

Model 2606B System SourceMeter® Instrument

Reference Manual

2606B-901-01 Rev. C November 2021



2606B-901-01C

Model 2606B
System SourceMeter® Instrument
Reference Manual

© 2021, 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™ and TSP-Link™ 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 (<https://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: 2606B-901-01 Rev. C November 2021

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 2018.

Table of contents

Introduction	1-1
Welcome	1-1
Extended warranty	1-1
Contact information	1-1
2606B documentation	1-2
Product software and drivers	1-3
Capabilities and features.....	1-4
What you should have received	1-5
Instrument and module serial numbers.....	1-5
General ratings.....	1-6
 Installation	 2-1
Installing the 2606B.....	2-1
Cooling vents	2-1
Dimensions	2-2
Turning the instrument on and off.....	2-4
Placing the 2606B in standby	2-5
Warmup period.....	2-5
Line frequency configuration	2-6
System information	2-6
 Instrument description	 3-1
Front panel	3-1
Rear panel.....	3-2
Channel connector	3-2
Ground	3-3
Line fuse and power receptacle	3-3
Remote communications interfaces	3-3
Remote interface connections.....	3-4
Output queue	3-4
USB communications.....	3-4
LAN communications	3-9
Software components for applications and instrument drivers	3-10
Keithley I/O layer.....	3-13
Using the web interface.....	3-15
How to access the web interface.....	3-16
Web interface Welcome page	3-16
Use the ID button to identify the module	3-17
Change the IP configuration through the web interface	3-17
Set the instrument password.....	3-19
View buffer data using the web interface	3-19

Download reading buffer data using the web interface	3-20
Use TSB Embedded	3-21
Flash Upgrade.....	3-21
Send individual commands using the web interface.....	3-21
Review events in the LXI Event Log.....	3-21
Help.....	3-24
Virtual Front Panel	3-24
Virtual front panel options	3-25
Menu overview	3-28
Beeper.....	3-33
Display mode	3-34
Display operations.....	3-35
Display functions and attributes	3-35
Display features	3-35
Display messages	3-37
Input prompting	3-41
Indicators.....	3-43
Local lockout	3-44
Load test menu	3-44
Running a test from the virtual front panel.....	3-46
Key-press codes	3-46
Operation.....	4-1
Limits	4-1
Virtual front-panel limit	4-1
Remote limit	4-2
Fundamental circuit configurations	4-2
DUT test connections.....	4-3
Input/output connectors.....	4-4
2-wire local sensing connections.....	4-6
4-wire remote sensing connections.....	4-6
Contact check connections	4-7
Guarding and shielding	4-7
Test fixtures.....	4-11
Floating a SMU	4-12
Saved setups	4-13
Saving user setups.....	4-14
Start-up configuration.....	4-15
Restoring the factory default setups using remote commands.....	4-16
USB storage overview.....	4-16
Connecting the USB flash drive	4-17
File system navigation.....	4-17
Displayed error and status messages.....	4-18
Sourcing and measuring	5-1
Basic operation	5-1
Source-measure capabilities	5-2
virtual front-panel source-measure procedure.....	5-2
Remote source-measure commands	5-4
Basic circuit configurations	5-5
Source V	5-5

Source I.....	5-7
Source I measure I, source V measure V	5-8
Measure only (voltage or current)	5-8
Contact check	5-9
Operation considerations for the ADC	5-10
Autozero.....	5-10
NPLC caching	5-11
Sense mode selection.....	5-11
Front-panel sense mode selection	5-11
Selecting the sense from the remote interface	5-12
Triggering in local mode.....	5-12
Configuring trigger attributes in local mode	5-13
Front-panel triggering example	5-14
Configuring for measure-only tests using the MODE key	5-14
Voltmeter and ammeter measurements.....	5-15
Ohms measurements.....	5-15
Ohms ranging.....	5-16
Basic ohms measurement procedure.....	5-16
Ohms sensing	5-18
Power measurements	5-19
Basic power measurement procedure.....	5-20
Power measurements using the remote interface	5-20
Contact check measurements	5-21
Contact check commands.....	5-23
Contact check programming example.....	5-23
Multiple SMU connections.....	5-24
Output-off modes	5-25
Normal output-off mode	5-25
High-impedance output-off mode	5-26
Zero output-off mode.....	5-26
Output-off function.....	5-26
Output-off limits (compliance).....	5-28
Remote programming output-off states quick reference	5-28
Range.....	5-29
Maximum source values and readings	5-29
Measure autodelay.....	5-29
Ranging limitations.....	5-29
Manual ranging	5-30
Autoranging.....	5-30
Low range limits	5-30
Range considerations	5-31
Range commands	5-31
Digits	5-32
Setting display resolution from the virtual front panel.....	5-33
Setting display resolution from a remote interface	5-33
Speed	5-33
Setting the speed from the virtual front panel.....	5-34
Setting the speed using the remote interface	5-34
Sweep operation	5-34
Linear staircase sweeps.....	5-36
Logarithmic staircase sweeps	5-37

List sweeps	5-41
Pulse mode sweeps	5-42
Configuring and running sweeps	5-43
Sweeping using factory scripts	5-45
Sweep programming examples	5-46
Relative offset	5-47
Enabling and disabling relative offset from the front panel	5-48
Defining a relative offset value from the front panel	5-48
Relative offset commands	5-48
Filters	5-49
Filter types	5-49
Response time	5-50
Enabling the filter from the front panel	5-50
Configuring the filter from the front panel	5-51
Setting the filter using a remote interface	5-51
High-capacitance mode	5-52
Understanding high-capacitance mode	5-52
Enabling high-capacitance mode	5-54
Reading buffers	6-1
Overview	6-1
Front-panel reading buffer control	6-1
Reading buffer options	6-2
Configuring reading buffers	6-3
Appending or overwriting existing reading buffers	6-4
Storage operation	6-4
Saving reading buffers	6-5
Recalling readings	6-6
Remote reading buffer programming	6-6
Dedicated reading buffer designations	6-7
Reading buffer commands	6-7
Buffer status	6-13
Dynamic reading buffers	6-13
Buffer examples	6-13
Dedicated reading buffer example	6-13
Dual buffer example	6-15
Dynamically allocated buffer example	6-16
Triggering	7-1
Triggering	7-1
Remote triggering overview	7-1
Using the remote trigger model	7-3
Configuring source and measure actions	7-6
Enabling pulse mode sweeps using the end pulse action	7-7
SMU event detectors	7-7
Clearing SMU event detectors	7-8
Using the TRIG key to trigger a sweep	7-8
Using trigger events to start actions on trigger objects	7-10
Action overruns	7-10
Digital I/O port and TSP-Link synchronization lines	7-11

Mode	7-11
Pulse width.....	7-11
Trigger configuration on hardware lines	7-11
Action overruns on hardware lines	7-13
Timers	7-13
Timer attributes	7-13
Triggering a timer	7-14
Timer action overruns	7-15
Using timers to perform pulse mode sweeps	7-15
Event blenders	7-21
Event blender modes	7-21
Assigning input trigger events	7-22
Action overruns	7-22
LAN triggering overview	7-22
Understanding hardware value and pseudo line state	7-22
Understanding LXI trigger event designations	7-23
Generating LXI trigger packets.....	7-23
Command interface triggering.....	7-24
Trigger generator	7-24
Manual triggering	7-25
Interactive triggering.....	7-25
Detecting trigger events using the wait() function.....	7-25
Using the assert function to generate output triggers.....	7-26
Using the release function of the hardware lines.....	7-26
Using the set function to bypass SMU event detectors	7-27
Event detector overruns	7-27
Examples using interactive triggering.....	7-28
Hardware trigger modes.....	7-29
Falling edge trigger mode	7-30
Rising edge master trigger mode	7-31
Rising edge acceptor trigger mode	7-31
Either edge trigger mode.....	7-32
Understanding synchronous triggering modes	7-33
Synchronous master trigger mode (SynchronousM)	7-33
Synchronous acceptor trigger mode (SynchronousA)	7-35
Synchronous trigger mode	7-36
Digital I/O	7-37
Port configuration	7-37
Digital I/O configuration.....	7-38
Controlling digital I/O lines.....	7-38
Using output enable	7-40
TSP-Link trigger lines.....	7-41
Remote TSP-Link trigger line commands.....	7-42

Theory of operation 8-1

Source-measure concepts	8-1
Limit principles	8-1
Overheating protection.....	8-2
Operating boundaries.....	8-3
Guard	8-12
Analog-to-digital converter	8-14

Measurement settling time considerations.....	8-15
Effects of load on current source settling time	8-16
Creating pulses with the 2606B SMU	8-17
Pulse rise and fall times	8-17
Pulse width.....	8-18
LAN concepts and settings	9-1
Overview	9-1
Install the crossover cable.....	9-1
Connecting to the LAN	9-2
Set the IP address to be set automatically	9-3
Set the LAN configuration to Manual.....	9-3
Configuring the Domain Name System (DNS)	9-4
Establishing a point-to-point connection	9-5
LAN speeds.....	9-7
Duplex mode	9-7
Viewing LAN status messages	9-8
Viewing the network settings	9-9
Confirming the active speed and duplex negotiation.....	9-9
Confirming port numbers.....	9-10
Selecting a LAN interface protocol.....	9-10
VXI-11 connection.....	9-10
Raw socket connection	9-11
Dead socket connection.....	9-11
Telnet connection.....	9-11
Logging LAN trigger events in the event log	9-12
Accessing the event log from the command interface.....	9-14
Status model	10-1
Overview	10-1
Status register set contents.....	10-1
Queues	10-2
Status function summary.....	10-3
Status model diagrams	10-4
Clearing registers	10-13
Programming enable and transition registers	10-13
Reading registers	10-14
Status byte and service request (SRQ)	10-14
Status Byte Register	10-14
Service Request Enable Register	10-16
Serial polling and SRQ.....	10-16
Status byte and service request commands.....	10-17
Enable and transition registers.....	10-18
Controlling node and SRQ enable registers.....	10-18
Status register sets	10-18
System Summary Registers.....	10-18

Standard Event Register	10-19
Operation Status Registers	10-21
Questionable Status Registers	10-22
Measurement Event Registers	10-22
Register programming example	10-23
TSP-Link system status	10-24
Status model configuration example	10-24
Maintenance	11-1
Introduction	11-1
Line fuse replacement	11-1
Upgrading the firmware	11-2
Using TSB to upgrade the firmware	11-3
Troubleshooting guide	12-1
Introduction	12-1
Error levels	12-1
Effects of errors on scripts	12-1
Retrieving errors	12-2
Error summary list	12-2
LAN troubleshooting suggestions	12-6
Introduction to TSP operation	13-1
Introduction to TSP operation	13-1
Controlling the instrument by sending individual command messages	13-1
Queries	13-3
Information on scripting and programming	13-3
About TSP commands	13-3
Beeper control	13-3
Bit manipulation and logic operations	13-4
Data queue	13-5
Digital I/O	13-5
Display	13-6
Error queue	13-6
Event log	13-6
File I/O	13-7
Instrument identification	13-8
LAN and LXI	13-8
Miscellaneous	13-9
Parallel script execution	13-10
Queries and response messages	13-10
Reading buffer	13-10
Reset	13-11
Saved setups	13-11
Scripting	13-12
SMU	13-12
SMU calibration	13-14
Status model	13-14
Time	13-15
Triggering	13-15

TSP-Link	13-17
TSP-Net	13-18
Userstrings	13-18
Factory scripts	13-19
Running a factory script	13-19
Retrieving and modifying a factory script listing	13-20
KISweep factory script	13-20
KIPulse factory script	13-20
KIHighC factory script	13-22
KIParlib factory script	13-22
KISavebuffer factory script	13-22

Instrument programming..... 14-1

Fundamentals of scripting for TSP	14-1
What is a script?.....	14-2
Runtime and nonvolatile memory storage of scripts.....	14-2
What can be included in scripts?.....	14-2
Commands that cannot be used in scripts	14-3
Manage scripts.....	14-3
Working with scripts in nonvolatile memory.....	14-10
Programming example: Interactive script.....	14-13
Fundamentals of programming for TSP	14-14
What is Lua?	14-14
Lua basics	14-14
Standard libraries	14-28
Programming example: User script.....	14-32
Test Script Builder.....	14-32
Installing the TSB software.....	14-32
Using Test Script Builder (TSB)	14-33
Project navigator	14-34
Script editor	14-35
Outline view.....	14-35
Programming interaction	14-36
Password management	14-36
Setting the password from a command or web interface	14-36
Unlocking the remote interface.....	14-37
Resetting the password.....	14-37
Working with TSB Embedded	14-38
Simplify the display	14-38
Create a script using TSB Embedded	14-39
Copy an existing script	14-39
Run a script.....	14-39
Delete a script	14-40
Modify a script.....	14-40
Import a script from a computer	14-40
Export a script to a computer	14-41
Export a script to the instrument USB	14-41
Reset the instrument using TSB Embedded	14-41
Send individual instrument commands with TSB Embedded	14-41
Advanced scripting for TSP	14-42
Global variables and the script.user.scripts table	14-42
Create a script using the script.new() command	14-43
Rename a script.....	14-46
Retrieve a user script	14-48
Delete user scripts from the instrument.....	14-49

Restore a script to the runtime environment	14-50
Memory considerations for the runtime environment	14-50
TSP-Link system expansion interface.....	14-52
Master and subordinates.....	14-52
TSP-Link nodes.....	14-53
Connections	14-53
Initialization	14-54
Resetting the TSP-Link network.....	14-55
Accessing nodes	14-56
Using the reset() command	14-56
Using the abort command	14-57
Triggering with TSP-Link.....	14-57
TSP advanced features.....	14-57
Using groups to manage nodes on TSP-Link network	14-60
Running simultaneous test scripts.....	14-61
Using the data queue for real-time communication	14-63
Copying test scripts across the TSP-Link network	14-63
Removing stale values from the reading buffer cache	14-63
TSP-Net	14-64
TSP-Net capabilities.....	14-64
Using TSP-Net with any ethernet-enabled instrument	14-65
TSP-Net compared to TSP-Link to communicate with TSP-enabled devices	14-67
TSP-Net instrument commands: General device control	14-67
TSP-Net instrument commands: TSP-enabled device control	14-67
Example: Using tspnet commands.....	14-68

