In this section:

Overview 16-1
Serial polling and SRQ 16-13
Programming enable registers 16-14
Reading the registers 16-14
Understanding bit settings 16-15
Clearing registers 16-16
Status model programming examples 16-17

Overview

The status model consists of status register sets and queues. You can monitor the status model to view instrument events and configure the status model to control the events.

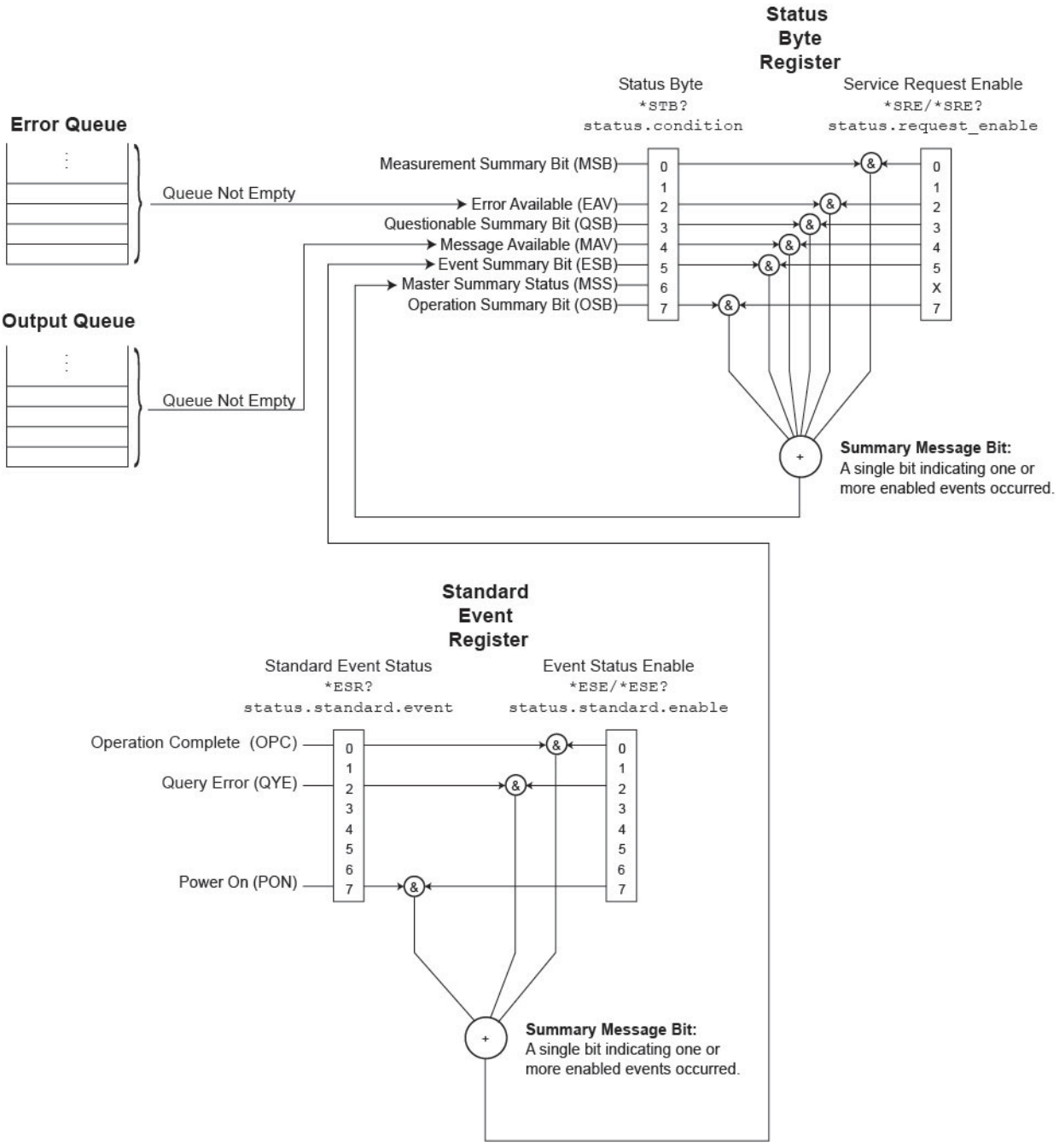
As you work with the status model, be aware that the result applies to the Status Byte Register. All the status register sets and queues flow into the Status Byte Register. Your test program can read this register to determine if a service request (SRQ) has occurred, and if so, which event caused it.

The Status Byte Register, register sets, and queues include:

- Standard Event Register
- Questionable Event Register
- Operation Event Register
- Output Queue
- Error Queue

The relationship between the Status Byte Register, Standard Event Register, event queue, and output queue is shown in the Non-programmable status registers diagram (on page 16-2). The relationship between the Status Byte Register, Questionable Event Register, and the Operation Event Register is shown in the Programmable status registers diagram (on page 16-5).

Figure 157: Non-programmable status registers diagram

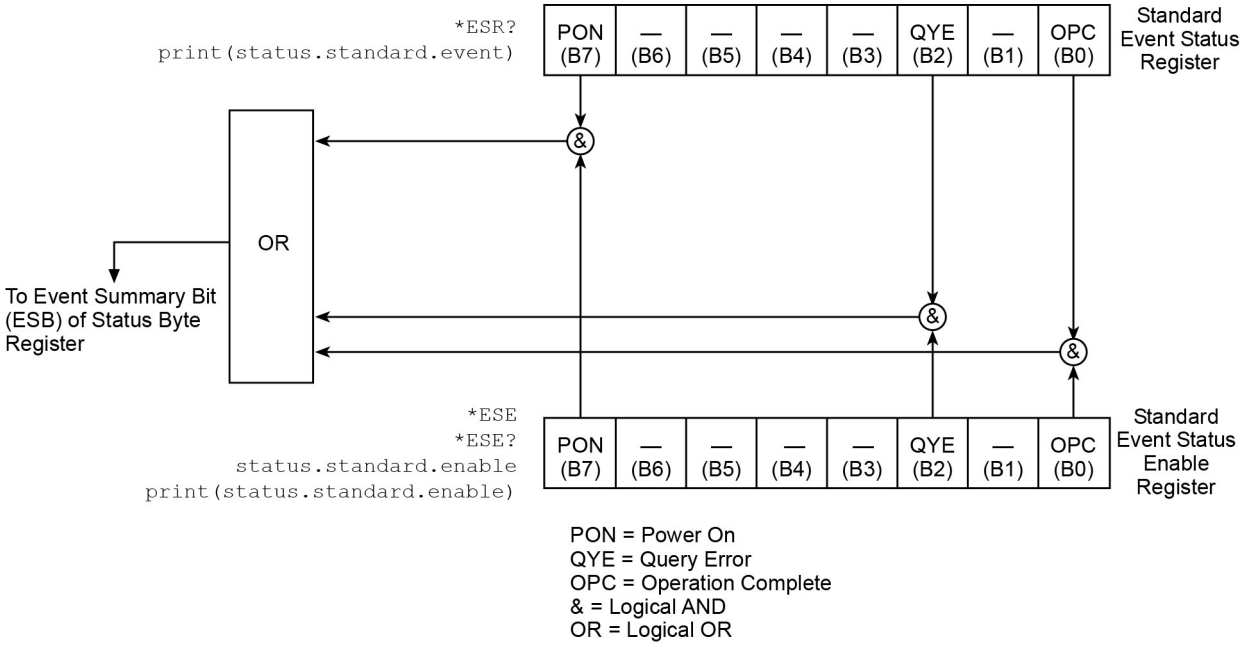


Standard Event Register

The Standard Event Register set includes two 8-bit registers:

- **Standard Event Status Register:** Reports when a predefined event has occurred. The register latches the event and the corresponding bit remains set until it is cleared by a read.
- **Standard Event Status Enable Register:** You can enable or disable bits in this register. This allows the predefined event (from the Standard Event Status Register) to set the ESB of the Status Byte Register.

Figure 158: 2470 Standard Event Register



Bit	When set, indicates the following has occurred:
0	Operation complete: All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-108) function.
1	Not used.
2	Query error: Attempt to read data from an empty Output Queue.
3	Not used.
4	Not used.
5	Not used.
6	Not used.
7	Power-on: The instrument has been turned off and turned back on since the last time this register was read.

You can use the following commands to read and set bits in the Standard Event Register.

Description	SCPI command	TSP command
Read the Standard Event Status Register	*ESR? (on page 15-4)	status.standard.event (on page 14-215)
Set or read the OR bits in the Standard Event Status Enable Register	*ESE (on page 15-2) ESE?	status.standard.enable (on page 14-213)

Programmable status register sets

You can program the registers in the Questionable Event Register and Operation Event Register sets.

These event registers contain bits that identify the state of an instrument condition or event. They also contain bits that determine if those events are sent to the Status Byte Register. You can enable the events, which causes the associated bit to be set in the Status Byte Register.