TSP command reference 15-1

TSP command programming notes	15-1
Placeholder text	15-1
Syntax rules	15-2
Time and date values.....	15-2
Using the TSP command reference.....	15-3
Command name and summary table	15-3
Command usage.....	15-5
Command details	15-5
Example section.....	15-6
Related commands and information.....	15-6
TSP commands.....	15-7
beeper.beep().....	15-7
beeper.enable	15-8
bit.bitand()	15-8
bit.bitor()	15-9
bit.bitxor()	15-10
bit.clear()	15-10
bit.get()	15-11
bit.getfield()	15-12
bit.set()	15-13
bit.setfield().....	15-14
bit.test()	15-15
bit.toggle()	15-16
bufferVar.appendmode	15-17
bufferVar.basetimestamp	15-18
bufferVar.cachemode	15-19
bufferVar.capacity	15-20
bufferVar.clear()	15-21
bufferVar.clearcache().....	15-21
bufferVar.collectsourcevalues	15-22

bufferVar.collecttimestamps	15-23
bufferVar.fillcount	15-24
bufferVar.fillmode	15-25
bufferVar.measurefunctions	15-25
bufferVar.measureranges	15-26
bufferVar.n	15-28
bufferVar.readings.....	15-29
bufferVar.sourcefunctions	15-30
bufferVar.sourceoutputstates	15-31
bufferVar.sourceranges.....	15-31
bufferVar.sourcevalues	15-32
bufferVar.statuses	15-34
bufferVar.timestampresolution	15-35
bufferVar.timestamps	15-36
ConfigPulseIMeasureV()	15-37
ConfigPulseIMeasureVSweepLin()	15-39
ConfigPulseIMeasureVSweepLog()	15-41
ConfigPulseVMeasureI()	15-43
ConfigPulseVMeasureISweepLin()	15-45
ConfigPulseVMeasureISweepLog()	15-47
dataqueue.add()	15-49
dataqueue.CAPACITY	15-50
dataqueue.clear()	15-50
dataqueue.count	15-51
dataqueue.next()	15-52
delay()	15-53
digio.readbit()	15-54
digio.readport()	15-54
digio.trigger[N].assert().....	15-55
digio.trigger[N].clear().....	15-56
digio.trigger[N].EVENT_ID	15-56
digio.trigger[N].mode	15-57
digio.trigger[N].overrun.....	15-58
digio.trigger[N].pulsewidth	15-59
digio.trigger[N].release().....	15-59
digio.trigger[N].reset()	15-60
digio.trigger[N].stimulus.....	15-61
digio.trigger[N].wait()	15-62
digio.writebit().....	15-63
digio.writeport()	15-63
digio.writeprotect	15-64
display.clear().....	15-65
display.getannunciators()	15-65
display.getcursor().....	15-67
display.getlastkey()	15-68
display.gettext().....	15-69
display.inputvalue()	15-70
display.loadmenu.add()	15-72
display.loadmenu.catalog()	15-73
display.loadmenu.delete()	15-74
display.locallockout	15-74
display.menu().....	15-75
display.numpad	15-76
display.prompt()	15-76
display.screen	15-78
display.sendkey()	15-78
display.setcursor().....	15-79
display.settext()	15-80
display.smuX.digits	15-82
display.smuX.limit.func.....	15-82
display.smuX.measure.func	15-83

display.trigger.clear()	15-84
display.trigger.EVENT_ID	15-84
display.trigger.overrun	15-84
display.trigger.wait()	15-85
display.waitkey()	15-86
errorqueue.clear()	15-87
errorqueue.count	15-88
errorqueue.next()	15-88
eventlog.all()	15-89
eventlog.clear()	15-90
eventlog.count	15-90
eventlog.enable	15-91
eventlog.next()	15-92
eventlog.overwritemethod	15-93
exit()	15-93
fileVar.close()	15-94
fileVar.flush()	15-95
fileVar.read()	15-96
fileVar.seek()	15-97
fileVar.write()	15-99
format.asciiprecision	15-100
format.byteorder	15-100
format.data	15-102
fs.chdir()	15-103
fs.cwd()	15-104
fs.is_dir()	15-104
fs.is_file()	15-105
fs.mkdir()	15-106
fs.readdir()	15-107
fs.rmdir()	15-107
gettimezone()	15-108
gm_isweep()	15-109
gm_vsweep()	15-110
i_leakage_measure()	15-111
i_leakage_threshold()	15-112
InitiatePulseTest()	15-113
InitiatePulseTestDual()	15-115
io.close()	15-117
io.flush()	15-117
io.input()	15-118
io.open()	15-119
io.output()	15-119
io.read()	15-121
io.type()	15-122
io.write()	15-123
lan.applysettings()	15-125
lan.autoconnect	15-125
lan.config.dns.address[N]	15-126
lan.config.dns.domain	15-127
lan.config.dns.dynamic	15-127
lan.config.dns.hostname	15-128
lan.config.dns.verify	15-129
lan.config.duplex	15-129
lan.config.gateway	15-130
lan.config.ipaddress	15-131
lan.config.method	15-131
lan.config.speed	15-132
lan.config.subnetmask	15-133
lan.linktimeout	15-133
lan.lxidomain	15-134
lan.nagle	15-135

lan.reset()	15-135
lan.restoredefaults()	15-136
lan.status.dns.address[N]	15-137
lan.status.dns.name	15-137
lan.status.duplex	15-138
lan.status.gateway	15-139
lan.status.ipaddress	15-139
lan.status.macaddress	15-140
lan.status.port.dst	15-140
lan.status.port.rawsocket	15-141
lan.status.port.telnet	15-141
lan.status.port.vxi11	15-142
lan.status.speed	15-142
lan.status.subnetmask	15-143
lan.timedwait	15-143
lan.trigger[N].assert()	15-144
lan.trigger[N].clear()	15-144
lan.trigger[N].connect()	15-145
lan.trigger[N].connected	15-146
lan.trigger[N].disconnect()	15-146
lan.trigger[N].EVENT_ID	15-147
lan.trigger[N].ipaddress	15-147
lan.trigger[N].mode	15-148
lan.trigger[N].overrun	15-149
lan.trigger[N].protocol	15-150
lan.trigger[N].pseudostate	15-150
lan.trigger[N].stimulus	15-151
lan.trigger[N].wait()	15-152
localnode.autolinefreq	15-153
localnode.description	15-154
localnode.license	15-154
localnode.linefreq	15-155
localnode.model	15-155
localnode.password	15-156
localnode.passwordmode	15-157
localnode.prompts	15-158
localnode.prompts4882	15-159
localnode.reset()	15-159
localnode.revision	15-160
localnode.serialno	15-161
localnode.showerrors	15-161
makegetter()	15-162
makesetter()	15-163
meminfo()	15-163
node[N].execute()	15-164
node[N].getglobal()	15-165
node[N].setglobal()	15-166
opc()	15-166
os.remove()	15-167
os.rename()	15-167
os.time()	15-168
print()	15-169
printbuffer()	15-170
printnumber()	15-171
PulseIMeasureV()	15-172
PulseVMeasureI()	15-173
QueryPulseConfig()	15-174
reset()	15-176
savebuffer()	15-176
script.anonymous	15-177
script.delete()	15-178

script.factory.catalog()	15-178
script.load()	15-179
script.new()	15-180
script.newautorun()	15-181
script.restore()	15-182
script.run()	15-182
script.user.catalog()	15-183
scriptVar.autorun	15-184
scriptVar.list()	15-185
scriptVar.name	15-185
scriptVar.run()	15-186
scriptVar.save()	15-187
scriptVar.source	15-188
settime()	15-188
settimezone()	15-189
setup.poweron	15-190
setup.recall()	15-191
setup.save()	15-192
smuX.abort()	15-193
smuX.buffer.getstats()	15-193
smuX.buffer.recalculatestats()	15-195
smuX.cal.adjustdate	15-196
smuX.cal.date	15-197
smuX.cal.due	15-198
smuX.cal.lock()	15-199
smuX.cal.password	15-199
smuX.cal.polarity	15-200
smuX.cal.restore()	15-201
smuX.cal.save()	15-202
smuX.cal.state	15-203
smuX.cal.unlock()	15-204
smuX.contact.calibratehi()	15-204
smuX.contact.calibratelo()	15-206
smuX.contact.check()	15-207
smuX.contact.r()	15-208
smuX.contact.speed	15-209
smuX.contact.threshold	15-210
smuX.makebuffer()	15-210
smuX.measure.autorangeY	15-211
smuX.measure.autozero	15-212
smuX.measure.calibrateY()	15-214
smuX.measure.count	15-215
smuX.measure.delay	15-216
smuX.measure.delayfactor	15-217
smuX.measure.filter.count	15-217
smuX.measure.filter.enable	15-218
smuX.measure.filter.type	15-219
smuX.measure.highcrangedelayfactor	15-220
smuX.measure.interval	15-221
smuX.measure.lowrangeY	15-222
smuX.measure.nplc	15-223
smuX.measure.overlappedY()	15-223
smuX.measure.rangeY	15-224
smuX.measure.rel.enableY	15-226
smuX.measure.rel.levelY	15-227
smuX.measure.Y()	15-228
smuX.measureYandstep()	15-229
smuX.nvbufferY	15-230
smuX.reset()	15-231
smuX.savebuffer()	15-231
smuX.sense	15-232

smuX.source.autorangeY	15-233
smuX.source.calibrateY()	15-234
smuX.source.compliance	15-235
smuX.source.delay	15-236
smuX.source.func	15-237
smuX.source.highc	15-237
smuX.source.levelY	15-238
smuX.source.limitY	15-239
smuX.source.lowrangeY	15-240
smuX.source.offfunc	15-241
smuX.source.offlimitY	15-241
smuX.source.offmode	15-242
smuX.source.output	15-243
smuX.source.outputenableaction	15-244
smuX.source.rangeY	15-245
smuX.source.settling	15-246
smuX.trigger.arm.count	15-247
smuX.trigger.arm.set()	15-247
smuX.trigger.arm.stimulus	15-248
smuX.trigger.ARMED_EVENT_ID	15-249
smuX.trigger.autoclear	15-250
smuX.trigger.count	15-250
smuX.trigger.endpulse.action	15-252
smuX.trigger.endpulse.set()	15-252
smuX.trigger.endpulse.stimulus	15-254
smuX.trigger.endsweep.action	15-255
smuX.trigger.IDLE_EVENT_ID	15-256
smuX.trigger.initiate()	15-256
smuX.trigger.measure.action	15-257
smuX.trigger.measure.set()	15-258
smuX.trigger.measure.stimulus	15-259
smuX.trigger.measure.Y()	15-261
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	15-262
smuX.trigger.PULSE_COMPLETE_EVENT_ID	15-262
smuX.trigger.source.action	15-263
smuX.trigger.source.limitY	15-264
smuX.trigger.source.linearY()	15-265
smuX.trigger.source.listY()	15-266
smuX.trigger.source.logY()	15-267
smuX.trigger.source.set()	15-268
smuX.trigger.source.stimulus	15-269
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	15-270
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	15-271
smuX.trigger.SWEEPING_EVENT_ID	15-271
status.condition	15-273
status.measurement.*	15-275
status.measurement.buffer_available.*	15-277
status.measurement.current_limit.*	15-278
status.measurement.instrument.*	15-280
status.measurement.instrument.smuX.*	15-281
status.measurement.reading_overflow.*	15-284
status.measurement.voltage_limit.*	15-285
status.node_enable	15-286
status.node_event	15-288
status.operation.*	15-290
status.operation.calibrating.*	15-292
status.operation.instrument.*	15-293
status.operation.instrument.digio.*	15-296
status.operation.instrument.digio.trigger_overnun.*	15-297
status.operation.instrument.lan.*	15-299
status.operation.instrument.lan.trigger_overnun.*	15-301

status.operation.instrument.smuX.*	15-303
status.operation.instrument.smuX.trigger_overrun.*	15-306
status.operation.instrument.trigger_blender.*	15-308
status.operation.instrument.trigger_blender.trigger_overrun.*	15-309
status.operation.instrument.trigger_timer.*	15-311
status.operation.instrument.trigger_timer.trigger_overrun.*	15-312
status.operation.instrument.tsplink.*	15-315
status.operation.instrument.tsplink.trigger_overrun.*	15-316
status.operation.measuring.*	15-318
status.operation.remote.*	15-319
status.operation.sweeping.*	15-320
status.operation.trigger_overrun.*	15-322
status.operation.user.*	15-324
status.questionable.*	15-326
status.questionable.calibration.*	15-328
status.questionable.instrument.*	15-330
status.questionable.instrument.smuX.*	15-331
status.questionable.over_temperature.*	15-333
status.questionable.unstable_output.*	15-334
status.request_enable	15-336
status.request_event	15-338
status.reset()	15-339
status.standard.*	15-340
status.system.*	15-342
status.system2.*	15-344
status.system3.*	15-347
status.system4.*	15-349
status.system5.*	15-351
SweepLinMeasureV()	15-353
SweepListMeasureV()	15-354
SweepLogMeasureV()	15-355
SweepVLinMeasureI()	15-356
SweepVListMeasureI()	15-358
SweepVLogMeasureI()	15-359
timer.measure.t()	15-360
timer.reset()	15-361
trigger.blender[N].clear()	15-362
trigger.blender[N].EVENT_ID	15-362
trigger.blender[N].orenable	15-363
trigger.blender[N].overrun	15-363
trigger.blender[N].reset()	15-364
trigger.blender[N].stimulus[M]	15-365
trigger.blender[N].wait()	15-366
trigger.clear()	15-367
trigger.EVENT_ID	15-367
trigger.generator[N].assert()	15-368
trigger.generator[N].EVENT_ID	15-368
trigger.timer[N].clear()	15-369
trigger.timer[N].count	15-369
trigger.timer[N].delay	15-370
trigger.timer[N].delaylist	15-371
trigger.timer[N].EVENT_ID	15-372
trigger.timer[N].overrun	15-372
trigger.timer[N].passthrough	15-373
trigger.timer[N].reset()	15-373
trigger.timer[N].stimulus	15-374
trigger.timer[N].wait()	15-376
trigger.wait()	15-376
tsplink.group	15-377
tsplink.master	15-378
tsplink.node	15-378

tsplink.readbit()	15-379
tsplink.readport()	15-379
tsplink.reset()	15-380
tsplink.state	15-381
tsplink.trigger[N].assert()	15-381
tsplink.trigger[N].clear()	15-382
tsplink.trigger[N].EVENT_ID	15-383
tsplink.trigger[N].mode	15-383
tsplink.trigger[N].overrun	15-385
tsplink.trigger[N].pulsewidth	15-386
tsplink.trigger[N].release()	15-386
tsplink.trigger[N].reset()	15-387
tsplink.trigger[N].stimulus	15-388
tsplink.trigger[N].wait()	15-389
tsplink.writebit()	15-390
tsplink.writeport()	15-390
tsplink.writeprotect	15-391
tspnet.clear()	15-392
tspnet.connect()	15-393
tspnet.disconnect()	15-394
tspnet.execute()	15-395
tspnet.idn()	15-396
tspnet.read()	15-397
tspnet.readavailable()	15-398
tspnet.reset()	15-399
tspnet.termination()	15-399
tspnet.timeout	15-400
tspnet.tsp.abort()	15-401
tspnet.tsp.abortonconnect	15-401
tspnet.tsp.rhtablecopy()	15-402
tspnet.tsp.runscript()	15-403
tspnet.write()	15-404
userstring.add()	15-404
userstring.catalog()	15-405
userstring.delete()	15-406
userstring.get()	15-407
waitcomplete()	15-407

Common commands..... 16-1

Common command summary	16-1
Script command equivalents	16-2
Command reference	16-3
Identification query: *IDN?	16-3
Operation complete and query: *OPC and *OPC?	16-3
Reset: *RST	16-3
Self-test query: *TST?	16-4
Trigger: *TRG	16-4
Wait-to-continue: *WAI	16-4

Calibration 17-1

Verification	17-1
Calibration test requirements	17-2
Restoring factory defaults	17-4
Performing the calibration test procedures	17-4
Current source accuracy	17-6
Current measurement accuracy	17-8

Voltage source accuracy	17-9
Voltage measurement accuracy	17-11
Adjustment	17-12
Environmental conditions	17-12
Adjustment considerations	17-13
Calibration adjustment overview	17-15
Calibration commands quick reference	17-16
Adjustment procedure	17-18
Frequently asked questions	18-1
How do I find the serial number?	18-1
How do I optimize performance?	18-2
Disabling autozero to increase speed	18-2
How do I upgrade the firmware?	18-2
How do I use the digital I/O port?	18-3
How do I trigger other instruments?	18-3
Triggering a scanner	18-3
Interactive trigger programming	18-4
More information about triggering	18-4
How do I store measurements in nonvolatile memory?	18-4
When should I change the output-off state?	18-5
How do I make contact check measurements?	18-5
How can I change the line frequency or voltage?	18-5
Where can I get the LabVIEW driver?	18-6
Why is the reading value 9.91e37?	18-6
Next steps	19-1
Additional 2606B information	19-1

Introduction

In this section:

Welcome	1-1
Extended warranty	1-1
Contact information	1-1
2606B documentation	1-2
Product software and drivers.....	1-3
Capabilities and features.....	1-4
What you should have received	1-5
Instrument and module serial numbers	1-5
General ratings.....	1-6

Welcome

Thank you for choosing a Keithley Instruments product. The 2606B System SourceMeter® instrument provides manufacturers of electronic components and semiconductor devices with an instrument that combines source and measurement capabilities in a single instrument called a source-measure unit (also called a SMU). This combination simplifies test processes by eliminating synchronization and connection issues associated with multiple instrument solutions. A 2606B provides a scalable, high throughput, highly cost-effective solution for precision dc, pulse, and low frequency ac source-measure testing that also maintains code compatibility throughout the Series 2600B instruments.

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 Tektronix corporate headquarters (toll-free inside the U.S. and Canada only) at 1-800-833-9200. For worldwide contact numbers, visit tek.com/contact-us.

2606B documentation

This reference manual contains the documentation for the 2606B.

It is organized into the following sections:

- **Installation:** Installation information, including dimension diagrams and power on information.
- **Instrument description:** Descriptions of the front panel, rear panel, remote communications interfaces, and use of the virtual front panel.
- **Operation:** Descriptions of limits, circuit configurations, DUT connections, saving setups, USB storage, and instrument messages.
- **Sourcing and measuring:** Basic source-measure operations, connection of multiple SMUs, sweep operation, relative offset, filters, and high-capacitance mode.
- **Reading buffers:** How to use the reading buffers.
- **Triggering:** Describes the trigger model, event detection, timers, event blenders, the digital I/O port, and TSP-Link trigger lines.
- **Theory of operation:** Describes the internal circuitry and software of the 2606B.
- **LAN concepts and settings:** How to set up the 2606B for use on a local area network.
- **Status model:** Description of the 2606B status model.
- **Maintenance:** Information on instrument maintenance, including line fuse replacement and firmware upgrades.
- **Troubleshooting guide:** Description of the error queue and basic LAN troubleshooting.
- **Introduction to TSP operation:** Describes the basics of using remote commands to control the instrument, including descriptions of the factory scripts.
- **Instrument programming:** Describes how to control the instrument using TSP, TSB and TSB Embedded, TSP-Link system expansion, and TSP-Net.
- **TSP command reference:** Programming notes and an alphabetic listing of all commands available for the 2606B.
- **Common commands:** Descriptions of IEEE Std. 488.2 common commands.
- **Calibration:** How to verify and adjust the 2606B.
- **Frequently asked questions (FAQs):** Information that addresses commonly asked questions.
- **Next steps:** Contains sources of additional information.

Product software and drivers

Go to the [Product Support and Downloads web page](http://tek.com/product-support) (tek.com/product-support) to download drivers and software for your instrument.

Available drivers and software include:

- **KickStart Software:** Enables quick test setup and data visualization when using one or more instruments.
- **Test Script Builder (TSB):** This software provides an environment to develop a test program and the ability to load the test program onto the instrument. Running a program loaded on the instrument eliminates the need to send individual commands from the host computer to the instrument when running a test.
- **IVI-COM Driver:** An IVI instrument driver you can use to create your own test applications in C/C++, VB.NET, or C# programming languages. It can also be called from other languages that support calling a DLL or ActiveX (COM) object. Refer to [IVI Foundation](http://ivifoundation.org) (ivifoundation.org) for additional information.
- **LabVIEW™ Software drivers:** Drivers to communicate with National Instruments LabVIEW Software.
- **Keithley I/O layer:** Manages the communications between Keithley instrument drivers and software applications and the instrument itself. The I/O Layer handles differences in communications required to support GPIB, serial, ethernet, and other communications buses so that drivers and software applications do not need to handle the differences themselves.

To identify IP addresses of instruments that are connected to the local area network (LAN) and support the VXI-11 discovery protocol, you can also use LXI Discovery Tool, available from the [Resources](http://lxistandard.org/Resources/Resources.aspx) (lxistandard.org/Resources/Resources.aspx) page of the [LXI Consortium website](http://lxistandard.org) (lxistandard.org).

Capabilities and features

The 2606B has the following features:

- 4.5, 5.5, or 6.5 digit display resolution
- Resistance and power measurement functions
- Two-quadrant source operation
- Contact check function
- High-capacitance mode for load impedance up to 50 μ F
- Linear, logarithmic, and custom sweeping and pulsing
- Filtering to reduce reading noise
- Trigger model supports extensive triggering and synchronization schemes at hardware speeds
- Internal memory stores five user setup options
- Dedicated reading buffers that can each store and recall over 140,000 measurements; additional dynamic reading buffers can be created
- USB flash drive access for saving data buffers, test scripts, and user setups
- Digital I/O port: Allows the 2606B to control other devices
- Web-based characterization tool that provides easy access to data gathering, sweeping, and pulsing features
- LXI® version 1.4 Core 2011 compliance
- Embedded TSP scripting engine accessible from any host interface; responds to high-speed test scripts comprised of instrument control commands
- TSP-Link® expansion bus that allows TSP-enabled instruments to trigger and communicate with each other; advanced Test Script Processor (TSP®) scripting engine features enable parallel script execution across the TSP-Link network
- Supports Universal Serial Bus (USB) and ethernet local area network (LAN) connections

What you should have received

In addition to the 2606B, you should have received the items listed in the following table.

Part number	Description	Quantity
CA-180-16	LAN crossover cable, 0.41 m (16 in.)	2
174710700	Shielded crossover cable with RJ-45 connector, 1.5 m (5 ft.)	1
Depends on country	Power line cord	1
CA-568	Green and yellow ground cable, 3 m (120 in.)	1
407612600	1U rack rail mount (part of the Model 4299-13 Rack Mount Kit)	2
10-32X3/8PHTRSH	10-32X3/8 Phillips truss head screw (part of the Model 4299-13 Rack Mount Kit)	4
10-32X5/8PHTRSH	10-32X5/8 Phillips truss head screw (part of the Model 4299-13 Rack Mount Kit)	8
FA-274	Cage nut (part of the Model 4299-13 Rack Mount Kit)	4
7709-308A	25-pin D-shell connector kit	2
0713411XX	Safety Precautions	1
CS-1195-8	Eight-pin custom cable connector (refer to Keithley document number PA-907 for instruction)	4
CS-1196-8	Cable housing (refer to Keithley document number PA-907 for instruction)	4

Instrument and module serial numbers

Each 2606B has three serial numbers, one for the instrument and one for each module.

The instrument serial number is on a label on the front panel. This serial number is not available through a remote interface.

The serial numbers for the modules are on labels on the rear panel. The serial number for a specific module can be accessed through the virtual front panel or by using the [localnode.serialno](#) (on page 15-161) command over a remote interface.

To access the serial number from the virtual front panel:

1. If the 2606B is in remote operation, press the **EXIT (LOCAL)** key once to place the instrument in local operation.
2. Select the **MENU** key.
3. Use the navigation wheel to scroll to the SYSTEM-INFO menu item.
4. Press the **ENTER** key. The SYSTEM INFORMATION menu is displayed.
5. Scroll to the **SERIAL#** menu item.
6. Press the **ENTER** key. The serial number for the module is displayed.

General ratings

The 2606B general ratings and connections of the instrument are listed in the following table.

Category	Specification
Power supply	100 V ac to 240 V ac, 50 Hz or 60 Hz (autosensing); 425 VA maximum
Input and output connections	See Front panel (on page 3-1) and Rear panel (on page 3-2)
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/°C, 35 °C to 50 °C Storage: -25 °C to 65 °C Pollution degree: 1 or 2

Installation

In this section:

Installing the 2606B.....	2-1
Cooling vents	2-1
Dimensions	2-2
Turning the instrument on and off	2-4
Placing the 2606B in standby.....	2-5
Warmup period.....	2-5
Line frequency configuration	2-6
System information	2-6

Installing the 2606B

The 2606B is intended for mounting in a rack only. For detailed instructions, refer to the documentation for the Model 4299-13 Rack Mount Kit (part number 0713574XX).

CAUTION

Rack mounting the 2606B requires two people. Failure to recognize and observe standard safety precautions could result in personal injury.

Cooling vents

The 2606B has front and side intakes and rear exhaust vents. All sides must be unobstructed to allow for air flow and to dissipate heat.

Excessive heat could damage the 2606B and degrade its performance. Only operate the 2606B in an environment where the ambient temperature does not exceed 50 °C.

CAUTION

To prevent damaging heat build-up and ensure specified performance, use the following guidelines.

- The rear exhaust vents and both side intake vents, and the front intake vents must be unobstructed to properly dissipate heat. Even partial blockage could impair proper cooling.
 - Do not position any devices adjacent to the 2606B that force air (heated or unheated) toward its cooling vents or surfaces. This additional airflow could compromise accuracy.
 - When rack mounting the 2606B, make sure there is adequate airflow around both sides of the instrument to ensure proper cooling. Adequate airflow enables air temperatures within approximately one inch of the 2606B surfaces to remain within specified limits under all operating conditions.
 - The front and sides of the 2606B are cool air intakes. It is recommended that there is $23\text{ }^{\circ}\text{C} \pm 5\text{ }^{\circ}\text{C}$ air available at these locations. The rear of the 2606B is the exhaust.
 - If you rack mount high power-dissipation equipment next to the 2606B, it could cause excessive heating. To produce specified 2606B accuracies, maintain the specified ambient temperature around the intake vents of the 2606B. In rack configurations with convection cooling only, which is not recommended, proper cooling practice places the hottest non-precision equipment (for example, the power supply) at the top of the rack away from and above precision equipment (such as the 2606B).
 - Mount precision equipment as low as possible in the rack, where temperatures are coolest.
-

Dimensions

The following figures show the instrument dimensions.

Figure 1: 2606B dimensions - top view

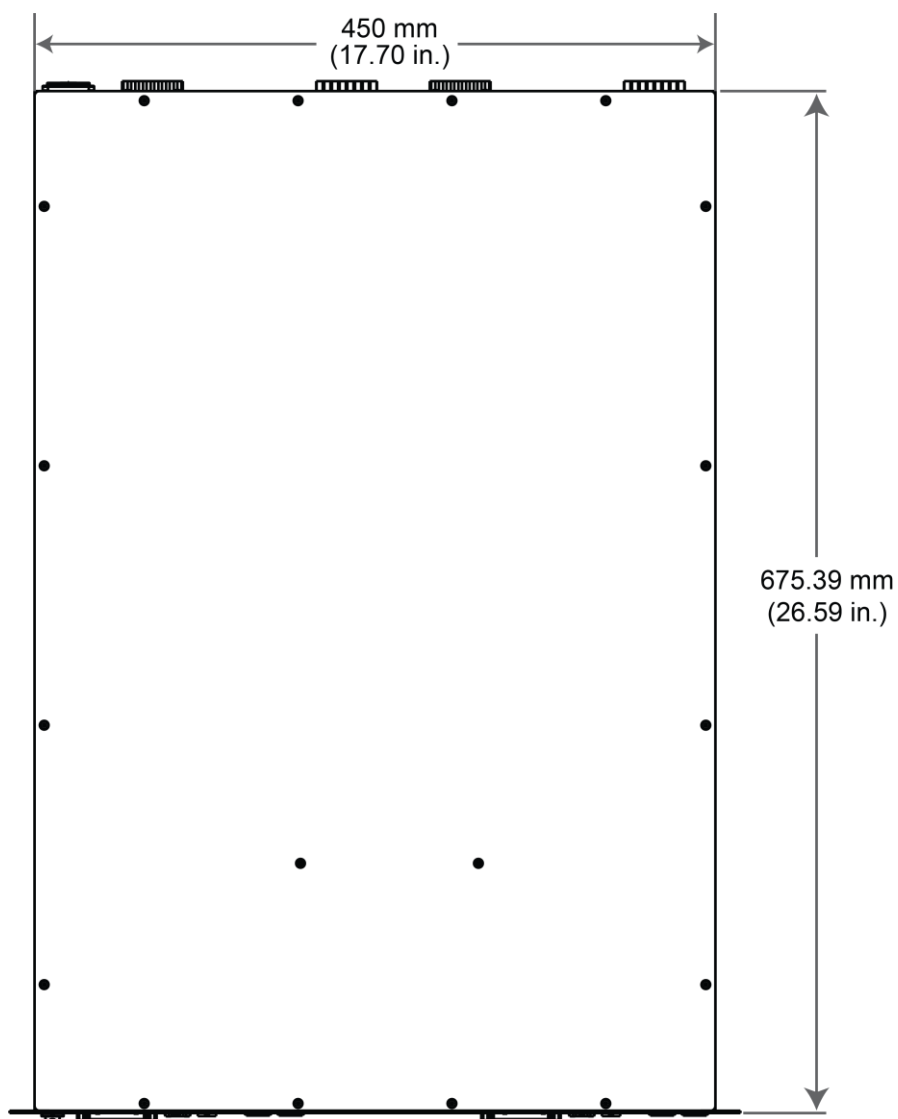


Figure 2: Dimensions - front view

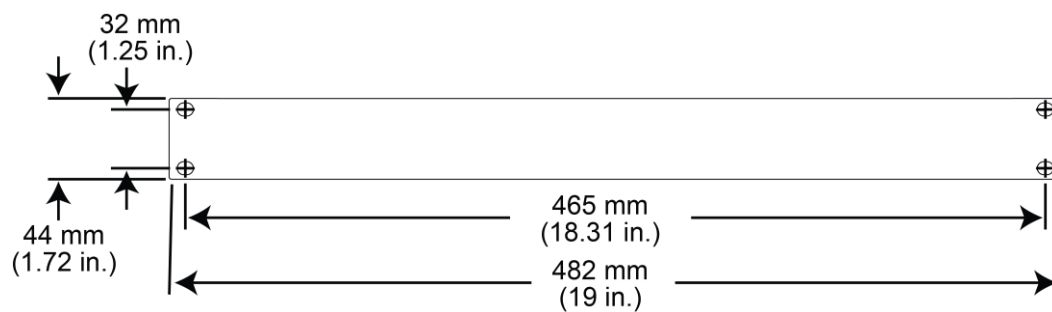
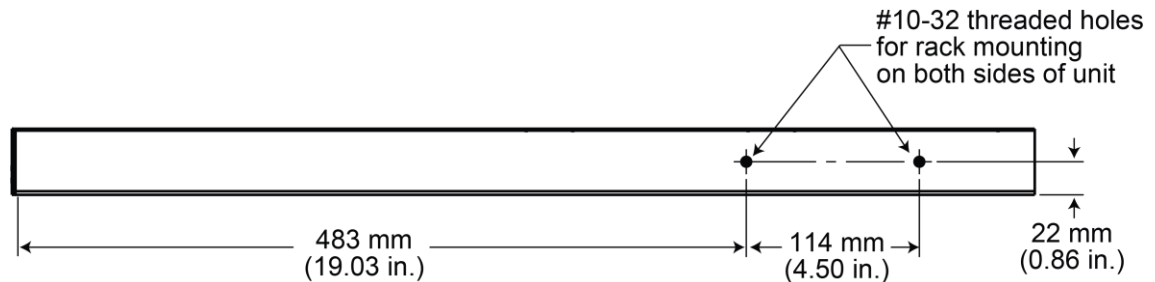


Figure 3: Dimensions - side view



Turning the instrument on and off

The 2606B operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz or 60 Hz. Line voltage is automatically sensed (there are no switches to set). Make sure the operating voltage in your area is compatible.

Follow the procedure below to connect the 2606B to line power and turn on the instrument.