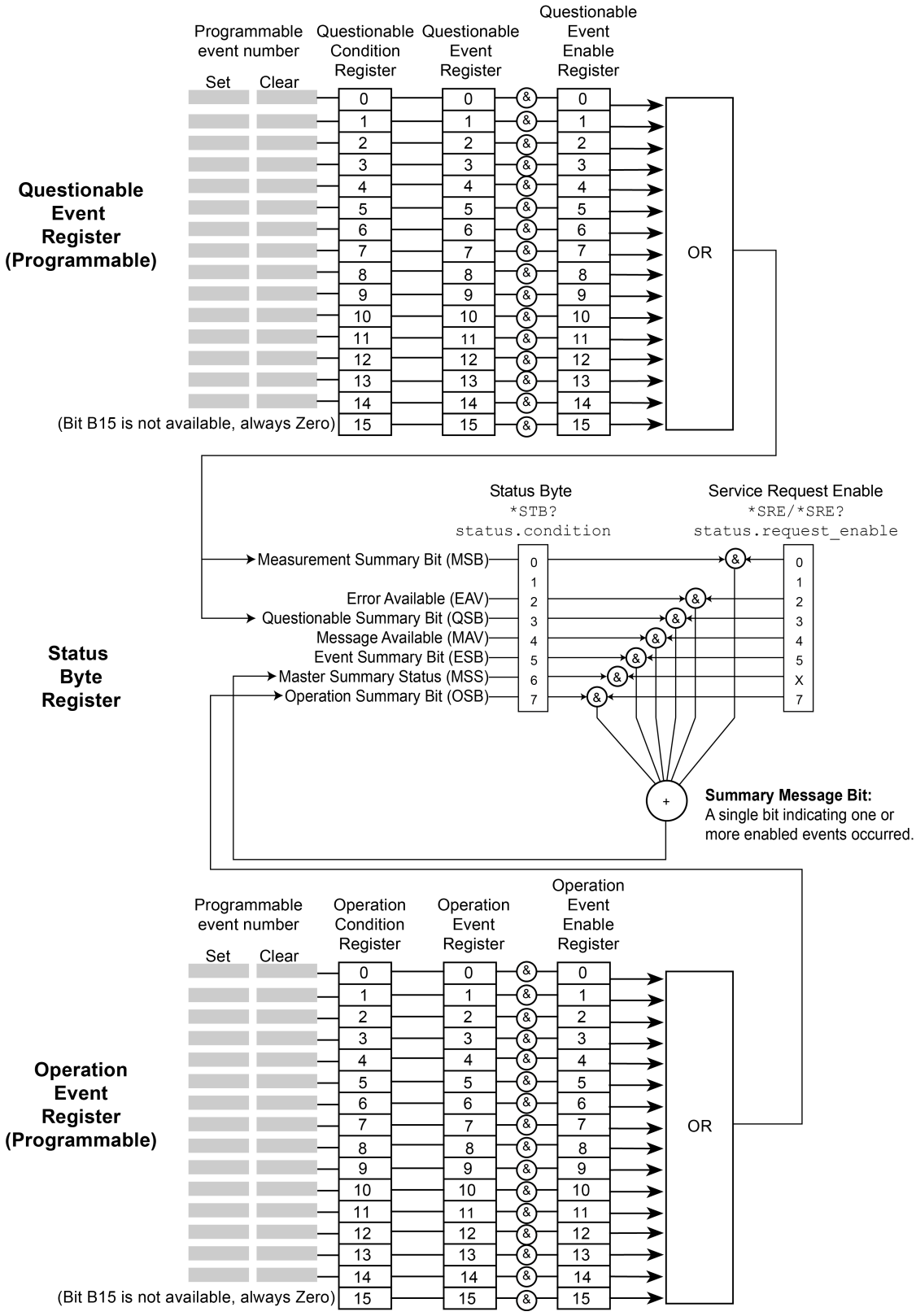
The Questionable and Operation Event Registers are identical except that they set different bits in the Status Byte Register. The Questionable Event Registers set the MSB and QSM bits. The Operation Event Registers set the OSB bit.

Each 16-bit register set includes the following registers:

- **Condition:** A read-only register that is constantly updated to reflect the present operating conditions of the instrument. You can determine which events set or clear the bits.
- **Event:** A read-only register that sets a bit to 1 when an applicable event occurs. The bit remains at 1 until the register is reset. This register is reset when power is cycled, when a `*CLS` command is sent, or when the register is read. You can determine which events set the bits.
- **Event enable:** A read-write register that determines which events set the summary bit in the Status Byte Register. For example, if a bit is a 1 in the event register and the corresponding bit is a 1 in the Event Enable Register, bits in the Status Byte Register are set. If the event enable bit is set in the Questionable Event Registers, the event sets the MSB and QSM bits in the Status Byte Register. If the event enable bit is set in the Operation Event Registers, the event sets the OSB bit in the Status Byte Register.

When the instrument is powered on, all bits in the Questionable Event and Operation Event Registers are set to 0.

Figure 159: Programmable status registers diagram



Questionable Event Register

You can program the bits in the Questionable Event Register to be cleared or set when an event occurs.

When an enabled Questionable Event Register bit is set (because the enabled event occurs), the corresponding bit B0 (MSB) and Bit B3 (QSB) of the Status Byte Register is set. The corresponding Questionable Event Register Condition Register reflects the present status of the instrument, so it is set while the event occurs.

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Understanding bit settings (on page 16-15).

You can use the following commands to read and set bits in the Questionable Event Register.

Description	SCPI command	TSP command
Read the Questionable Condition Register	:STATus:QUEStionable:CONDition? (on page 12-101)	status.questionable.condition (on page 14-209)
Set or read the contents of the Questionable Event Enable Register	:STATus:QUEStionable:ENABle (on page 12-101)	status.questionable.enable (on page 14-209)
Read the Questionable Event Register	:STATus:QUEStionable[:EVENt]? (on page 12-103)	status.questionable.event (on page 14-210)
Request the mapped set event and mapped clear event status for a bit in the Questionable Event Register	:STATus:QUEStionable:MAP (on page 12-102)	status.questionable.getmap() (on page 14-211)
Map event to a bit in the Questionable Event Register	:STATus:QUEStionable:MAP (on page 12-102)	status.questionable.setmap() (on page 14-211)

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register. See Event numbers (on page 16-9) for information about event numbers. You can use a copy of the following table to record settings for your instrument.

Bit	Decimal value	Set Event	Clear Event	When set, indicates the following has occurred:
0	1			
1	2			
2	4			
3	8			
4	16			
5	32			
6	64			
7	128			
8	256			
9	512			
10	1024			
11	2048			
12	4096			
13	8192			
14	16 384			

Operation Event Register

You can program the bits in the Operation Condition and Operation Event Status Registers to be cleared or set when an event occurs.

When an enabled Operation Event Register bit is set (because the enabled event occurs), the corresponding bit B7 (OSB) of the Status Byte Register is set. The corresponding Operation Event Register Condition Register reflects the present status of the instrument, so it will be set while the event occurs.

You can use the following commands to read and set bits in the Operation Event Register.

Description	SCPI command	TSP command
Read the Operation Condition Register	:STATus:OPERation:CONDition? (on page 12-98)	status.operation.condition (on page 14-205)
Set or read the contents of the Operation Event Enable Register	:STATus:OPERation:ENABle (on page 12-98)	status.operation.enable (on page 14-205)
Read the Operation Event Register	:STATus:OPERation[:EVENt]? (on page 12-99)	status.operation.event (on page 14-206)
Request the mapped set event and mapped clear event status for a bit in the Operation Event Registers	:STATus:OPERation:MAP (on page 12-99)	status.operation.getmap() (on page 14-207)
Map events to bit in the Operation Event Register	:STATus:OPERation:MAP (on page 12-99)	status.operation.setmap() (on page 14-207)

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register. See Event numbers (on page 16-9) for information about event numbers. You can use a copy of the following table to record settings for your instrument.

Bit	Decimal value	Set Event	Clear Event	When set, indicates the following has occurred:
0	1			
1	2			
2	4			
3	8			
4	16			
5	32			
6	64			
7	128			
8	256			
9	512			
10	1024			
11	2048			
12	4096			
13	8192			
14	16 384			

Mapping events to bits

To program the Questionable and Operation Event Registers, you map events to specific bits in the register. This causes a bit in the condition and event registers to be set (or cleared) when the specified event occurs. You can map events to bits B0 through B14 (bit B15 is always set to zero).

When you have a mapped-set event, the bits in the corresponding condition register and event register are set when the mapped-set event is detected. The bits remain at 1 until the event register is read or the status model is reset.

When you have a mapped-clear event, the bit in the condition register is cleared to 0 when the event is detected.

You can map any event to any bit in these registers. An event is the number that accompanies an error, warning, or informational message that is reported in the event log. For example, for the event code "Error -221, Settings Conflict," the event is –221. Note that some informational messages do not have a related event number, so they cannot be mapped to a register.

You do not need to map clear events to generate SRQs. However, if you want to read the condition register to report status, you must map both a set event and a clear event. If no clear event is mapped, the bits are cleared only when the instrument power is turned off and turned on.

You can use the following SCPI commands to read and map events to bits in the programmable registers:

- :STATus:OPERation:MAP (on page 12-99)
  This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.
- :STATus:QUEStionable:MAP (on page 12-102)
  This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.

You can use the following TSP commands to read and map events to bits in the programmable registers:

- status.operation.getmap() (on page 14-207)
  This command reads the mapped set and clear status for the specified operation event bit.
- status.operation.setmap() (on page 14-207)
  This command maps the set and clear events to a specified operation event register bit.
- status.questionable.getmap() (on page 14-211)
  This command reads the mapped set and clear status for the specified questionable event bit.
- status.questionable.setmap() (on page 14-211)
  This command maps the set and clear events to a specified questionable event register bit.

You can map any event that appears with a number in the event queue to any available bit in a programmable register. The programmable registers and their relationships to the Status Byte Register are shown in the Programmable status registers diagram (on page 16-5). The following example event queue log entries contain actual events that can be mapped to a status model bit.

```
2731 Trigger Model Initiated "Trigger model #1 has been initiated"
2732 Trigger Model Idle  "Trigger model #1 has been idled"
4917 Reading buffer cleared "Reading buffer <buffer name> is  0% filled"
4918 Reading buffer full "Reading buffer <buffer name> is  100% filled"
5080 SMU Source Limit Tripped "Source limiting is active on output"
5081 SMU Source Limit Cleared "Source limiting is no longer necessary and output is
   normal"
```

See Using the event log (on page 3-50) for additional information on finding events.

Event numbers

You can decide what events to use to set and clear the bits in the Operation Event Register and Questionable Event Register. The following table lists the event numbers and descriptions.

NOTE

This table lists frequently used event numbers. This table does not list all event numbers.

Event number	Event description for user
2728	Trigger model was running, but then aborted by user action.
2730	The trigger model was running but has failed and returned to the idle possibly due to an error in settings.
2731	Trigger model was idle but is now running.
2732	Trigger model was running, but completed successfully, and is now in idle.
2733	The trigger model has ended in specified block number due to the specified condition.
2734	A trigger model block has logged a user-defined informational event message.
2735	A trigger model block has logged a user-defined informational event message.
2736	A trigger model block has logged a user-defined informational event message.
2737	A trigger model block has logged a user-defined informational event message.
2738	A trigger model block has logged a user-defined warning event message.
2739	A trigger model block has logged a user-defined warning event message.
2740	A trigger model block has logged a user-defined warning event message.
2741	A trigger model block has logged a user-defined warning event message.
2742	A trigger model block has logged a user-defined error event message.
2743	A trigger model block has logged a user-defined error event message.
2744	A trigger model block has logged a user-defined error event message.
2745	A trigger model block has logged a user-defined error event message.
2771	The instrument is nearing an internal temperature where corrective action will be necessary. The next limit will automatically turn off source outputs or limit other functionality.
2774	The instrument is experiencing an internal temperature malfunction since it is unable to supply readings and requires immediate service.
2775	An internal malfunction had cleared itself and the instrument is operating normally.
2776	The instrument is operating at normal temperature levels and fully functional.

Event number	Event description for user
2777	The instrument temperature has reached a level in which source output and other functionality will not be available.
2778	The instrument temperature has fallen to a level in which source output and other functionality is now available.
4917	The specified reading buffer is cleared (empty).
4918	The specified reading buffer is full.
5080	The source output has gone over the source limit level and is being limited.
5081	The source output has returned below the source limit level and is no longer being limited.

Status Byte Register

The Status Byte Register monitors the registers and queues in the status model and generates service requests (SRQs).

When bits are set in the status model registers and queues, they generate summary messages that set or clear bits of the Status Byte Register. You can enable these bits to generate an SRQ.

Service requests (SRQs) instruct the controller that the instrument needs attention or that some event has occurred. When the controller receives an SRQ, the controller can interrupt existing tasks to perform tasks that address the request for service.

For example, you might program your instrument to send an SRQ when a specific instrument error event occurs. To do this, you set the Status Request Enable bit 2 (EAV). In this example, the following actions occur:

- The error event occurs.
- The error event is logged in the Error Queue.
- The Error Queue sets the EAV bit of the Status Byte Register.
- The EAV bits are summed.
- The RQS bit of the Status Byte Register is set.
- On a GPIB system, the SRQ line is asserted. On a VXI-11 or USB system, an SRQ event is generated.

For an example of this, see the example code provided in SRQ on error (on page 16-21).

The summary messages from the status registers and queues set or clear the appropriate bits (B0, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are solely dependent on the summary messages (0 or 1). For example, if the Standard Event Register is read, its register will clear. As a result, its summary message resets to 0, which in turn resets the ESB bit in the Status Byte Register.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status (MSS) bit.

When using the GPIB, USB, or VXI-11 serial poll sequence of the 2470 to get the status byte (serial poll byte), bit B6 is the RQS bit. See Serial polling and SRQ (on page 16-13) for details on using the serial poll sequence.

When using the `*STB?` common command or `status.condition` command to read the status byte, bit B6 is the MSS bit.

To reset the bits of the Service Request Enable Register to 0, use 0 as the parameter value for the command (for example, `*SRE 0` or `status.request_enable = 0`).

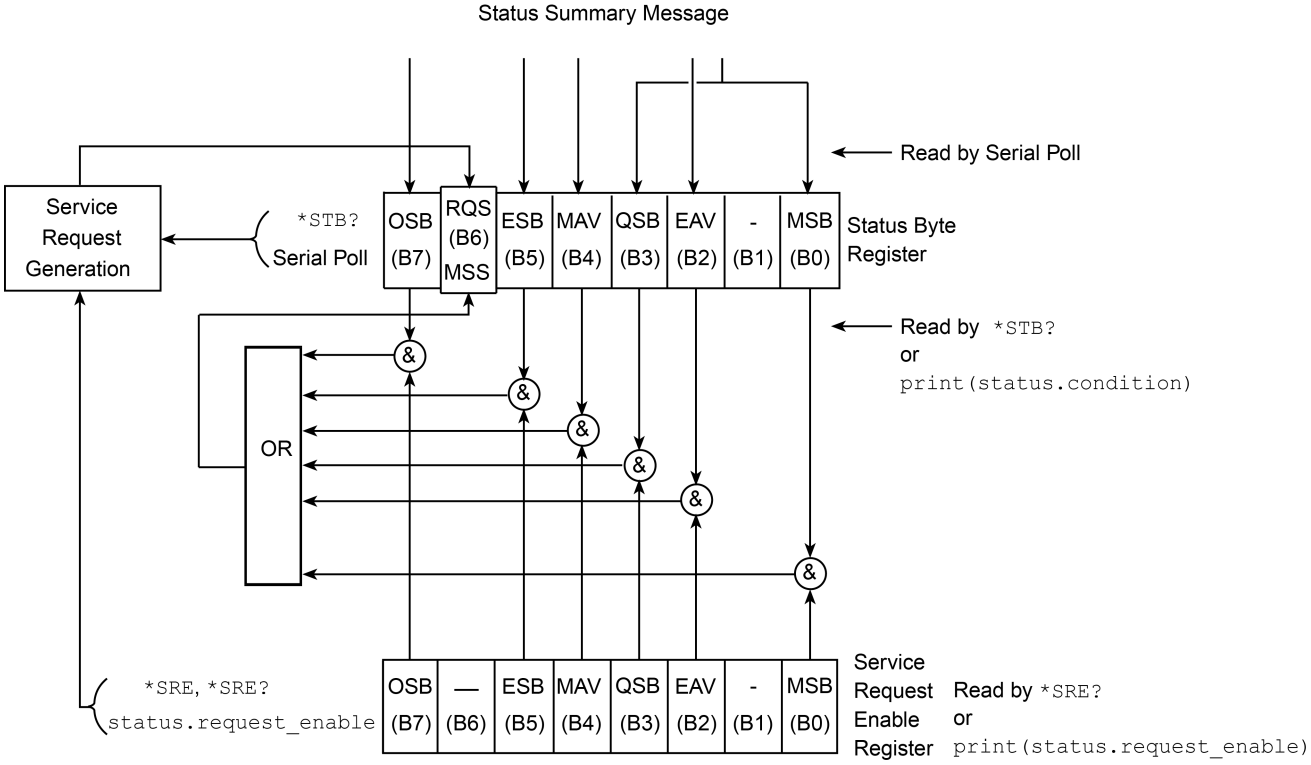
You can read and set which bits to AND in the Status Byte Register using the following commands.

Description	SCPI command	TSP command
Read the Status Byte Register	*STB? (on page 15-8)	status.condition (on page 14-204)
Read the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-212)
Enable bits in the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-212)

Status Byte Register diagram

The Status Byte Register consists of two 8-bit registers that control service requests, the Status Byte Register and the Service Request Enable Register. These registers are shown in the following figure.

Figure 160: 2470 Status Byte Register



The bits in the Status Byte Register are described in the following table.