CAUTION

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

To turn a 2606B on and off:

1. Before plugging in the power cord, make sure that the front-panel POWER switch is in the off (O) position.
2. Connect the socket end of the supplied power cord to the ac receptacle on the rear panel.
3. Connect the other end of the power cord to a grounded ac outlet.

⚠ WARNING

The power cord supplied with the 2606B 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 chassis ground 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.

WARNING

Hazardous voltages may be present in the test system. To prevent injury or death, remove power from the instrument or test system and discharge any energy storage components (for example, capacitors or cables) before changing any connections that might allow contact with an uninsulated conductor.

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.

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

Placing the 2606B in standby

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 2606B 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 2606B before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

When the instrument is on, you can place the output in an active output state (output on) or a standby mode (output off).

To place the output in standby, send the command:

```
smuX.source.output = 0
```

Even though the instrument is placed in standby, the output may not be actually off.

Warmup period

The 2606B must be turned on and allowed to warm up for at least two hours to achieve rated accuracies.

Line frequency configuration

The factory configures the 2606B to detect the power line frequency automatically at each power-up. This detected line frequency (either 50 Hz or 60 Hz) is used for aperture (NPLC) calculations.

In noisy environments, you can manually configure the instrument to match the actual line frequency.

To configure the line frequency from the virtual front panel:

1. Press the **MENU** key, then turn the navigation wheel to select **LINE-FREQ**, and then press the **ENTER** key.
2. Turn the navigation wheel to select the appropriate frequency and then press the **ENTER** key. To configure the instrument to automatically detect line frequency at each power-up, select **AUTO**.
3. Press the **EXIT (LOCAL)** key to return to the main display.

To configure the line frequency from a remote interface:

Set the `localnode.linefreq` or the `localnode.autolinefreq` attribute. To set the line frequency to 60 Hz, send:

```
localnode.linefreq = 60
```

To configure the instrument to automatically detect line frequency at each power-up:

```
localnode.autolinefreq = true
```

System information

You can retrieve serial number, firmware revision, calibration dates, and memory usage from the instrument.

To view the system information from the virtual front panel:

1. Press the **MENU** key.
2. Select **SYSTEM-INFO**.
3. Select one of the following:
 - **FIRMWARE**
 - **SERIAL#**
 - **CAL**
 - **MEMORY-USAGE**

To retrieve system information from a remote interface:

To retrieve the firmware revision and serial number, send the `*IDN?` query (see [Identification query: *IDN?](#) (on page 16-3) for more information).

To determine memory usage, see the [meminfo\(\)](#) (on page 15-163) function.

To determine when calibration was last run, see [smuX.cal.date](#) (on page 15-197).

To determine when calibration is due, see [smuX.cal.due](#) (on page 15-198).

Instrument description

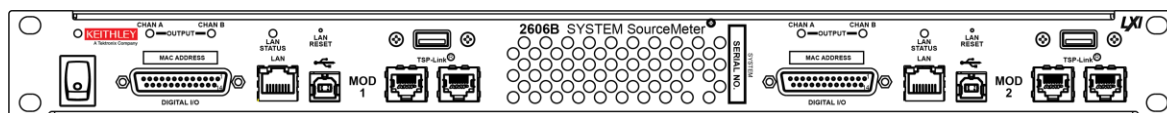
In this section:

Front panel	3-1
Rear panel.....	3-2
Remote communications interfaces	3-3
Using the web interface.....	3-15
Virtual Front Panel	3-24
Beeper	3-33
Display mode	3-34
Display operations.....	3-35

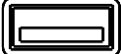
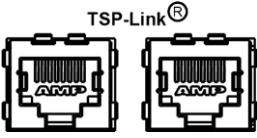
Front panel

The front panel of the 2606B is shown below. The descriptions of the front-panel controls, USB port, and indicators follow the figure.

Figure 4: 2606B front panel



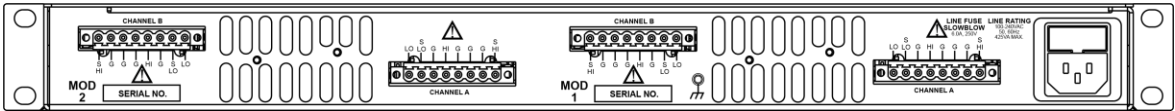
Power indicator and switch		The power switch turns the instrument on or off. The indicator is illuminated when the instrument is on. The power switch powers both modules.
Digital I/O port		DB-25 socket. Use a cable equipped with a DB-25 plug (L-com part number CSMN25MF-5). Pins provided: Fourteen digital input or output pins, seven GND pins, three +5 V pins, and an output enable pin.
OUTPUT	CHAN A CHAN B 	Indicates if the source output is on or off. The source output for the channel is on when the LED is illuminated (blue).
LAN STATUS indicator		Illuminates when the instrument is connected to a local area network (LAN).
LAN port		RJ-45 connector for a local area network (LAN). The LAN interface supports Auto-MDIX, so you can use either a CAT-5e crossover cable or a normal CAT-5e straight-through cable. See LAN communications (on page 3-9).

LAN RESET		Reverts the LAN settings and the instrument password to default values. Insert a straightened paper clip into the hole above LAN RESET to reset the LAN. See LAN communications (on page 3-9).
USB Type B port		USB Type B connection for communication, control, and data transfer. For details, see USB communications (on page 3-4).
USB Type A port		USB Type A connection. You can use a USB flash drive to store reading buffer data, scripts, and user setups. You can also use it to upgrade the firmware.
TSP-Link ports		Connections for the 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 14-52).

Rear panel

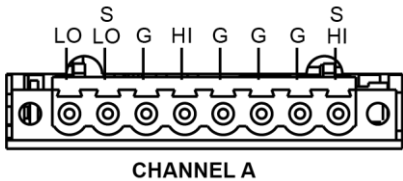
The rear panel of the 2606B is shown below. The descriptions of the rear-panel components follow the figure.

Figure 5: 2606B rear panel

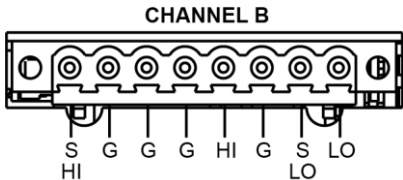


Channel connector

Channel A



Channel B



This connector provides input/output connections for HI and LO, sense (S HI/S LO), and guard (G). Connections are as follows:

- LO = LO
- S LO = Sense LO
- G = Guard
- S HI = Sense HI
- HI = HI

Ground

Chassis ground



Ground terminal for connecting output HI or LO to chassis ground. This provides a connection terminal to the equipment frame.

Line fuse and power receptacle

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 [Turning the instrument on and off](#) (on page 2-4) and [Line fuse replacement](#) (on page 11-1).

Remote communications interfaces

You can choose from one of several communication interfaces to send commands to and receive responses from the 2606B.

You can control the 2606B 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 2606B automatically detects the type of communications interface when you connect to the respective port on 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.

The 2606B supports the following remote interfaces:

- **LAN.** Local area network (LAN) communications provide the flexibility to build scalable and functional test or data acquisition systems with a large degree of flexibility.
- **USB.** Communicate with the instrument over a USB connection.

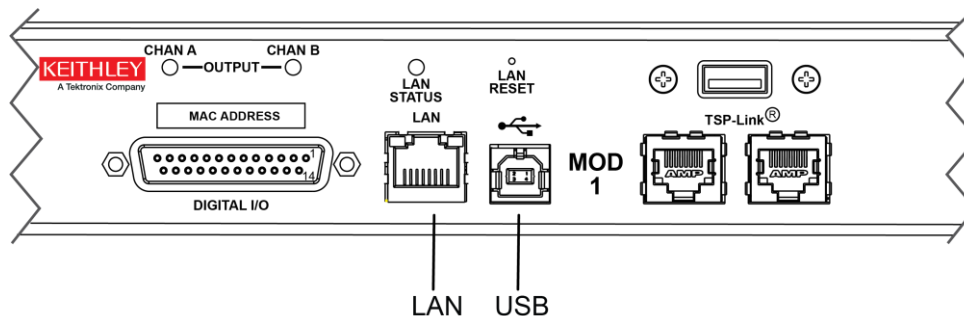
For more information about the remote interfaces, see:

- [LAN concepts and settings](#) (on page 9-1)
- [USB communications](#) (on page 3-4)

Remote interface connections

The remote interface connections are on the front panel. Each module has its own set of connections.

Figure 6: Remote interface connections



Output queue

Response messages, such as those generated from print commands, are placed in the output queue. All remote command interfaces share the same output queue.

The output queue sets the message available (MAV) bit in the status model.

The data in the output queue is cleared by the *CLS command.

USB communications

To use the front-panel USB port, you need a driver that communicates using the USB Test and Measurement Class (USBTMC) protocol, such as NI-VISA, on the host computer. See [Software components for applications and instrument drivers](#) (on page 3-10) for more information.

When installed, the USBTMC protocol 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 driver automatically detects the device. Note that the driver 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.

Communicate with the instrument

When using Virtual Instrument Software Architecture (VISA) to communicate with the USB device, you need to use a VISA communication driver. VISA requires a resource string in the following format to connect to the correct USB instrument:

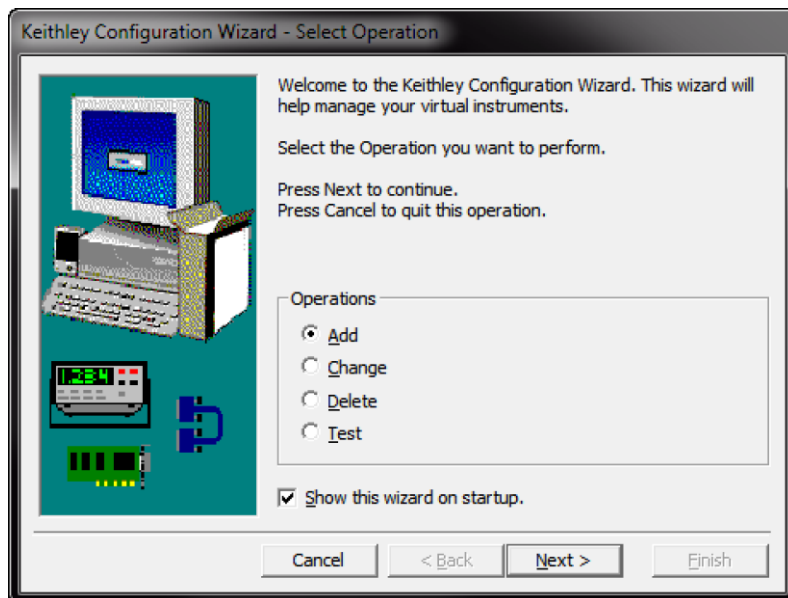
```
USB[board]::manufacturer ID::model code::serial number[::USB interface number] [::INSTR]
```

This requires that you determine the parameters. You can gather this information by running a utility that automatically detects all instruments connected to the computer. If you installed the Keithley I/O Layer, the Keithley Configuration Panel is available from the Microsoft® Windows® Start menu in the Keithley Instruments menu.

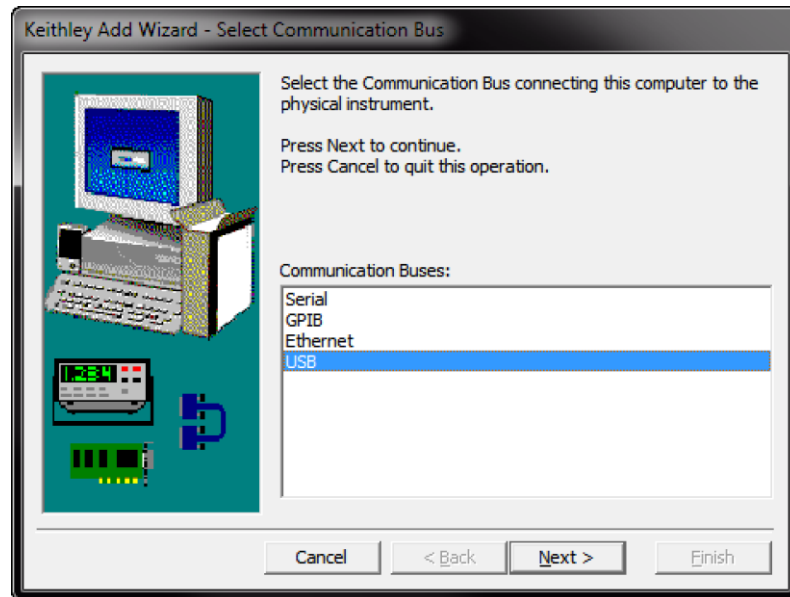
To use the Keithley Configuration Panel to determine the VISA resource string:

1. Start the Keithley Configuration Panel. The Select Operation dialog box is displayed.
2. Select **Add**.

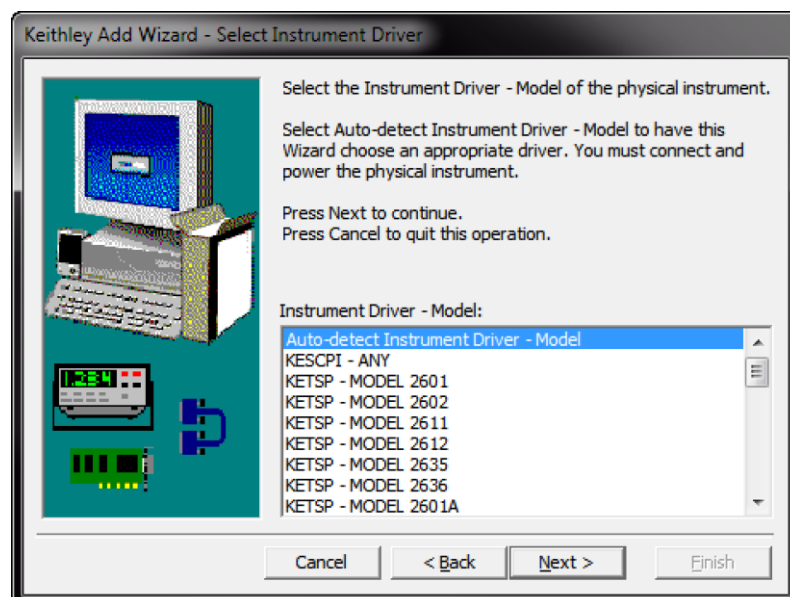
Figure 7: Select Operation dialog box



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

Figure 8: Select Communication Bus dialog box

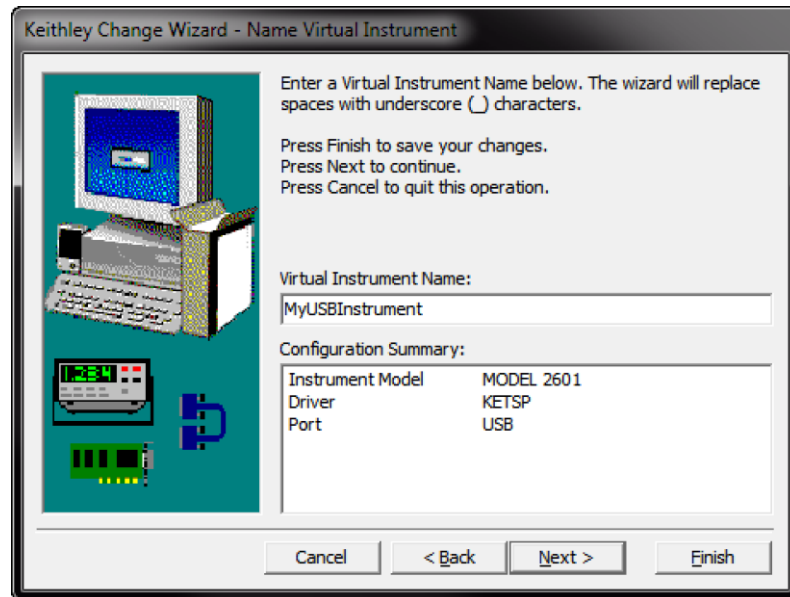
4. Select **USB**.
5. Select **Next**. The Select Instrument Driver dialog box is displayed.

Figure 9: 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 highlighted.

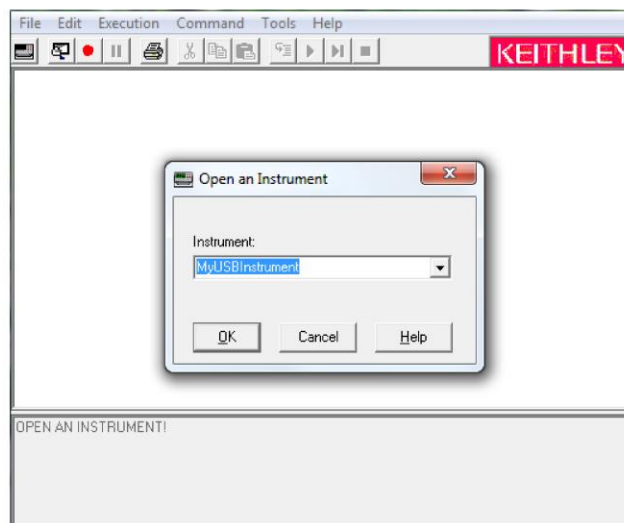
8. Select **Next**. The Name Virtual Instrument dialog box is displayed.

Figure 10: Name Virtual Instrument dialog box



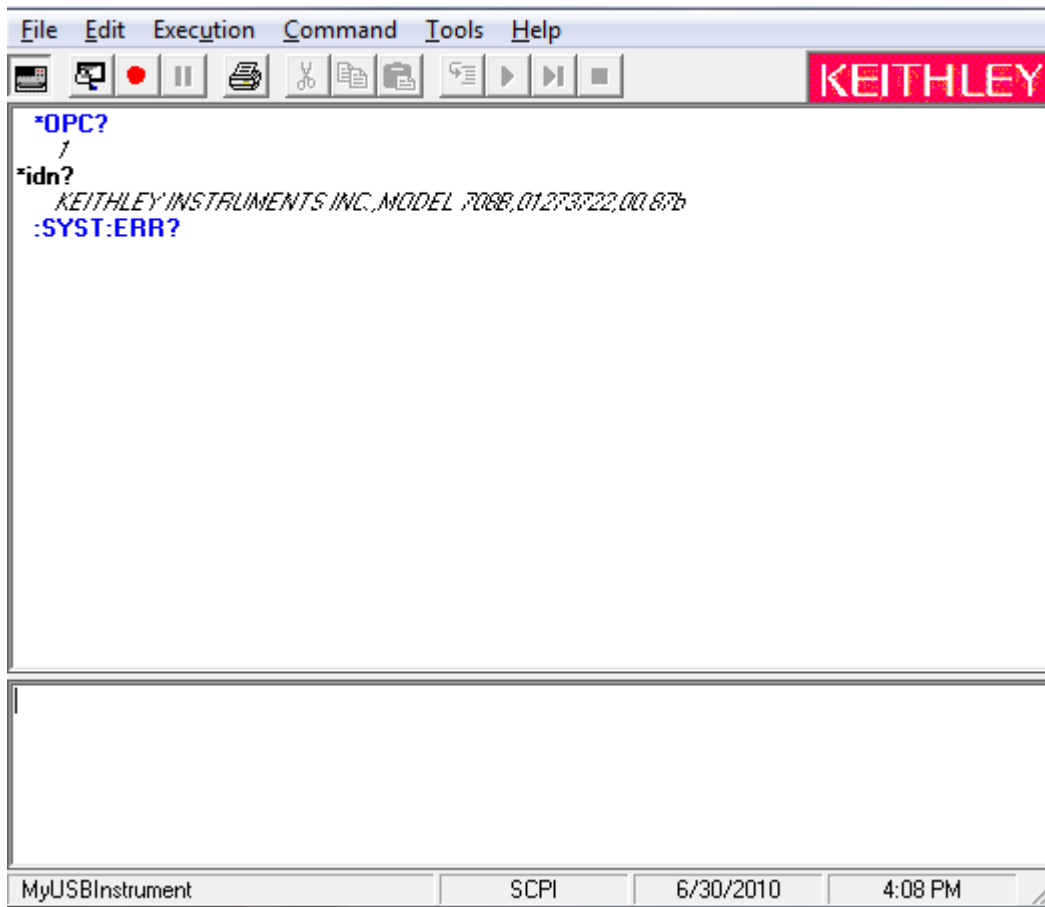
9. In the **Virtual Instrument Name** box, enter a name that you want to use to refer to the instrument.
10. Select **Finish**.
11. Select **Cancel** to close the wizard.
12. Save the configuration. From the Configuration Utility, select **File > Save**.
13. In the Keithley Communicator, select **File > Open Instrument** to open the instrument you just named.

Figure 11: Keithley Communicator Open an Instrument



14. Select **OK**.
15. Send a command to the instrument and see if it responds.

Figure 12: Send a command to the instrument



NOTE

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

If you have the Keysight IO Libraries on your system, you can run Keysight Connection Expert to check out your USB instruments. See their documentation for information.

Connecting multiple USB instruments to the computer

The most convenient way to connect USB instrumentation to the computer is to plug a USB cable directly from the instrument to the computer. If you have more than one USB instrument or have other USB devices, such as printers, keyboards, and mice, you might not have enough USB connectors on the computer.

To gain more ports, you can use a USB hub or add more USB controller cards if you have available PCI or PCI Express slots.

LAN communications

The 2606B is an LXI version 1.4 Core 2011 compliant instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet). The LAN ports for the modules are on the front panel of the 2606B and support full connectivity on a 10 Mbps or 100 Mbps network.

Through the LAN interface, you can use a web interface to control the instrument.

Once the 2606B is connected to the LAN, you can use the LXI™ Discovery Tool to identify the IP addresses of instruments that are connected to the LAN and that support the VXI-11 discovery protocol. You can also manually set up the IP address.

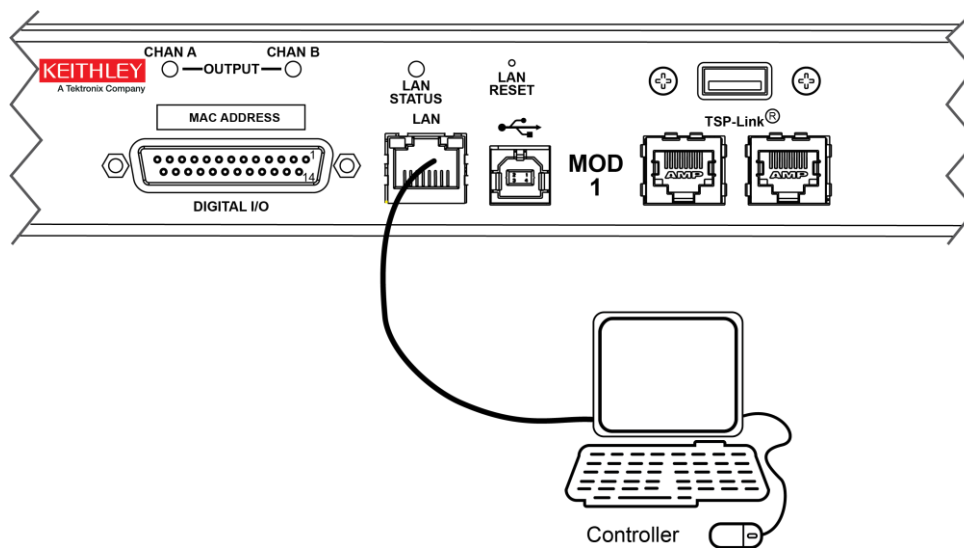
For detailed information about setting up your LAN interface, refer to [LAN concepts and settings](#) (on page 9-1).

LAN cable connection

Use a LAN crossover cable (RJ-45, male to male) or a straight-through cable to connect your equipment. The 2606B includes a cable that you can use for the TSP-Link® network or LAN communications. The instrument automatically senses which cable you have connected.

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

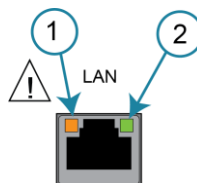
Figure 13: LAN connection



LAN status LEDs

The figure below illustrates the two status light-emitting diodes (LEDs) that are on the LAN port of the instrument. The table below the figure provides explanations of the LED states. The LED labeled 1 indicates the LAN port is connected to a 100 Mbps network. The LED labeled 2 indicates the LAN port is connected to a 10 Mbps network.

Figure 14: LAN status



When an LED is:	The network:
Off	is not connected
On	is connected
Blinking	is sending or receiving data

Monitoring the LAN

The `lan.autoconnect` command configures the instrument to monitor the LAN for lost connections. All ethernet connections are disconnected if the LAN link is disconnected for longer than the time-out value specified in the `lan.linktimeout` attribute.

For detail on these commands, refer to the following command descriptions:

- [lan.autoconnect](#) (on page 15-125)
- [lan.linktimeout](#) (on page 15-133)

Software components for applications and instrument drivers

The majority of software applications and all instrument drivers from Keithley Instruments depend on the following software components:

- NI-VISA
- VISA shared components
- IVI shared components
- NI CVI runtime engine
- NI IVI compliance package
- Keithley instrument driver

These software components are available for download from the [Product Support and Downloads web page](#) (tek.com/product-support).

Instrument driver types

There are several different styles of instrument drivers. Keithley Instruments provides the following instrument drivers for the 2606B:

- A native LabVIEW driver
- An IVI-C driver
- An IVI-COM driver

You need to pick the style that best suits the application development environment (ADE) that you are using. For example, if you are using LabVIEW, pick a native LabVIEW driver. If a native LabVIEW driver is not available, you can use an IVI-C driver because LabVIEW has the option of creating a wrapper for the IVI-C driver.

LabVIEW supports IVI-COM drivers, but they are not preferred. However, if they are the only driver types available for the instrument, they can be used.

If LabWindows/CVI or C/C++ is your programming language, an IVI-C driver is the best option. For Microsoft® Visual Basic® 6.0 and any .NET language (C#, VB.NET, and so on), an IVI-COM driver is the best option.

Sometimes instrument vendors do not provide all three driver types. Most languages can accommodate other driver types, but this is not optimal.

The following sections describe the different driver types in more detail.

VXIIPnP drivers

VXI`plug&play` (VXIIPnP) style drivers are Win32 DLLs that have some standard functions defined by the [IVI Foundation](http://ivifoundation.org) (ivifoundation.org), such as:

- init
- close
- error_message
- reset
- self_test
- read
- initiate
- fetch
- abort

The application programming interface (API) was defined so that users of instruments have a familiar API from instrument to instrument. There are some basic guidelines when creating APIs for your instrument, such as using VISA data types and how to construct the CVI hierarchy.

LabVIEW drivers

Native LabVIEW drivers

A native LabVIEW™ driver is a LabVIEW driver that is created using entirely built-in LabVIEW VIs. It does not make any calls to external DLLs or Library files. This makes the driver portable to all the platforms and operating systems that LabVIEW and VISA supports (such as Linux® on x86, Mac OS® X, and Microsoft® Windows®).

LabVIEW driver wrappers

All IVI-C drivers have a function panel file (file name extension .fp) that shows a hierarchy of the function calls into a DLL. It is a tool that guides a user to select the correct function call in the driver, because a DLL only has a flat API entry point scheme (unlike COM or .NET).

Any CVI-generated .fp files can be imported into LabVIEW and LabVIEW generates a wrapper for the DLL. The drawback here is that the driver is dependent on the DLL, which is not portable and is therefore specific to the Windows operating system.

Getting instrument drivers

To see what drivers are available for your instrument:

1. Go to the [Product Support and Downloads web page](http://tek.com/product-support) (tek.com/product-support).
2. Enter the model number of your instrument.
3. Select **Software** from the list.
4. Select and download the file.

For LabVIEW™, you can also go to the National Instruments website and search their instrument driver database.

Instrument driver examples

All Keithley drivers come with examples written in several programming languages that show you how to do common tasks with the instruments. The examples are available in the drivers and through Test Script Builder.

IVI shared components

The IVI shared components are similar in concept to the VISA shared components. The IVI Foundation provides class drivers for:

- All the supported instruments (DMM, Scope, Fgen, and so on)
- The configuration store

The IVI shared components also create the installation folders and registry keys that all IVI drivers and support files use for installation.

Interchangeable Virtual Instruments (IVI) style drivers

The [IVI Foundation](http://ivifoundation.org) (ivifoundation.org) defined a set of application programming interfaces (APIs) for instruments, including digital multimeters, arbitrary waveform/function generators, dc power supplies, ac power supplies, oscilloscopes, switches, spectrum analyzers, RF signal generators, and power meters.

There are two types of IVI drivers, IVI-COM and IVI-C. IVI-COM drivers use Microsoft® COM technology to expose driver functionality. IVI-C drivers use conventional Microsoft® Windows® DLLs to export simple C-based functions.

For more information about IVI drivers and the differences between the COM, C, and .NET interfaces, see [Understanding the Benefits of IVI](http://ivifoundation.org/docs/Understanding%20the%20Benefits%20of%20IVI[3].pdf) ([ivifoundation.org/docs/Understanding%20the%20Benefits%20of%20IVI\[3\].pdf](http://ivifoundation.org/docs/Understanding%20the%20Benefits%20of%20IVI[3].pdf)).

NI CVI runtime engine

IVI-C drivers that are created using the National Instruments (NI™) LabWindows/CVI environment depend on either the CVI runtime DLL (`cvirte.dll`) or the instrument support runtime DLL (`instrsup.dll`). These DLLs must be present on the system for them to run.

NI IVI Compliance Package

The National Instruments (NI™) IVI Compliance Package (ICP) is a software package that contains IVI class drivers and support libraries that are needed for the development and use of applications that leverage IVI instrument interchangeability. The IVI Compliance Package also is based on and is compliant with the latest version of the instrument programming specifications defined by the IVI Foundation.

The NI ICP installer installs the IVI shared components, CVI runtime engine, and the instrument support runtime engine.

Keithley I/O layer

The Keithley I/O Layer (KIOL) is a software package that contains several utilities and drivers. It is mainly used as a supplement to IVI drivers or application software like Test Script Builder (TSB).

For additional detail on the Keithley I/O layer, including computer requirements and installation instructions, see the tek.com/keithley website.

NI-VISA Runtime

NI-VISA is National Instruments (NI™) implementation of the VISA standard. The Keithley I/O Layer (KIOL) contains a licensed version of the NI-VISA Runtime Engine that contains only the binary files (DLLs) that allow the NI-VISA drivers to operate.

If you already have NI software (such as LabVIEW™ or LabWindows™) installed, you have a valid license that can be used with Keithley drivers and application software.

If you do not have NI software installed, you must install the Keithley I/O layer to install the drivers.

Keithley Configuration Panel

The Keithley Configuration Panel is a configuration utility for IVI drivers, similar to NI-MAX. It can also autodetect USBTMC instruments and LAN instruments that support the VXI-11 protocol.

Keithley Communicator

The Keithley Communicator is a dumb terminal program that uses VISA to communicate with the instrument.

How to install the Keithley I/O Layer

NOTE

Before installing, it is a good practice to check the [Product Support and Downloads web page \(tek.com/product-support\)](http://tek.com/product-support) to verify that you have the latest version of the Keithley I/O Layer.

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

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

1. Download the Keithley I/O Layer Software from the Product Support and Downloads web page. The software is a single compressed file.
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.

Using the web interface

If you are connected to the 2606B using LAN communications, you can use an internet browser to connect to the instrument through the 2606B web interface.