Bit	Decimal value	Bit name	When set, indicates the following has occurred:
0	1	Measurement summary Bit (MSB)	An enabled questionable event
1	2	Not used	Not applicable
2	4	Error available (EAV)	An error is present in the error queue (warning and information messages do not affect this bit)
3	8	Questionable summary bit (QSB)	An enabled questionable event
4	16	Message available (MAV)	A response message is present in the output queue
5	32	Event summary bit (ESB)	An enabled standard event
6	64	Request for service (RQS)/Master summary status (MSS)	An enabled summary bit of the Status Byte Register is set; depending on how it is used, this is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit
7	128	Operation summary bit (OSB)	An enabled operation event

Service Request Enable Register

This register is programmed by the user and is used to enable or disable the setting of bit B6 (RQS/MSS) by the Status Summary Message bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in the Status Byte Register (on page 16-10) topic, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic "1" output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

You can set or clear the individual bits of the Service Request Enable Register by using the `*SRE` common command or `status.request_enable`. To read the Service Request Enable Register, use the `*SRE?` query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a `*SRE 0` or `status.request_enable = 0` is sent). You can program and read the SRQ Enable Register using the following commands.

Description	SCPI command	TSP command
Read the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-212)
Enable bits in the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-212)

Queues

The instrument includes an Output Queue and an Error Queue. The Output Queue holds messages from readings and responses. The Error Queue holds error event messages from the event log. Both are first-in, first-out (FIFO) registers.

Output Queue

The output queue holds response messages to SCPI query and TSP `print()` commands.

When data is placed in the Output Queue, the Message Available (MAV) bit in the Status Byte Register is set. The bit is cleared when the Output Queue is empty.

To clear data from the Output Queue, read the messages. To read a message from the Output Queue, address the instrument to talk after the appropriate query is sent.

Error Queue

The Error Queue holds any error events that are posted in the event log. When an error event occurs, it is posted to the Error Queue, which sets the Error Available (EAV) bit in the Status Byte Register.

The instrument clears error event messages from the event log when it retrieves the event log. When the error event messages are cleared from the event log, the EAV bit in the Status Byte Register is cleared.

You can clear the Error Queue by sending the common command `*CLS` or the TSP command `status.clear()`. Note that `status.clear()` also clears all event registers.

For information regarding the event log, see Using the event log (on page 3-50).

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ occurred and what caused it. If an SRQ occurred, the program can, for example, branch to an appropriate subroutine that will service the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

The serial poll does not clear the low-level registers that caused the SRQ to occur. You must clear the low-level registers explicitly. Refer to Clearing registers (on page 16-16).

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit remains set until all enabled Status Byte Register summary bits are reset.

For information on serial polling on a GPIB system, see SPE, SPD (on page 2-13).

Programming enable registers

You can program the bits in the enable registers of the Status Model registers.

When you program an enable register bit to 0, no action occurs if the bits in the corresponding registers are set (1).

When you program an enable register bit to 1, if the bits in the corresponding registers are set (1), the AND condition occurs and a bit in the Status Byte Register is set to (1).

You must program all bits in an enable register at the same time. This means you need to determine what each bit value in the register will be, then add them together to determine the value of all the bits in the register. See Understanding bit settings (on page 16-15) for more information on determining the value of the bits in the registers.

For example, you might want to enable the Standard Event Register to set the ESB bit in the Status Byte Register whenever an operation complete occurs or whenever an operation did not execute properly because of an internal condition. To do this, you need to set bits 0 and 3 of the Standard Event Register to 1. These bits have decimal values of 1 and 8, so to set both bits to 1, you set the register to 9.

Using SCPI:

Send the command:

```
*ese 9
```

Using TSP

Send the command:

```
status.standard.enable = 9
```

Reading the registers

You can read any register in the status model. The response is a decimal value that indicates which bits in the register are set. See Understanding bit settings (on page 16-15) for information on how to convert the decimal value to bits.

Using SCPI commands:

If you are using SCPI, you use the query commands in the STATus subsystem and common commands to read registers.

Using TSP commands:

If you are using TSP, you print the TSP command to read the register. You can use either `print()`, which returns the decimal value, or `print(tostring())`, which returns the string equivalent of the decimal value.

You can also send the common commands to read the register.

For example, you can send any one of the following commands to read the Status Enable Register of the Standard Event Register:

```
print(status.standard.enable)
*ese?
print(tostring(status.standard.enable))
```

Understanding bit settings

When you write to or read a status register, you can use binary, decimal, or hexadecimal values to represent the binary values of the bit states. When the value is converted to its binary equivalent, you can determine which bits are set on or clear. Zero (0) indicates that all bits are clear.

In the 2470, the least significant bit is always bit B0. The most significant bit differs for each register, but in most cases is either bit B7 or bit B15.

Bit position	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
Binary value	1000 0000	0100 0000	0010 0000	0001 0000	1000	0100	0010	0001
Decimal value	128	64	32	16	8	4	2	1
Weight	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Bit position	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	1000 0000 0000 0000	0100 0000 0000 0000	0010 0000 0000 0000	0001 0000 0000 0000	1000 0000 0000	0100 0000 0000	0010 0000 0000	0001 0000 0000
Decimal value	32768	16384	8192	4096	2048	1024	512	256
Weight	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

For example, if a value of 129 is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set and all other bits are cleared. If you read a value of 12288 for the condition register, the binary equivalent is 0011 0000 0000 0000. This value indicates that bits B12 and B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

When bit B12 (4096) and bit B13 (8192) are set (1), the decimal equivalent is 4096 + 8192 = 12,288.

Clearing registers

Registers in the status model can be cleared using commands or by instrument actions. When a register is cleared, the bits in the register are set to 0.

The event log and all registers are cleared when instrument power is cycled.

When you read a bit from the Operation Event, Questionable Event, or Standard Event Status Register, the entire 16-bit or 8-bit register value is returned. The event register is cleared or set to 0.

Using SCPI commands:

To clear the event registers of the Questionable Event Status Register and Operation Event Status Register sets, Standard Event Register, and Status Byte Register, send:

```
*CLS
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register and Operation Event Enable Register sets.

To clear the Questionable Event Status Register, the Operation Event Status Register sets, Standard Event Register, and Status Byte Register, send:

```
STATus:CLEar
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register or Operation Event Enable Register sets.

Using TSP commands:

To clear the event registers of the Questionable Event Status Register and Operation Event Status Register sets, Standard Event Register, and Status Byte Register send:

```
*CLS
```

To clear the Questionable Event Status Register, the Operation Event Status Register sets, Standard Event Register, and Status Byte Register send:

```
status.clear()
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register or Operation Event Enable Register sets.

Status model programming examples

The following examples illustrate how to generate an SRQ using the status model.

SRQ when the SMU reaches its source limit

This example demonstrates how to generate a service request (SRQ) when the source-measure unit (SMU) detects it has reached its source limit. After configuring the status model, in this example the source will be configured to output current and measure voltage. If the output terminals of the SMU are left open when running this example, the SMU will reach its source limit and generate an SRQ.

Using SCPI commands:

```
*RST
STAT:CLE
STAT:OPER:MAP 0, 5080, 5081
STAT:OPER:ENAB 1
*SRE 128
SOUR:FUNC CURR
SOUR:CURR:RANG 1e-3
SOUR:CURR 1e-3
SOUR:CURR:VLIM 1
SENS:FUNC "VOLT"
OUTP ON
READ?
OUTP OFF
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()
-- Map the event numbers
-- Map bit 0 of operational status register to set on reaching the
-- source limit (5080) and clear on dropping below the source
-- limit (5081)
status.operation.setmap(0, 5080, 5081)
-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1
-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB
-- This code will make SMU reach V limit if output terminals are open
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 1e-3
smu.source.level = 1e-3
smu.source.vlimit.level = 1
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.source.output = smu.ON
print(smu.measure.read())
smu.source.output = smu.OFF
```

SRQ when trigger model is finished

This example shows you how to generate a service request (SRQ) when the trigger model is completed and the 2470 has returned to the Idle state. After configuring the status model, this code configures and runs the trigger model. When the trigger model completes, the instrument generates an SRQ and the data is returned.

Using SCPI commands:

```
*RST
TRAC:CLE
STAT:CLE
STAT:OPER:MAP 0, 2732, 2731
STAT:OPER:ENAB 1
*SRE 128
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
TRIG:BLOC:BUFF:CLE 1
TRIG:BLOC:SOUR:STAT 2, ON
TRIG:BLOC:DEL:CONS 3, 100e-3
TRIG:BLOC:MDIG 4, "defbuffer1"
TRIG:BLOC:BRAN:COUN 5, 9, 3
TRIG:BLOC:SOUR:STAT 6, OFF
INIT
*WAI
TRAC:DATA? 1, 9, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the reading buffer
defbuffer1.clear()
-- Clear the status byte
status.clear()
-- Map bit 0 of operational status register to set on trigger
-- model exit (2732) and clear on trigger model enter (2731).
status.operation.setmap(0, 2732, 2731)
-- Enable bit 0 to flow through to the status byte
status.operation.enable = 1
-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB
-- Configure a simple measurement loop using the trigger model
smu.source.level = 1
smu.source.ilimit.level = 10e-3
-- Configure a simple trigger model to take 10 measurements
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 100e-3)
trigger.model.setblock(4, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 9, 3)
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Start the trigger model
trigger.model.initiate()
-- Instrument will generate an SRQ when done
waitcomplete()
printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ on trigger model notify event

This example shows you how to use the trigger model's Log Event block to generate a service request (SRQ). This example configures the trigger model to perform a sweep and to repeat that sweep multiple times. The Log Event block is used to generate events to indicate when a sweep starts and when it ends. An SRQ is generated each time the sweep ends. This example is most useful when you are gathering several sweeps of data on a device and you want to notify the controlling computer when each sweep has completed so it can retrieve the data from the sweep without interrupting the test.

Using SCPI commands:

```
*RST
SOUR:CONF:LIST:CRE "sourceList"
SOUR:VOLT:RANG 10
SOUR:VOLT:ILIM 10e-3
SENS:NPLC 1
SENS:CURR:RANG 10e-3
SENS:AZER:ONCE
SOUR:VOLT 1
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 2
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 3
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 4
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 5
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 6
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 7
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 8
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 9
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 10
SOUR:CONF:LIST:STORE "sourceList"
TRAC:CLE
TRIG:BLOC:CONF:RECALL 1, "sourceList"
TRIG:BLOC:SOUR:STAT 2, ON
TRIG:BLOC:LOG:EVEN 3, INFO1, "Sweep started."
TRIG:BLOC:BRAN:ONCE 4, 6
TRIG:BLOC:CONF:NEXT 5, "sourceList"
TRIG:BLOC:DEL:CONS 6, 1e-3
TRIG:BLOC:MDIG 7, "defbuffer1"
TRIG:BLOC:BRAN:COUN 8, 11, 5
TRIG:BLOC:LOG:EVEN 9, INFO2, "Sweep complete."
TRIG:BLOC:BRAN:COUN 10, 5, 3
TRIG:BLOC:SOUR:STAT 11, OFF
STAT:CLE
STAT:OPER:MAP 0, 2735, 2734
```

```
STAT:OPER:ENAB 1
*SRE 128
INIT
*WAI
TRAC:DATA? 1, 55, "defbuffer1", READ
```

Using TSP commands:

```
reset()
smu.source.configlist.create("sourceList")
smu.source.range = 10
smu.source.ilimit.level = 10e-3
smu.measure.nplc = 1
smu.measure.range = 10e-3
smu.measure.autozero.once()
for i = 0, 10 do
   smu.source.level = i
   smu.source.configlist.store("sourceList")
end
defbuffer1.clear()
-- Configure the trigger model.
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "sourceList")
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(3, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "Sweep
   started.")
trigger.model.setblock(4, trigger.BLOCK_BRANCH_ONCE, 6)
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "sourceList")
trigger.model.setblock(6, trigger.BLOCK_DELAY_CONSTANT, 1e-3)
trigger.model.setblock(7, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(8, trigger.BLOCK_BRANCH_COUNTER, 11, 5)
trigger.model.setblock(9, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Sweep
   complete.")
trigger.model.setblock(10, trigger.BLOCK_BRANCH_COUNTER, 5, 3)
trigger.model.setblock(11, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Configure the Status Model
--======================
-- Clear the status byte.
status.clear()
-- Map the notify messages to operational register bit 0.
-- The Sweep Complete message sets the bit. The Sweep
-- Started message clears the bit.
status.operation.setmap(0, trigger.LOG_INFO2, trigger.LOG_INFO1)
-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1
-- Enable the Operational Summary Bit to set the Master Summary Bit/SRQ.
status.request_enable = status.OSB
-- Start the trigger model.
trigger.model.initiate()
waitcomplete()
print(string.format("Number of Readings in Buffer: %d", defbuffer1.n))
printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ on error

This example shows you how to generate a service request (SRQ) when an instrument error event occurs.

Using SCPI commands:

```
*RST
SYST:CLE
STAT:CLE
*SRE 4
MAKEERROR
```

Using TSP commands:

```
reset()
-- Clear Error Queue so EAV bit can go low.
eventlog.clear()

-- Clear the status byte.
status.clear()

-- Enable SRQ on error available.
status.request_enable = status.EAV

-- Send a line of code that will generate an error event.
beeper = 1
```

SRQ when reading buffer becomes full

This example shows you how to generate a service request (SRQ) when the 2470 reading buffer is full. You can use this to notify the controlling computer that it needs to read back the data and empty the buffer. After configuring the status model, this code configures the default reading buffer 1 to a size of 100, and then configures the 2470 to fill the buffer. After the buffer is full, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST
STAT:CLE
STAT:OPER:MAP 0, 4918, 4917
STAT:OPER:ENAB 1
*SRE 128
TRAC:CLE
TRAC:POIN 100, "defbuffer1"
SOUR:VOLT:RANG 1
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
COUNT 100
OUTP ON
READ? "defbuffer1"
OUTP OFF
TRAC:DATA? 1, 100, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()

-- Map bit 0 of operational status register to set on default buffer
-- full (4918) and clear on buffer empty (4917).
status.operation.setmap(0, 4918, 4917)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB

-- Clear the buffer and make it smaller
defbuffer1.clear()
defbuffer1.capacity = 100
smu.source.range = 1
smu.source.level = 1
smu.source.ilimit.level = 10e-3

-- Set the measure count to fill the buffer
smu.measure.count = 100
smu.measure.range = 10e-3
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
smu.source.output = smu.OFF

printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ when a measurement completes

This example shows you how to generate a service request (SRQ) when a measurement completes. This is most useful when you have a measurement that will take a long time to complete and you wish to free up the controlling computer to do other things while it is waiting. This can happen if you have configured the measurement with a long delay value, a large aperture, or have enabled filtering.

This code configures the source-measure unit (SMU) for a long reading, and then configures the trigger model to generate notify events before and after the measurement. The status model is then configured to generate an SRQ based on the "Measurement Done" notify event. The output is turned on and the trigger model is used to take the measurement. When the measurement completes, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST
TRAC:CLE
SOUR:VOLT:RANG 1
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
SENS:CURR:RANG 10e-3
SENS:NPLC 10
TRIG:BLOC:LOG:EVEN 1, INFO1, "Measurement Started."
TRIG:BLOC:MDIG 2, "defbuffer1"
TRIG:BLOC:LOG:EVEN 3, INFO2, "Measurement Done."
STAT:CLE
STAT:OPER:MAP 0, 2735, 2734
STAT:OPER:ENAB 1
*SRE 128
OUTP ON
INIT
*WAI
OUTP OFF
TRAC:DATA? 1, 1, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the reading buffer
defbuffer1.clear()
-- Configure the source and take a measurement
smu.source.range = 1
smu.source.level = 1
smu.source.ilimit.level = 10e-3
smu.measure.range = 10e-3
-- Setup the NPLC for a really long measurement
smu.measure.nplc = 10
-- Use the trigger model to create events before and after
-- the measurement to generate SRQ when measurement is done.
trigger.model.setblock(1, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "Measurement
   Started.")
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(3, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Measurement
   Done.")
-- Clear the status byte
status.clear()
-- Map bit 0 of the Operation Event Register to set on the Measurement
-- Done log notification (trigger.LOG_INFO2) and clear on the
-- Measurement Started log notification (trigger.LOG_INFO1).
status.operation.setmap(0, trigger.LOG_INFO2, trigger.LOG_INFO1)
-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1
-- Enable the Operational Summary Bit to set the Master Summary
-- Bit/RQS
status.request_enable = status.OSB
smu.source.output = smu.ON
trigger.model.initiate()
waitcomplete()
smu.source.output = smu.OFF
printbuffer(1, defbuffer1.n, defbuffer1)
```

Section 17

Frequently asked questions

In this section:

I see a command that is not in the manual. What is it? 17-1
How do I display the instrument's serial number? 17-2
What VISA resource name is required? 17-2
Can I use Keysight GPIB cards with Keithley drivers? 17-3
How do I check the USB driver for the device? 17-3
Which Microsoft Windows operating systems are supported? 17-4
What to do if the GPIB controller is not recognized? 17-5
I am receiving GPIB timeout errors. What should I do? 17-5
How do I change the command set? 17-5
How do I upgrade the firmware? 17-6
Where can I find updated drivers? 17-7
Why can't the 2470 read my USB flash drive? 17-8
How do I download measurements onto the USB flash drive? 17-8
How do I save the present state of the instrument? 17-9
Why did my settings change? 17-10
What is NPLC? 17-10
What are the Quick Setup options? 17-10
What is the output-off state? 17-11
Why is OVP displayed? 17-12
How do I store readings into the buffer? 17-13
What should I do if I get a 5074 interlock error? 17-14
How do I trigger a sweep? 17-15
What are source limits? 17-15
What is offset compensation? 17-16
What is a configuration list? 17-16
What does –113 "Undefined header" error mean? 17-17
Why do I see the "incompatible settings" message? 17-17
What does –410 "Query interrupted" error mean? 17-17
What does –420 "Query unterminated" error mean? 17-18
How do I use the digital I/O port? 17-18
How do I trigger other instruments? 17-18

I see a command that is not in the manual. What is it?