The LXI web interface allows you to change settings and control your instrument through the web interface. The web interface includes:

- A virtual front panel that you can use to control the instrument.
- TSB Embedded, a script management tool that can also be used to send individual commands to the instrument.
- The instrument status, model, serial number, firmware revision, and the last LXI message.
- An ID button to help you locate the instrument.
- Ability to download data from specific reading buffers into a `.csv` file.
- Administrative options, LAN information, and LXI information.

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

You can also upgrade the instrument from the web interface. Refer to [Upgrading the firmware](#) (on page 11-2) for instructions.

NOTE

To set up LAN connections, refer to [LAN communications](#) (on page 3-9).

To set up the LAN connections and find the IP address, refer to [LAN concepts and settings](#) (on page 9-1).

Once the 2606B is connected to the LAN, you can use the LXI™ Discovery Tool to identify the IP addresses of instruments that are connected to the LAN and that support the VXI-11 discovery protocol. The LXI Discovery Tool is available from the Resources tab of the LXI Consortium website.

How to access the web interface

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 12-6).

Web interface Welcome page

The Welcome page provides information about the instrument. It includes:

- The instrument model number, manufacturer, firmware revision number, and instrument serial number.
- LXI information.
- LAN connection information.
- The TCP Raw Socket number and Telnet Port number.
- Instrument address strings.
- Calibration dates.
- The ID button, which helps you find the instrument if you have multiple instruments.

Figure 15: Web interface Welcome page

LXI Web Interface - Welcome Page	
Instrument Model:	2612B SYSTEM SourceMeter®
Manufacturer:	Keithley Instruments
Firmware Revision:	3.4.0
Serial Number:	01000000
Description:	Keithley Instruments SMU 2612B-01000000
LXI Extended Functions:	
LXI Version:	1.4 LXI Core 2011
MAC Address:	98:9B:11:10:8F:00
TCP/IP Address:	192.168.0.1
Gateway:	0.0.0.0
Subnetmask:	255.255.0.0
Lan Connection:	100Mbps, Full Duplex
Host Name (FQDN):	192.168.0.1:43512B-01000000.local
DNS Servers:	
Ports:	DST: 5030 Raw Socket: 5025 Telnet: 23 VXI-11 Core: 1024
Instrument Address String:	ASRL[board]: INSTR CPRI[board]: 26 INSTR TCP[board]: 192.168.0.1:43512B-01000000 TCP[board]: 192.168.0.1:43512B-01000000 USB[board]: 0x160C:0x0010:01000000:43512B
Smu A Calibration Date:	Mon Feb 22 18:27:12 2016
Smu B Calibration Date:	Mon Feb 22 18:27:12 2016
Smu A Calibration Due:	
Smu B Calibration Due:	

Use the ID button to identify the module

You can select the ID button to determine which module you are communicating with.

The ID button is below the menu on every page of the web interface.

To identify the module:

1. On the Home page, select the **ID** button. The button turns green and the LAN STATUS indicator on the module blinks.
2. Select the **ID** button again to return the button to its original color and return the LAN STATUS indicator to steady on.

Change the IP configuration through the web interface

You can change the LAN settings through the web interface on the instrument. The LAN settings you can change include the IP address, subnet mask, gateway, and DNS address.

If you change the IP address through the web interface, the web interface 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 an IP address assignment that uses a static address to an IP address assignment that uses a DHCP server. If this happens, you need to revert to either using the virtual 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 using TSP commands. See [LAN concepts and settings](#) (on page 9-1) for information.

To change the IP configuration using the instrument web page:

1. Access the internal web page as described in [How to access the web interface](#) (on page 3-16).
2. From the navigation bar on the left, select **IP Configuration**.
3. Select **Modify**. The Modify IP Configuration page is displayed.

Figure 16: 2606B Modify IP Configuration screen

LXI - Keithley Instruments - 2606B - Modify IP Configuration	
Host Name:	k-2606B-1398407
Description:	Keithley Instruments SMU 2606B - 1398407
TCP/IP Configuration Mode:	☒ Automatic ☐ Manual
Static IP Address:	
Subnet Mask:	
Default Gateway:	0.0.0.0
DNS Verify:	☒ Enabled ☐ Disabled
Dynamic DNS:	☒ Enabled ☐ Disabled
mDNS Discovery:	☒ Enabled ☐ Disabled
DNS Servers:	
Domain Name:	
Speed:	☒ 100 Mbps ☐ 10 Mbps
Duplex:	☒ Full ☐ Half
Submit	

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.

Set the instrument password

You can change the instrument password from the web interface. The instrument password is used for access to the instrument from any remote interface. When password usage is enabled by the `localnode.passwordmode` command, you must supply a password to change the configuration or to control an instrument from a remote command interface.

NOTE

When a password is set for the web interface and `localnode.passwordmode` is set to an option that requires a password, you cannot make changes using the web interface options Virtual Front Panel, Flash Upgrade, and TSB Embedded.

The default password is "" (no password).

You cannot retrieve a lost password from any command interface. You can reset the password from the front panel. Select **MENU**, then **RESET-PASSWORD**.

For information on the TSP command, refer to [localnode.passwordmode](#) (on page 15-157).

To change the password:

1. Select **Set Password**.
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**.

View buffer data using the web interface

You can view the data in the reading buffers on the Reading Buffers page of the Virtual Front Panel web page.

To view the data, at the top of the virtual front panel, select **Reading Buffers**. On the Reading Buffers page, select a reading buffer from the Reading Buffer list. The data is displayed in the table.

Initially, the reading number and reading measurement values are displayed. You can display additional data using the filters to the right of the table. You can choose:

- **Measurement:** Include or exclude measurement information.
- **Source:** Include or exclude source information.
- **Other:** Include or exclude status, status detail, and timestamp information. For detail on the Status value, refer to [Buffer status](#) (on page 6-13).

An example of the Reading Buffers page with the measurement and source functions displayed is shown in the following figure.

Figure 17: Web interface Reading Buffers page

Virtual Front Panel Reading Buffers

Buffer : smua.nvbuffer1

Reading Buffer	Reading Number	Reading	Measure Function	Source Function
smua.nvbuffer1	1	0.000986814	Voltage	Voltage
smua.nvbuffer2	2	0.00098753	Voltage	Voltage
smub.nvbuffer1	3	0.000985861	Voltage	Voltage
smub.nvbuffer2	4	0.000985384	Voltage	Voltage
	5	0.000984907	Voltage	Voltage
	6	0.000985622	Voltage	Voltage
	7	0.000985622	Voltage	Voltage
	8	0.000986814	Voltage	Voltage
	9	0.000984669	Voltage	Voltage
	10	0.000985384	Voltage	Voltage
	11	0.000985384	Voltage	Voltage
	12	0.000985861	Voltage	Voltage
	13	0.000985861	Voltage	Voltage
	14	0.00098443	Voltage	Voltage
	15	0.000985861	Voltage	Voltage
	16	0.000984907	Voltage	Voltage
	17	0.000984907	Voltage	Voltage
	18	0.000985861	Voltage	Voltage
	19	0.000985384	Voltage	Voltage

Measurement: ☒ Functions, ☐ Ranges

Source: ☒ Functions, ☐ Ranges, ☐ Values, ☐ Output, ☐ States

Other: ☐ Status, ☐ Status, ☐ Details, ☐ Timestamp

Save

Download reading buffer data using the web interface

You can download reading buffer data to a .csv file from the Reading Buffers page of the web interface.

The data shown in the table on the Reading Buffers page is downloaded.

To download buffer data:

1. From the web interface, select **Virtual Front Panel**.
2. Select **Reading Buffers**.
3. From the Reading Buffer list, select the reading buffer that contains the data you want to download.
4. From the options on the right, filter the reading buffer list to display the data you want to download.
5. Select **Save**. The file is downloaded to your computer.
6. Follow the instructions for your browser to open the file. Typically, the file opens in Microsoft Excel.

Use TSB Embedded

TSB Embedded is a script management tool that is available through the web interface of the instrument. You can use TSB Embedded to create, modify, and save test scripts, and to send individual commands. TSB Embedded provides some of the features of Test Script Builder (TSB). TSB is a software tool that simplifies building test scripts for Keithley Instruments that are enabled to use the Test Script Processor (TSP®) scripting engine.

For more information on TSB Embedded, refer to [Working with TSB Embedded](#) (on page 14-38).

Flash Upgrade

Allows you to upgrade the firmware for the module from the web interface. See [Upgrading the firmware](#) (on page 11-2) for details.

Send individual commands using the web interface

To send individual commands using the web interface:

1. Select **TSB Embedded**.
2. In the **Console** box, enter the command.
3. Select **Enter** to send the command to the instrument. The command is displayed in the Output box. If there is a response to the command, it is displayed after the command.
4. To clear the Output box, right-click and select **Clear**.

Review events in the LXI Event Log

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

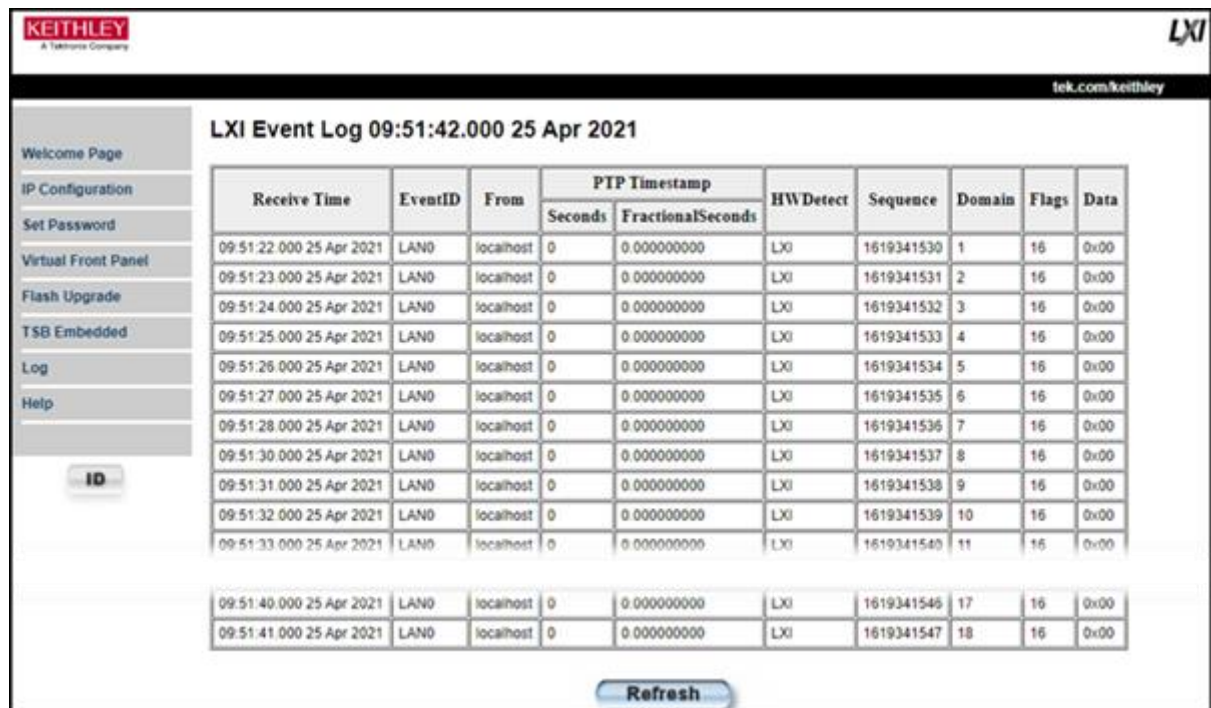
- The EventID column, which shows the identifier of the event that generated the event message.
- The PTP Timestamp column, which displays the seconds and fractional (nano) seconds when the event occurred.
- The Data column, which displays the text of the event message.

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

Logging LAN trigger events in the event log

You can use the event log to record all LXI triggers generated and received by the 2606B. You can view the event log using any command interface or the embedded web interface. The following figure shows the view of the LXI event log from the Log option in the embedded web interface.

Figure 18: LXI Event Log



The screenshot shows the LXI Event Log web interface. On the left is a navigation menu with options: Welcome Page, IP Configuration, Set Password, Virtual Front Panel, Flash Upgrade, TSB Embedded, Log (selected), and Help. Below the menu is an 'ID' button. The main content area is titled 'LXI Event Log 09:51:42.000 25 Apr 2021'. It contains a table with the following columns: Receive Time, EventID, From, PTP Timestamp (subdivided into Seconds and FractionalSeconds), HWDetect, Sequence, Domain, Flags, and Data. The table lists 18 events, all from LAN0 to localhost. At the bottom of the table is a 'Refresh' button.

Receive Time	EventID	From	PTP Timestamp		HWDetect	Sequence	Domain	Flags	Data
			Seconds	FractionalSeconds					
09:51:22.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341530	1	16	0x00
09:51:23.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341531	2	16	0x00
09:51:24.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341532	3	16	0x00
09:51:25.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341533	4	16	0x00
09:51:26.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341534	5	16	0x00
09:51:27.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341535	6	16	0x00
09:51:28.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341536	7	16	0x00
09:51:30.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341537	8	16	0x00
09:51:31.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341538	9	16	0x00
09:51:32.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341539	10	16	0x00
09:51:33.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341540	11	16	0x00
09:51:40.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341546	17	16	0x00
09:51:41.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341547	18	16	0x00

The timestamp, event identifier, IP address, and the domain name identify the incoming and outgoing LXI trigger packets. The following table provides detailed descriptions for the columns in the event log.

Event log descriptions

Column title	Description	Example
Receive Time	Displays the date and time that the LAN trigger occurred in UTC, 24-hour time	11:46:44.000 11 Mar 2020
Event ID	Identifies the <code>lan.trigger[N]</code> that generates an event	LAN0 = <code>lan.trigger[1]</code> LAN1 = <code>lan.trigger[2]</code> LAN2 = <code>lan.trigger[3]</code> LAN3 = <code>lan.trigger[4]</code> LAN4 = <code>lan.trigger[5]</code> LAN5 = <code>lan.trigger[6]</code> LAN6 = <code>lan.trigger[7]</code> LAN7 = <code>lan.trigger[8]</code>
From	Displays the IP address for the device that generates the LAN trigger	localhost 192.168.5.20

Event log descriptions

Column title	Description	Example
Timestamp	A timestamp that identifies the time the event occurred; the timestamp uses the following: ■ PTP timestamp ■ Seconds ■ Fractional seconds; the 2606B does not support the IEEE Std 1588 standard; the values in this field are always 0 (zero)	
HWDetect	Identifies a valid LXI trigger packet	LXI
Sequence	Each instrument maintains independent sequence counters: ■ One for each combination of UDP multicast network interface and UDP multicast destination port ■ One for each TCP connection	
Domain	Displays the LXI domain number; the default value is 0 (zero)	0
Flags	Contains data about the LXI trigger packet; values are: ■ 1 - Error ■ 2 - Retransmission ■ 4 - Hardware ■ 8 - Acknowledgments ■ 16 - Stateless bit	16
Data	The values for this are always 0 (zero)	

Accessing the event log from the command interface

You can access the event log from any remote command interface. The event log must be enabled before LXI trigger events can be viewed. To enable the event log, send:

```
eventlog.enable = 1
```

To view the event log from a remote interface, send:

```
print(eventlog.all())
```

This command outputs one or more strings similar to the following:

```
14:14:02.000 17 Jun 2019, LAN0, 10.80.64.191, LXI, 0, 1560780842, not available, 0, 0x10, 0x00
```

The string displays the same information as the web interface. Commas separate the fields. The fields output in the following order:

- Received time (UTC time)
- Event ID
- From (Sender)
- HWDetect / version
- Domain
- Sequence number
- Timestamp (PTP time)
- Epoch (from 1588)
- Flags
- Data

See the table in [Logging LAN trigger events in the event log](#) (on page 3-22) for detailed descriptions.