You may see commands that are internal to the instrument if you:

- Have the event log set to record commands
- Capture commands with the create setup feature
- Store settings in a configuration list

If a command is not described in the Command Reference section, do not use it.

The commands that you might see include:

- `smu.measure.configlist.set()`
- `smu.source.configlist.set()`
- `smu.setfuncs()`

How do I display the instrument's serial number?

The instrument serial number is on a label on the rear panel of the instrument. You can also access the serial number from the front panel using the front-panel and using remote commands.

To view the system information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Info/Manage**. The system information displays, including the serial number.
3. To return to the home screen, select the **HOME** key.

To view system information using SCPI commands:

Send the command:

```
*IDN?
```

To view system information using TSP commands:

Send the command:

```
print(localnode.serialno)
```

What VISA resource name is required?

To determine the VISA resource name that is required to communicate with the instrument, you can run the Keithley Configuration Panel. The Configuration Panel automatically detects all instruments connected to the computer.

If you installed the Keithley I/O Layer, you can access the Keithley Configuration Panel through the Microsoft® Windows® Start menu.

To run the Configuration Panel, select **Start > All Programs > Keithley Instruments > Keithley Configuration Panel** and follow the steps in the wizard.

Can I use Keysight GPIB cards with Keithley drivers?

Yes, if the instrument driver uses VISA for instrument communication. This is true for any instrument driver that is IVI or VXI/PnP based.

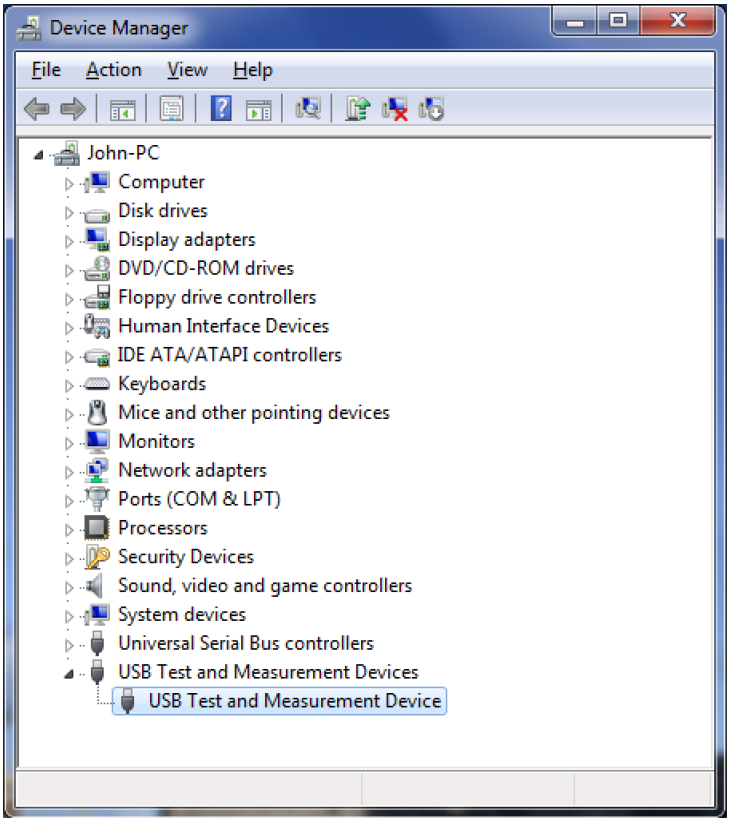
How do I check the USB driver for the device?

To check the driver for the USB Test and Measurement Device:

1. Open Device Manager. From the **Start** menu, you can enter `devmgmt.msc` in the Run box or the Windows search box to start Device Manager.
2. Under USB Test and Measurement Devices, look for USB Test and Measurement Device.

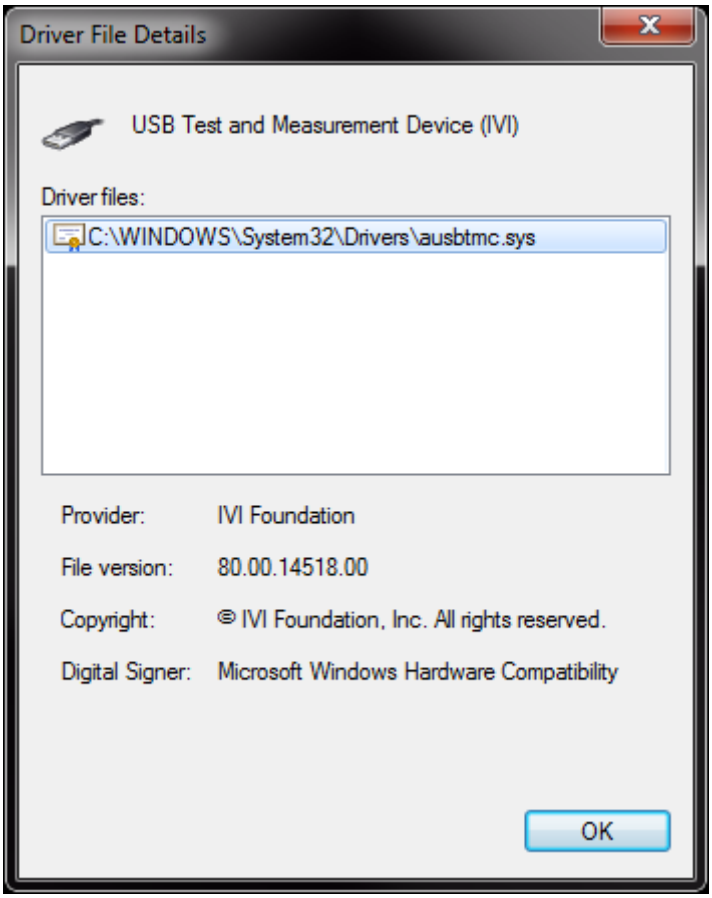
   If the device is not there, either VISA is not installed or the instrument is not plugged in and switched on.

Figure 161: Device Manager dialog box showing USB Test and Measurement Device



3. Right-click the device.
4. Select **Properties**.
5. Select the **Driver** tab.
6. Select **Driver Details**.
7. Verify that the device driver is the `ausbtmc.sys` driver from IVI Foundation.

Figure 162: Driver File Details dialog box



8. If the incorrect driver is installed, select **OK**.

If this does not work, uninstall VISA, unplug the instrument, and follow the steps to reinstall VISA in the section Modifying, repairing, or removing Keithley I/O Layer software (on page 2-34).

Which Microsoft Windows operating systems are supported?

Microsoft Windows 8 and Windows 10 are supported.

What to do if the GPIB controller is not recognized?

If the hardware is not recognized by the computer:

1. Check for newer drivers on the vendor's website.
2. Check that the drivers are valid for the operating system you have and any updates that might be necessary. This information is typically found in the readme file that comes with the drivers.
3. Follow vendor instructions on updating drivers.

If it is still not recognized, you can try a different computer using a different operating system to rule out operating system issues.

If this does not resolve the issue, contact the vendor of the GPIB controller for assistance.

I am receiving GPIB timeout errors. What should I do?

If your GPIB controller is recognized by the operating system, but you get a timeout error when you try to communicate with the instrument, check the following:

1. Confirm that the GPIB address you assigned to the instrument is unique and between 0 to 30. Do not use 0 or 21 because they are common controller addresses.
2. Check cabling connections. GPIB cables are heavy and can fall out of the connectors if they are not screwed in securely.
3. Substitute cables to verify cable integrity. For example, if you can send and receive ASCII text, but you cannot do a binary transfer, check your program and the decoding of the binary data. If that does not resolve the problem, try another cable. ASCII text only uses seven data lines in the cable; the binary transfer requires all eight lines.

How do I change the command set?

You can change the command set that you use with the 2470. The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

NOTE

As delivered from Keithley Instruments, the 2470 is set to work with the SCPI command set.

To set the command set from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the appropriate **Command Set**.

You are prompted to confirm the change to the command set and reboot.

To verify which command set is selected from a remote interface, send the command:

```
*LANG?
```

To change to the SCPI command set from a remote interface, send the command:

```
*LANG SCPI
```

Reboot the instrument.

To change to the TSP command set from a remote interface, send the command:

```
*LANG TSP
```

Reboot the instrument.

How do I upgrade the firmware?

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

CAUTION

Do not initialize TSP-Link before starting the upgrade.

NOTE

You can upgrade or downgrade the firmware from the front panel or from the virtual front panel. Refer to Using the 2470 virtual front panel (on page 2-30) for information.