To generate log traffic, send the code:

```
local id = 1
lan.trigger[id].ipaddress = lan.status.ipaddress
lan.trigger[id].connect()
for domain = 1, 255 do
    print(domain)
    lan.lxidomain = domain
    lan.trigger[id].assert()
    delay(1)
end
```

Help

The help page in the web interface provides basic help for the web interface, a glossary of terms, and a link to licenses for the virtual front panel, firmware upgrade tool, and TSB Embedded.

Virtual Front Panel

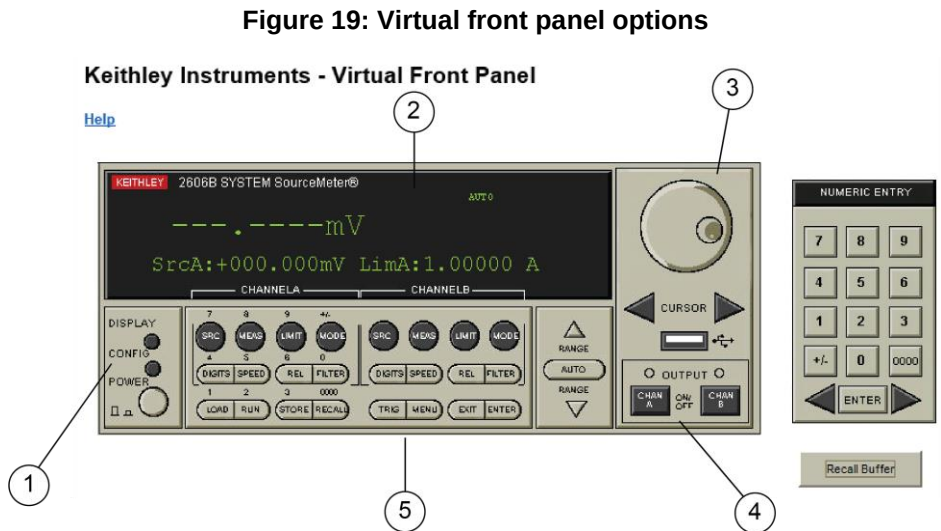
The Virtual Front Panel page allows you to control the instrument from a computer as if you were using an instrument with a display and keyboard. You can operate the instrument using a mouse to select options.

NOTE

The 2606B only 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 only view the virtual front panel.

Virtual front panel options

The virtual front panel of the 2606B is shown below. The descriptions of the controls and indicators follow the figure.



1. Power switch, display and configuration keys



Power switch. The in position turns the 2606B on (I); the out position turns it off (O). This is for display only on the virtual front panel.



Toggles between the source-measure display and the user message display.



Configures a function or operation.

2. SMU setup, performance control, special operation, and numbers

SMU (source-measure unit) setup



SRC	Selects the source function (voltage or current) and places the cursor in the source field for editing.
MEAS	Cycles through measure functions (voltage, current, resistance, or power).
LIMIT	Places the cursor in the compliance limit field for editing. Also selects the limit value to edit (voltage, current, or power).
MODE	Selects a meter mode (I-METER, V-METER, OHM-METER, or WATT-METER).

Performance control

DIGITS	Sets the display resolution (4½, 5½, or 6½ digits).
SPEED	Sets the measurement speed (FAST, MEdium, NORMAL, HI-ACCURACY, or OTHER). Speed and accuracy are set by controlling the measurement aperture. Also see Speed (on page 5-33).
REL	Controls relative measurements, which allows a baseline value to be subtracted from a reading.
FILTER	Enables or disables the digital filter. You can use this filter to reduce reading noise.

Special operation

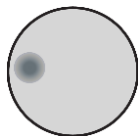
LOAD	Loads a test for execution (FACTORY, USER, or SCRIPTS).
RUN	Runs the last selected factory or user-defined test.
STORE	Accesses reading buffers and makes readings: ▪ TAKE_READINGS: Use to make readings and store them in a reading buffer. ▪ SAVE: Use to save a reading buffer to nonvolatile memory or to a user-installed flash drive (USB1) in CSV or XML format. Readings can include measurements, source values, and timestamp values.
RECALL	Recalls information (DATA or STATISTICS) stored in a reading buffer: ▪ DATA: Includes stored readings, and if configured, source values and timestamp values. ▪ STATISTICS: Includes MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, PK-PK.
TRIG	Triggers readings.
MENU	Accesses the main menu (on page 3-29). You can use the main menu to configure many functions and features.
EXIT	Cancels the selection and returns to the previous menu or display. Also used as a LOCAL key to take the instrument out of remote operation.
ENTER	Accepts the selection and moves to the next choice or exits the menu.

Numbers

Number keys	When enabled and in EDIT mode, the number keys (0 to 9, +/-, 0000) allow direct numeric entry. Press the navigation wheel to enter EDIT mode. For more information, see Setting a value (on page 3-32).



3. Navigation wheel



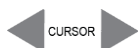
Turn the navigation wheel to:

- Move the cursor to the left and the right (the cursor indicates the selected value or item)
- While in edit mode, increase or decrease a selected source or compliance value

Push the navigation wheel to:

- Enable or disable edit mode for the selected source or compliance value
- Open menus and submenu items
- Select a menu option or a value

4. Cursor keys



Use the CURSOR keys to move the cursor left or right. When the cursor is on the source or compliance value digit, press the navigation wheel to enter edit mode, and turn the navigation wheel to edit the value. Press the navigation wheel again when you finish editing.

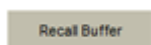
Use the CURSOR keys or the navigation wheel to move through menu items. To view a menu value, use the CURSOR keys for cursor control, and then press the navigation wheel to view the value or submenu item.

5. Numeric Entry



When enabled and in EDIT mode, the number keys (0-9, +/-, 0000) allow direct numeric entry. Press the navigation wheel to enter EDIT mode. For more information, see [Setting a value](#) (on page 3-32).

6. Recall Buffer



Opens the Reading Buffers application, which you can use to download and save data from the reading buffers.

7. Output control



Turns the source output on or off.

8. Range keys



Selects the next higher source or measure range.



Enables or disables source or measure autorange.



Selects the next lower source or measure range.

In addition to selecting range functions, the up and down range keys change the format for numbers that are not ranges, such as the limit value.

Display indicators (not shown)

The items listed below represent the possible display indicators and their meanings.

Indicator	Meaning
4W	Remote (4-wire) sense is selected
AUTO	Source or measure autorange is selected
EDIT	Instrument is in editing mode
ERR	Questionable reading or invalid calibration step
FILT	Digital filter is enabled
LSTN	Instrument is addressed to listen
REL	Relative mode is enabled
REM	Instrument is in remote mode
SRQ	Service request is asserted
TALK	Instrument is addressed to talk
* (asterisk)	Readings are being stored in the buffer

Menu overview

The following topics describe how to work with the virtual front-panel menus.

Menu navigation

To navigate through the menus and submenus, the 2606B must not be in edit mode (the EDIT indicator is not illuminated).

Selecting menu items

To navigate the Main and Configuration menus, use the virtual front-panel keys as follows:

- Press either **CURSOR** arrow key to highlight an option.
- Rotate the navigation wheel (clockwise or counterclockwise) to highlight an option.
- Press the **ENTER** key (or the navigation wheel) to select an option.
- Use the **EXIT (LOCAL)** key to cancel changes or to return to the previous menu or display.

NOTE

For quick menu navigation, turn the navigation wheel to highlight an option and then press the navigation wheel to select the highlighted option.

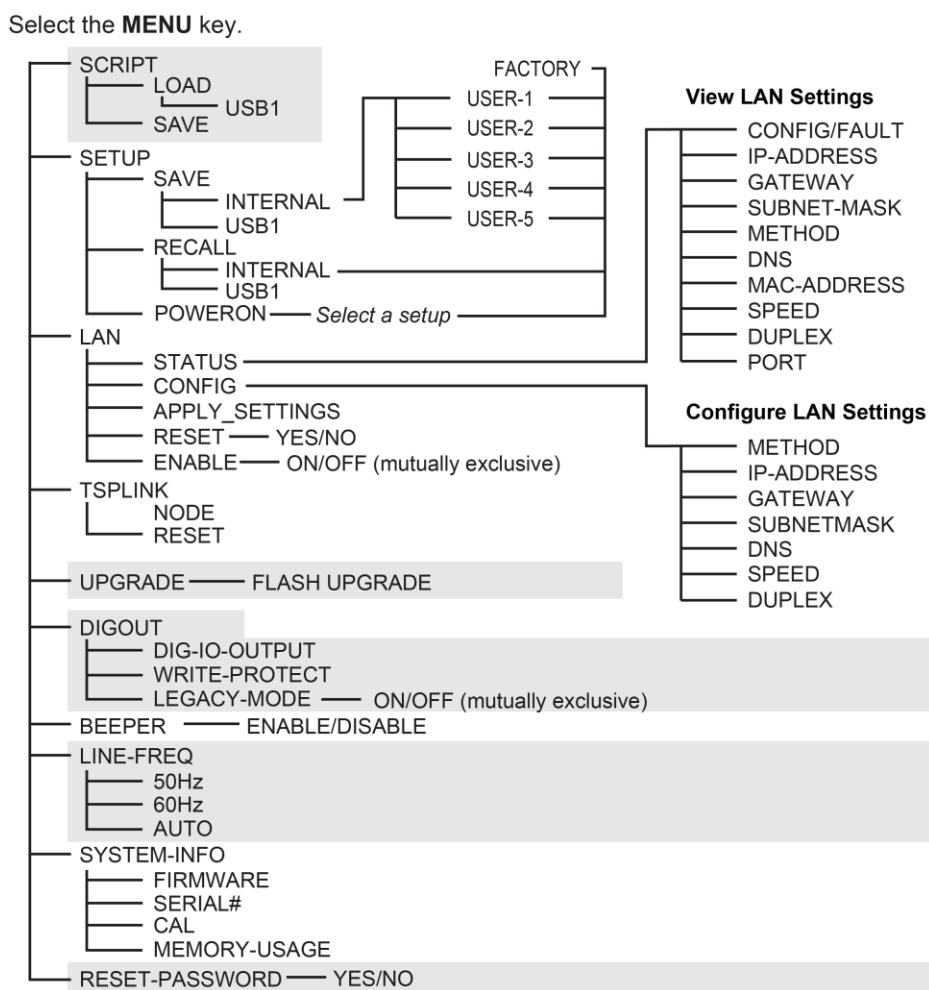
Menu trees

You can configure instrument operation through the menus that are accessed from the virtual front panel.

Main menu

The main menu structure is summarized in the following figure and table. For other menu items, see [Configuration menus](#) (on page 3-30).

Figure 20: 2606B main menu tree



The following table contains descriptions of the main menu options and cross-references to related information. To access a menu option, press the **MENU** key, turn the navigation wheel to move the cursor to select an item, and press the navigation wheel.

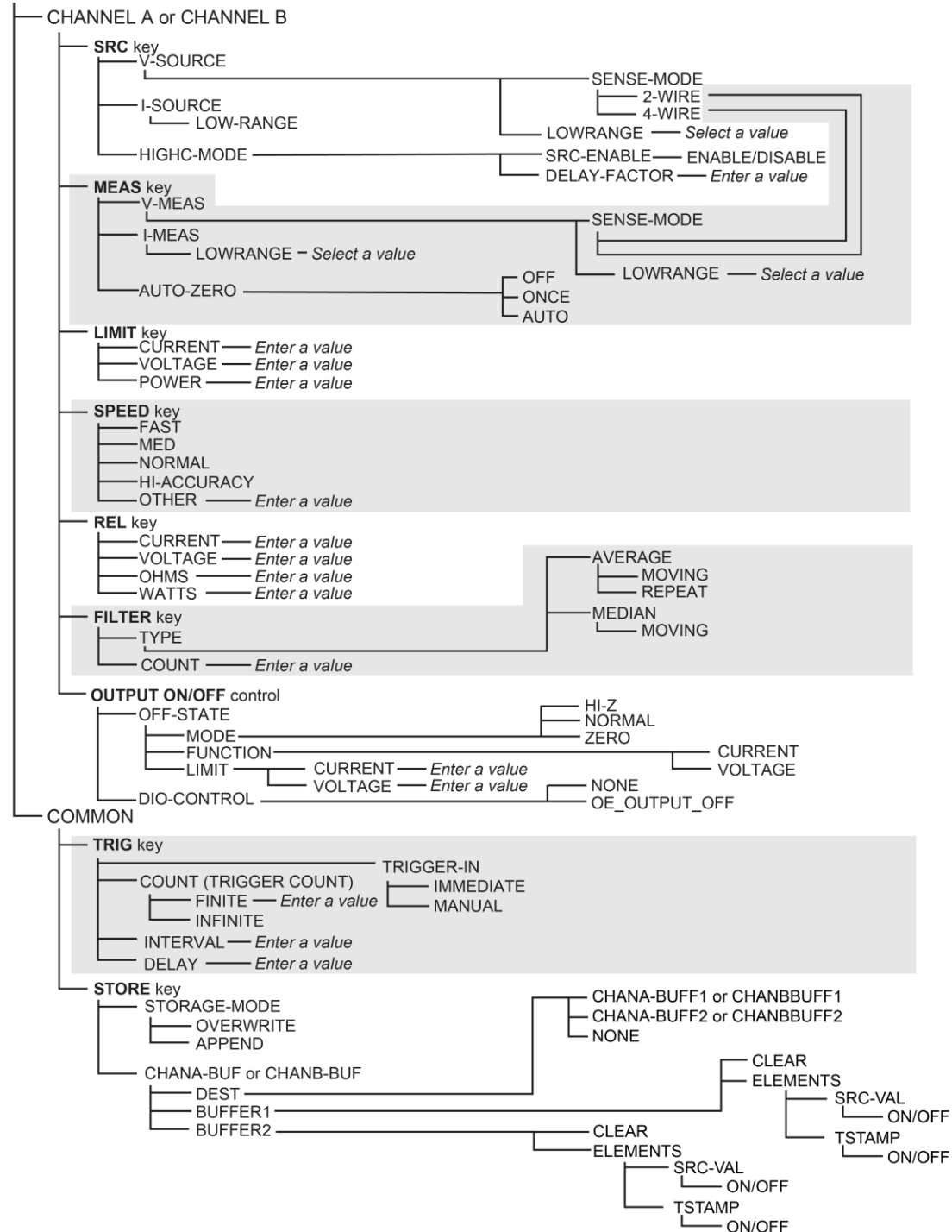
Menu selection	Description	For more information, see:
SCRIPT - LOAD - SAVE	Saves and recalls user scripts Loads scripts into nonvolatile memory Saves scripts	Manage scripts (on page 14-3)
SETUP - SAVE - RECALL - POWERON	Saves and recalls user and factory setup options Saves user setup options Recalls user setup options Sets the configuration used during startup	Saved setups (on page 4-13)
LAN - STATUS - CONFIG - APPLY_SETTINGS - RESET - ENABLE	Configures the local area network (LAN) Displays LAN connection status Configures the LAN IP address and gateway Applies changes made using the CONFIG menu Restores the default settings Enables and disables the LAN interface	LAN concepts and settings (on page 9-1)
TSPLINK - NODE - RESET	Configure the instrument in a TSP-Link® network Selects the instrument node identifier Resets the TSP-Link network	TSP-Link system expansion interface (on page 14-52)
UPGRADE	Upgrades the firmware from a USB flash drive	Upgrading the firmware (on page 11-2)
DIGOUT - DIG-IO-OUTPUT - WRITE-PROTECT	Controls digital outputs Selects the digital I/O values Write-protects specific digital I/O lines	Digital I/O (on page 7-37)
BEEPER - ENABLE - DISABLE	Controls the key beeps Enables the key beeps Disables the key beeps	Beeper (on page 3-33)
LINE-FREQ - 50Hz - 60Hz - AUTO	Configures the line frequency Set the line frequency to 50 Hz Set the line frequency to 60 Hz Enables automatic line frequency detection during start up	Line frequency configuration (on page 2-6)
SYSTEM-INFO - FIRMWARE - SERIAL# - CAL - MEMORY-USAGE	Displays the system information Displays the version of firmware installed Displays the serial number of the unit Displays the last calibration date Displays memory usage in percentage	System information (on page 2-6)
RESET-PASSWORD	Resets the system password	Password management (on page 14-36)

Configuration menus

The configuration menu structure is summarized in the following figure and table. For directions on navigating the menu, see [Menu navigation](#) (on page 3-28). For other menu items, see [Main menu](#) (on page 3-29).