From the front panel or virtual front panel:

1. Copy the firmware file (`.upg` file) to a USB flash drive.
2. Verify that the firmware file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect any input and output terminals that are attached to the instrument.
4. Turn the instrument power off. Wait a few seconds.
5. Turn the instrument power on.
6. Insert the flash drive into the USB port on the front panel of the instrument.
7. From the instrument front panel, press the **MENU** key.
8. Under System, select **Info/Manage**.
9. Choose an upgrade option:
   - To upgrade to a newer version of firmware, select **Upgrade to New**.
   - To return to a previous version of firmware, select **Downgrade to Older**.
10. If the instrument is controlled remotely, a message is displayed. Select **Yes** to continue.
11. When the upgrade is complete, reboot the instrument.

A message is displayed while the upgrade is in progress.

For additional information about upgrading the firmware, see Upgrading the firmware (on page 10-3).

Where can I find updated drivers?

For the latest drivers and additional support information, see the Keithley Instruments support website.

To see what drivers are available for your instrument:

1. Go to tek.com/support.
2. Enter the model number of your instrument.
3. Select **Software** from the filter list.
4. Select **Driver** from the filter list.

NOTE

If you use the native LabVIEW™ or IVI driver, you must configure the 2470 to use the SCPI command set. For information on changing the command set, refer to How do I change the command set? (on page 17-5).

Why can't the 2470 read my USB flash drive?

Verify that the flash drive is formatted with the FAT32 file system. The 2470 only supports FAT and FAT32 drives using Master Boot Record (MBR).

In Microsoft® Windows®, you can check the file system by checking the properties of the USB flash drive.

NOTE

Higher capacity USB drives take longer to be read and loaded by the instrument.

How do I download measurements onto the USB flash drive?

From the front panel, you can download measurements from a reading buffer to a .csv file on a USB flash drive.

Using the front panel to save or append buffer content to a file:

1. Insert a USB flash drive into the USB port.
2. Press the **MENU** key.
3. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 163: MANAGE READING BUFFERS window



4. Select the reading buffer that you want to save.
5. Select the **Save To USB** button.

6. Enter the name of the file in which to save the readings and select **OK**.

NOTE

You only have to enter the name of the file you want to save. You do not need to enter the file extension. All files are saved as `.csv` files.

7. Select **Yes** to confirm saving the file.

How do I save the present state of the instrument?

You can save the settings in the instrument as a script using the front-panel menus or from a remote interface. After they are saved, you can recall the script or copy it to a USB flash drive.

From the front panel:

1. Configure the 2470 to the settings that you want to save.
2. Press the **MENU** key.
3. Under Scripts, select **Create Setup**.
4. Select **Create**. A keyboard is displayed.
5. Use the keyboard to enter the name of the script.
6. Select the **OK** button on the displayed keyboard. The script is added to internal memory.

Using SCPI commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
   `*SAV <n>`
   Where <n> is an integer from 0 to 4.

NOTE

In the front-panel script menus, the setups saved with the `*SAV` command have the name `Setup0`*x*, where *x* is the value you set for <n>.

Using TSP commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
   `createconfigscript("`*setupName*`")`
   Where *setupName* is the name of the setup script that is created.

Why did my settings change?

Many of the commands in the 2470 are saved with the source or measure function that was active when you set them. For example, assume you have the measure function set to current and you set a value for display digits. When you change the measure function to voltage, the display digits value changes to the value that was last set for the voltage measure function. When you return to the current measure function, the display digits value returns to the value you set previously.

What is NPLC?

You can adjust the amount of time that the input signal is measured. Adjustments to the amount of time affect the usable measurement resolution, the amount of reading noise, and the reading rate of the instrument.

The amount of time is specified in parameters that are based on the number of power line cycles (NPLCs). Each power line cycle for 60 Hz is 16.67 ms (1/60); for 50 Hz, it is 20 ms (1/50).

The shortest amount of time results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

What are the Quick Setup options?

The **QUICKSET** key opens a screen that provides access to function selection, performance adjustments, and quick setups.

The Function button on the Quickset menu allows you to select a source or measure function. The options are the same as those available when you use the front-panel **FUNCTION** key.

The Performance slider allows you to adjust speed and resolution. As you increase speed, you lower the amount of resolution. As you increase resolution, you decrease the reading speed. These settings take effect the next time the output is turned on and measurements are made.

The Quick Setups allow you to set the instrument to operate as a Voltmeter, Ammeter, Ohmmeter, or Power Supply.

CAUTION

When you select a Quick Setup, the instrument turns the output on. Carefully consider and configure the appropriate output-off state, source, and limits before connecting the 2470 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

What is the output-off state?

When the source of the instrument is turned off, it may not completely isolate the instrument from the external circuit. You can use the output-off setting to place the 2470 in a known, noninteractive state during idle periods, such as when changing the device under test. The appropriate output-off state depends on your system and the device under test. Different types of connected devices or loads require different behaviors from the 2470 when the output is turned off. For example, a passive device such as a diode is not affected by a 0 V source connected across its terminals when the output is turned off. However, connecting a 0 V source to the terminals of a battery causes the battery to discharge.

The output-off states that can be selected for a 2470 are normal, high-impedance, zero, or guard.

CAUTION

Carefully consider and configure the appropriate output-off state, source, and source limits before connecting the 2470 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and source limits may result in damage to the instrument or to the device under test (DUT).

When the 2470 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10% of the full scale of the present measurement function autorange value
- If source readback is off, Output Off is displayed in the home screen Source area
- If source readback is on, the actual measurement is displayed in the home screen Source area
- If measurement is set to resistance, dashes (--.----) are shown in the home screen Source area
- The Source button on the home screen shows the value that will be sourced when the output is turned on again

When the zero output-off state is selected and you turn off the output:

- The measurement sense is changed to 2-wire
- The voltage source is selected and set to 0 V
- The range is set to the presently selected range (turn off autorange)
- If the source is voltage, the current limit is not changed
- If the source is current, the current limit is set to the programmed source current value or to 10% full scale of the present current range, whichever is greater

When the zero output-off state is selected, you can use the instrument as an ammeter because it is outputting 0 V.

When the high-impedance output-off state is selected and the output is turned off:

- The measurement sense is set to 2-wire
- The output relay opens, disconnecting the instrument as a load

Opening the relay disconnects external circuitry from the inputs and outputs of the instrument. To prevent excessive wear on the output relay, do not use this output-off state for tests that turn the output off and on frequently.

The high-impedance output-off state should be used when the instrument is connected to a power source or another source-measure instrument. In some cases, it may also be appropriate for devices such as capacitors.

When the guard output-off state is selected and the output is turned off, the following actions occur:

- The measurement sense is changed to 2-wire
- The current source is selected and set to 0 A if the source is set to current (amps); otherwise, the output remains a voltage source when the output is turned off
- The voltage limit is set to 10% full scale of the present voltage range

Why is OVP displayed?

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced. This protects the device under test (DUT) from high voltage levels.

For example, if a sense lead is disconnected or broken during a 4-wire sense measurement, the instrument can interpret the missing sense lead as a decrease in voltage and respond by increasing the source output. If overvoltage protection is set, the sourced output is not allowed to exceed the overvoltage protection limit.

The value set for overvoltage protection takes precedence over the source limit settings. When it is enabled, it is always in effect.

When overvoltage protection is set and the sourced voltage exceeds the setting:

- The output is clamped at the overvoltage protection value
- On the front panel, an indicator to the right of the voltage displays OVP

When overvoltage protection is used in a test sequence, it should be set before turning the source on.

NOTE

Even when overvoltage protection is set to None, overvoltage protection is active to 1165 V. If the instrument indicates that OVP is active, check the sense connections and setttings to make sure the connections and settings are correct and that there are no broken leads.

WARNING

Even with the overvoltage protection set to the lowest value (20 V), never touch anything connected to the terminals of the 2470 when the instrument is on. Always assume that a hazardous voltage (greater than 30 V_{RMS}) is present when the instrument is on. To prevent damage to the device under test or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

For more information on overvoltage protection and how to set it, see Overvoltage protection (on page 4-35).

How do I store readings into the buffer?

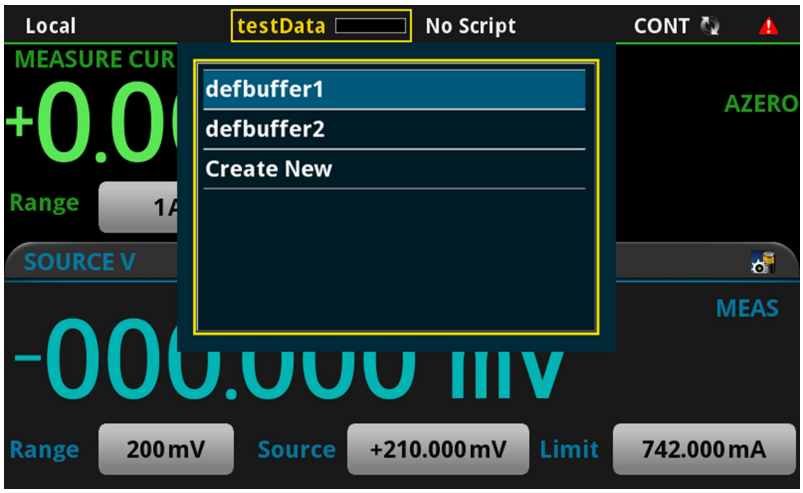
Readings are automatically stored into default buffer 1 (`defbuffer1`).

To store readings into a different buffer, you can select another buffer from the buffer indicator on the home screen.

Located to the right of the instrument active state indicator arrows, this indicator shows the name of the active reading buffer. Select the indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar.

The green bar next to the buffer name indicates how full the buffer is.

Figure 164: 2470 active buffer indicator expanded



For more information on buffers, including information on creating a user-defined buffer, see Reading buffers (on page 6-1).

What should I do if I get a 5074 interlock error?

The instrument provides an interlock circuit on the rear panel. You must enable this circuit in order for the instrument to set source voltages greater than ±42 V DC. If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, you see event code 5074, “Output voltage limited by interlock.”

WARNING

The 2470 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

The action when the interlock signal is not asserted depends on the Interlock setting. If Interlock is set to Off, if the safety interlock signal is not asserted, the following occurs:

- The nominal output is limited to less than ±42 V.
- The front-panel INTERLOCK indicator is not illuminated.
- You can output voltages less than ±42.

If Interlock is set to On, when the safety interlock signal is not asserted, the following occurs:

- You cannot turn on the source output.
- The front-panel INTERLOCK indicator is not illuminated.

Whenever the interlock changes state (from asserted to not asserted or vice versa), the output is turned off.

To recover from this error, properly engage the interlock using a safe test fixture before turning on the 2470 output. The Source setting on the SOURCE swipe screen displays the value that was selected for the voltage source, but the source value is limited to less than ±42 V. The SOURCE swipe screen shows the value that the instrument is outputting.

See Using the interlock (on page 4-3) for more information.

How do I trigger a sweep?

Sweeps are set up as a trigger model, so to start the sweep, initiate the trigger model. You can initiate the trigger model from the front panel by pressing the TRIGGER key.

What are source limits?

The source limits (also known as compliance) prevent the instrument from sourcing a voltage or current over a set value. This helps prevent damage to the device under test (DUT).

The values that can be set for the limits must be below the setting for the overvoltage protection limit.

This limit can also be restricted by the measurement range. If a specific measurement range is set, the limit must be more than 0.1 percent of the measurement range. If not, an event is generated and the limit is automatically changed to an appropriate value for the selected range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

If you attempt to change the source limit to a value that is not appropriate for the selected source range, the source limit is not changed and a warning is generated. You must change the source range before you can select the new limit.

The lowest allowable limit is based on the load and the source value. For example, if you are sourcing 1 V to a 1 kΩ resistor, the lowest allowable current limit is 1 mA (1 V/1 kΩ = 1 mA). Setting a limit lower than 1 mA limits the source.

The effective source limit is the lesser of either the programmed source limit or 105% of the active measure range. If you use fixed measure ranges, the instrument prevents you from selecting different limit and measure ranges. However, if measure autorange is selected, it is possible for the autorange process to cause the ranges to differ because the instrument may go down to a range that is lower than the one on which the source limit is programmed. This causes the effective source limit to drop to 105% of the newly selected measure range. For example, the output may be limited if you make a measurement that causes the range to be lowered and then increase the source level or make a change to the device under test or an external source. In this situation, the source voltage or current is limited to conform with the lower measurement range until another measurement causes the instrument to select a higher range to accommodate the new source value. If no further measurement is made, the source may remain limited to the present measurement range indefinitely.

To prevent the source output from being limited below the source limit setting, you can enable the autorange rebound feature. When autorange rebound is enabled, after an autoranged measurement is made, the measure range is restored to match the limit range when the autoranged measurement is complete. This ensures that the source does not limit at less than the full limit setting.

If the source output exceeds the source limit:

- On the home screen, LIMIT is displayed to the right of the source voltage.
- The Source value changes to yellow.

The source is clamped at the maximum limit value.

What is offset compensation?

Offset compensation is a measuring technique that reduces or eliminates thermoelectric EMFs in low-level resistance measurements. The voltage offsets because of the presence of thermoelectric EMFs (V_{EMF}) can adversely affect resistance measurement accuracy.

To overcome these offset voltages, you can use offset-compensated ohms.

What is a configuration list?

A configuration list is a list of stored instrument settings. You can restore these instrument settings to change the active state of the instrument. Configuration lists allow you to record the active state of the instrument, store it, and then return the instrument to that state as needed.

If you are using TSP, configuration lists run faster than a script that is set up to configure the same settings.

The 2470 supports source configuration lists and measure configuration lists, making it possible to sequence through defined source settings, measure settings, or both.

Each configuration list consists of a list of configuration indexes. A configuration index contains all instrument source or measure settings that were active at a specific point. You can cycle through the configuration indexes using a trigger model.

For more detail, see Configuration lists (on page 4-82).

What does –113 "Undefined header" error mean?

When using the SCPI command language, you may see the –113, "Undefined header," error. This error indicates that what you sent to the instrument did not contain a recognizable command name. The most likely causes for this are:

- A missing space between the command and its parameter. There must be one or more spaces between the command and its parameter. For example,
  `:disp:volt:digits5`
  The correct entry is
  `:disp:volt:digits 5`
- Incorrect short or long form. Check the SCPI command reference (on page 12-1) documentation for the correct command name.
- Spaces in the command name. You cannot use spaces in the command name. For example:
  `syst: err?`
  The correct entry is:
  `:syst:err?`

Why do I see the "incompatible settings" message?

The "Continuous measurements have been terminated because of incompatible settings" message indicates that the combination of settings that are presently configured make it impossible for the instrument to make a valid measurement.

To resolve this problem, make changes to your settings.

What does –410 "Query interrupted" error mean?

This error occurs when you have sent a valid query to the instrument and then send it another command, query, or a Group Execute Trigger (GET) before it has had a chance to send the entire response message (including the line-feed/EOI terminator). The most likely causes are:

- Sending a query to the instrument and then sending another command or query before reading the response to the first query. For example, the following sequence of commands causes an error -410:
  ```
  syst:err?
  *opc?
  ```
  You must read the response to `syst:err?` before sending another command or query.
- Incorrectly configured IEEE 488 driver. The driver must be configured so that when talking on the bus it sends line-feed with EOI as the terminator, and when listening on the bus it expects line-feed with EOI as the terminator. See the reference manual for your IEEE 488 interface.

What does –420 "Query unterminated" error mean?

This error occurs when you address the instrument to talk and it has nothing to say. The most likely causes are:

- A query was not sent. You must send a valid query to the instrument before addressing it to talk. You cannot get a reading until you send the instrument a query.
- An invalid query was sent. If you sent a query and get this error, make sure that the instrument is processing the query without error. For example, sending a query that generates an "Undefined header" error and then addressing the instrument to talk will generate a "Query unterminated" error.
- A valid query in a command string that also contains an invalid command. This can occur when you send multiple commands or queries in one command string (program message). When the instrument detects an error in a command string, it discards all further commands in the command string until the end of the string. For example, this command string would result in a query unterminated error:
  `:sens:date?;:sens:func?`

  The first command (`:sens:date?`) generates error -113, "Undefined header" and the instrument discards the second command (`:sens:func?`), even though it is a valid query.

How do I use the digital I/O port?

You can use the 2470 digital input/output with the trigger model or to control an external digital circuit, such as a device handler used to perform binning operations. To control or configure any of the six digital input/output lines, send commands to the 2470 over a remote interface.

To use the 2470 digital I/O in a trigger link system (TLINK), connect it using a 2470-TLINK Trigger Link Cable and configure the 2470 digital input and output lines.

For more information about the 2470 digital I/O port, see Digital I/O (on page 8-12).

How do I trigger other instruments?

You can use the 2470 digital input/output to control an external digital circuit, such as a device handler used to perform binning operations. For more information about the 2470 digital I/O port, see Digital I/O (on page 8-12).

You can also use the digital I/O in a trigger link system (TLINK) using a 2450-TLINK Trigger Link Cable.

Another option is the Keithley Instruments TSP-Link® interface, a high-speed trigger synchronization and communication bus that you can use to connect multiple instruments in a master and subordinate configuration. See TSP-Link System Expansion Interface (on page 9-1) for additional information.

Section 18

Next steps

In this section:

Additional 2470 information..... 18-1

Additional 2470 information

For additional information about the 2470, refer to the Keithley Instruments website (tek.com/keithley), which contains the most up-to-date information. From the website, you can access:

- The *Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*
- The *Semiconductor Device Test Applications Guide*
- Application notes
- Updated drivers
- Updated firmware
- Information about related products, including:
  - Model 2450 Interactive SourceMeter® Instrument
  - Model 2460 High-Current Interactive SourceMeter® Instrument
  - Model 2461 High-Current Interactive SourceMeter Instrument
  - The Series 2600B System SourceMeter® Instruments

You can also contact your local Field Applications Engineer: They can help you with product selection, configuration, and usage. Check the website for contact information.

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments.
All other trademarks and trade names are the property of their respective companies.

Keithley Instruments
Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-800-935-5595 • tek.com/keithley

KEITHLEY
A Tektronix Company