Figure 21: 2606B CONFIG menu tree

Press the **CONFIG** key, then select:



NOTE

Press the **EXIT** key to return to a previous menu.

The following table contains descriptions of the configuration menus with cross references to related information.

To select a menu:

1. Select the **CONFIG** key.
2. Select **CHANNEL-A**, **CHANNEL-B**, or **COMMON**. **COMMON** accesses settings for **TRIG** and **STORE**.
3. Select the key associated with the menu.

To access, select:	Options	For more information, see:
SRC	V-source sense, low range; I-source low range; and high capacitance mode	Range (on page 5-29), Basic source-measure procedure
MEAS	V and I-measure range, V-measure sense, low range; autozero	Range (on page 5-29), Basic source-measure procedure
LIMIT	V-source and I-source compliance limits	Limits (on page 4-1)
SPEED	Measurement speed (NPLC)	Speed (on page 5-33)
REL	Relative values	Relative offset (on page 5-47)
FILTER	Control digital filter	Filters (on page 5-49)
OUTPUT ON/OFF	Off-state, control digital I/O	Output-off states (on page 5-25)
TRIG	Trigger in, count, interval, and delay	Triggering (on page 7-1)
STORE	Buffer count and destination	Source-measure concepts (on page 8-1)

Setting a value

Through the virtual front panel, you can adjust a value using the navigation wheel and the keypad.

To adjust a value:

1. Use the **CURSOR** arrow keys or the navigation wheel to move the cursor to the digit that needs to be changed.
2. Select the navigation wheel or the **ENTER** key to enter edit mode. The **EDIT** indicator is illuminated.
3. Rotate the navigation wheel or use the number pad to set the value. If you use the number pad, the cursor moves to the next digit on the right.
4. Select the **ENTER** key to select the value or press the **EXIT (LOCAL)** key to cancel the change.
5. To return to the main menu, press the **EXIT (LOCAL)** key.

NOTE

To set an option to zero (if it is an option for the entry), select the **0000** numeric entry key. To toggle the polarity of a value, press the **+/-** numeric entry key.

Setting source and compliance values

When the 2606B is in the edit mode (EDIT indicator is on), the editing controls are used to set source and compliance values. Note that when you edit the source value, source autoranging is turned off and remains off until you turn it on again.

NOTE

To cancel source editing, press the **EXIT (LOCAL)** key.

To edit the source value:

1. Press the **SRC** key. The cursor flashes in the source value field.
2. Use the **CURSOR** keys (or turn the navigation wheel) to move the cursor to the digit that needs to be changed.
3. Press the navigation wheel or the **ENTER** key to edit the source value. The EDIT indicator is illuminated.
4. Change the source value (see [Setting a value](#) (on page 3-32)).

NOTE

The **+/-** key toggles the polarity. The **0000** key sets the value to 0.

5. When finished, press the **ENTER** key (the EDIT indicator is not illuminated).

To edit compliance limit values:

1. Press the **LIMIT** key.
2. Select the type of compliance (CURRENT, VOLTAGE, or POWER).
3. Press the navigation wheel or the **ENTER** key to enter edit mode. The EDIT indicator is illuminated.
4. Change the compliance value (see [Setting a value](#) (on page 3-32)).
5. When finished, press the **ENTER** key (the EDIT indicator is not illuminated).

NOTE

The up and down range keys change the format of the limit value.

Beeper

The 2606B includes a beeper. When it is enabled, a beep indicates one of the following actions occurred:

- **A virtual front-panel key was pressed:** A short beep, similar to a key click, is issued.
- **The navigation wheel was turned or pressed:** A short beep is issued.
- **The output source was changed:** A longer beep is issued when you select the OUTPUT ON/OFF control (turn the output on or off).

To turn the beeper on or off from the virtual front panel:

1. Press the **MENU** key, and then select **BEEPER**.
2. Select one of the following:
 - **ENABLE**
 - **DISABLE**

To turn the beeper on or off from the TSP command interface:

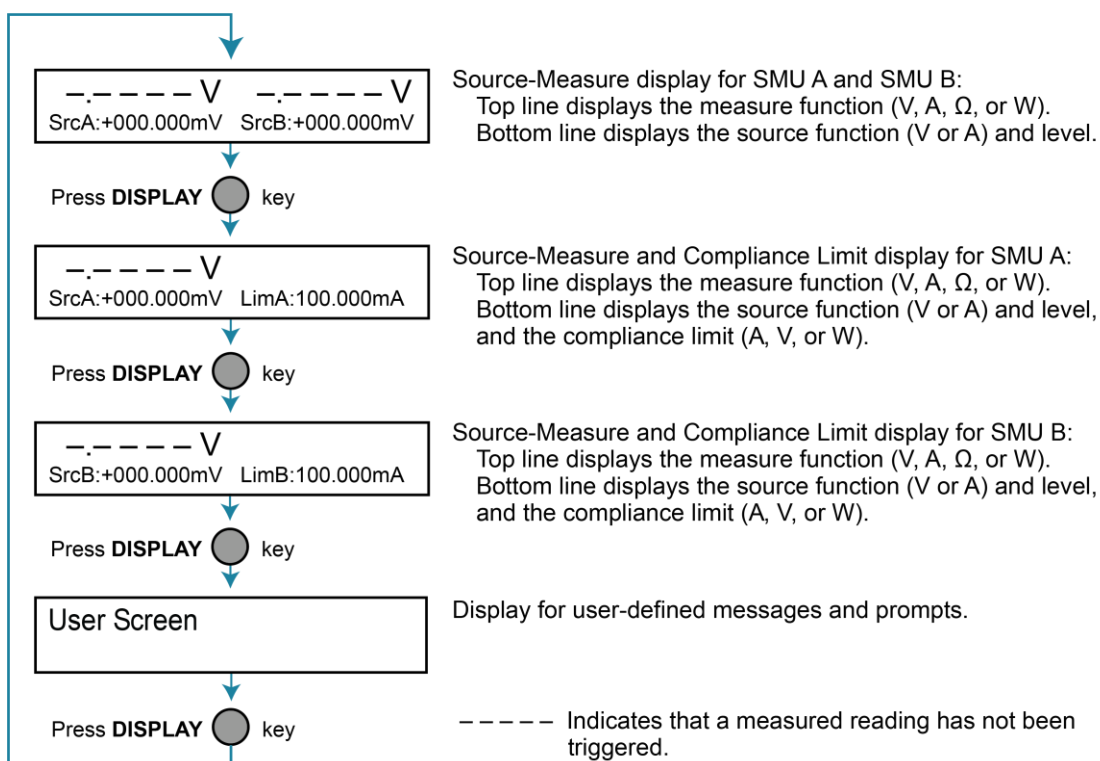
Set the `beeper.enable` attribute. For example, to enable the beeper, send:

```
beeper.enable = 1
```

Display mode

Use the **DISPLAY** key to scroll through the various display modes shown in the figure below. Refer to [Display operations](#) (on page 3-35) for more information about the display.

Figure 22: Display modes



Display operations

This section describes methods for using the display and determining what is displayed.

Display functions and attributes

The display functions and attributes for the virtual front panel are described in this section. The following table lists each display function and attribute (in alphabetical order) and cross references it to the section topic where the function or attribute is explained.

The [TSP command reference](#) (on page 15-1) provides additional information about the display functions and attributes.

Cross-referencing functions and attributes to section topics

Function or attribute*	Section topic
<code>display.clear()</code>	Clearing the display (on page 3-37)
<code>display.getannunciators()</code>	Indicators (on page 3-43)
<code>display.getcursor()</code>	Cursor position (on page 3-38)
<code>display.getlastkey()</code>	Capturing key-press codes (on page 3-46)
<code>display.gettext()</code>	Displaying text messages (on page 3-39)
<code>display.inputvalue()</code>	Parameter value prompting (on page 3-42)
<code>display.loadmenu.add()</code> <code>display.loadmenu.catalog()</code> <code>display.loadmenu.delete()</code>	Load test menu (on page 3-44)
<code>display.locallockout</code>	LOCAL lockout (on page 3-44)
<code>display.menu()</code>	Menu (on page 3-41)
<code>display.numpad</code>	Setting a value (on page 3-32)
<code>display.prompt()</code>	Parameter value prompting (on page 3-42)
<code>display.screen</code>	Display screen (on page 3-36)
<code>display.sendkey()</code>	Sending key codes (on page 3-46)
<code>display.setcursor()</code>	Cursor position (on page 3-38)
<code>display.settext()</code>	Displaying text messages (on page 3-39)
<code>display.smuX.digits</code>	Display resolution (on page 3-36)
<code>display.smuX.limit.func</code>	display.smuX.limit.func (on page 15-82)
<code>display.smuX.measure.func</code>	Measurement functions (on page 3-36)
<code>display.trigger.clear()</code> <code>display.trigger.wait()</code>	Display trigger wait and clear (on page 3-36)
<code>display.waitkey()</code>	Capturing key-press codes (on page 3-46)

* smuX can be smua for channel A or smub for channel B

Display features

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

Display screen

The virtual front panel displays source-measure values and readings or user-defined messages. The display screen options include:

- **Source-measure, compliance screens:** Display SMU source-measure readings and compliance values.
- **User screen:** Displays user-defined messages and prompts.

Configure the type of source-measure and compliance displayed by setting the `display.screen` attribute. The following programming example illustrates how to display source-measure and compliance values, and measure readings for SMU A:

```
display.screen = display.SMUA
```

Measurement functions

With a source-measure screen selected, the measured reading can be displayed as volts, amperes, ohms, or watts. Configure the type of measured reading displayed by setting the `display.smuX.measure.func` attribute. The following programming example illustrates how to set SMU A to display ohms measurements:

```
display.smua.measure.func = display.MEASURE_OHMS
```

Display resolution

Display resolution for measured readings can be set to 4½, 5½, or 6½. Configure the type of resolution displayed by setting the `display.smuX.digits` attribute. The following programming example illustrates how to set SMU A for 5½ digit resolution for measured readings:

```
display.smua.digits = display.DIGITS_5_5
```

Display trigger wait and clear

To set the instrument to wait for the virtual front-panel TRIG key to be pressed, send the `display.trigger.wait()` function. To clear the trigger event detector, send the `display.trigger.clear()` function.

Display messages

You can define text messages that can be displayed on the front panel of the instrument. Most of the display functions and attributes that are associated with display messaging automatically select the user screen. The attribute for the display screen is explained in [Display screen](#) (on page 3-36).

For example, while a test is running, the following message can be displayed on the 2606B virtual front panel:

```
Test in Process
Do Not Disturb
```

The top line of the display can accommodate up to 20 characters (including spaces). The bottom line can display up to 32 characters (including spaces) at a time.

NOTE

The display functions `display.clear()`, `display.setcursor()`, and `display.settext()` are overlapped, nonblocking commands. The script does not wait for one of these commands to complete.

These nonblocking functions do not immediately update the display. For performance considerations, they write to a background file and update the display as soon as processing time becomes available.

The reset functions `reset()` and `smua.reset()` do not change the defined display message or its configuration. The reset functions set the display mode to the previous source-measure display mode. To show the user-defined message again, press DISPLAY until the User screen is displayed.

Clearing the display

When sending a command to display a message, a previously defined user message is not cleared. The new message starts at the end of the old message on that line. It is good practice to routinely clear the display before defining a new message.

After displaying an input prompt, the message is displayed even after the operator performs the prescribed action. The `clear()` function must be sent to clear the display. To clear both lines of the display, but not affect any of the indicators, send the following function:

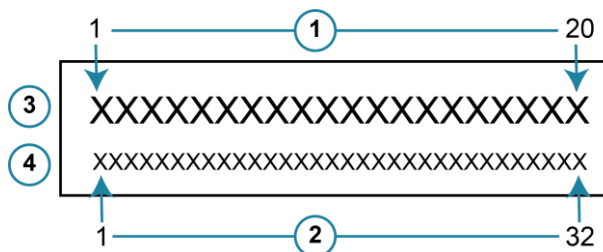
```
display.clear()
```

Cursor position

When displaying a message, the cursor position determines where the message starts. On power-up, the cursor is positioned at row 1, column 1 (see the following figure). At this cursor position, a user-defined message is displayed on the top row (row 1).

Top line text does not wrap to the bottom line of the display automatically. Any text that does not fit on the current line is truncated. If the text is truncated, the cursor is left at the end of the line.

Figure 23: Row and column format for display messaging



1 Columns for Row 1
2 Columns for Row 2
3 Row 1
4 Row 2
X Display character

The function to set cursor position has the following options:

```
display.setcursor(row, column)
display.setcursor(row, column, style)
```

Where:

```
row          1 or 2
column       1 to 20 (row 1)
              1 to 32 (row 2)
style        0 (invisible)
              1 (blink)
```

When set to 0, the cursor is not visible. When set to 1, a display character blinks to indicate the cursor position.

The `display.getcursor()` function returns the present cursor position. You can use it in these ways:

```
row, column, style = display.getcursor()
row, column = display.getcursor()
row = display.getcursor()
```

The following programming example illustrates how to position the cursor on row 2, column 1, and then read the cursor position:

```
display.setcursor(2, 1)
row, column = display.getcursor()
print(row, column)
```

Output:

```
2.000000e+00 1.000000e+00
```

Displaying text messages

To define and display a message, use the `display.settext(text)` function, where `text` is the text string to be displayed. The message starts at the present cursor position. The following programming example illustrates how to display `Test in Process` on the top line, and `Do Not Disturb` on the bottom line:

```
display.clear()
display.setcursor(1, 1, 0)
display.settext("Test in Process")
display.setcursor(2, 6, 0)
display.settext("Do Not Disturb")
```

Character codes

The following special codes can be embedded in the `text` string to configure and customize the message:

- `$N`: Starts text on the next line (newline). If the cursor is already on line 2, text is ignored after the '`$N`' is received.
- `$R`: Sets text to Normal.
- `$B`: Sets text to Blink.
- `$D`: Sets text to Dim intensity.
- `$F`: Set text to background blink.
- `$$`: Escape sequence to display a single "\$".

In addition to displaying alphanumeric characters, you can display other special characters.

The following programming example illustrates how to display the Greek symbol omega (Ω):

```
display.clear()
c = string.char(18)
display.settext(c)
```

The following programming example illustrates how to use the `$N` and `$B` character codes to display the message `Test in Process` on the top line and the blinking message `Do Not Disturb` on the bottom line:

```
display.clear()
display.settext("Test in Process $N$BDo Not Disturb")
```

The following programming example illustrates how to use the `$$` character code to display the message `You owe me $8` on the top line:

```
display.clear()
display.setcursor(1, 1)
display.settext("You owe me $$8")
```

If the extra `$` character is not included, the `$8` is interpreted as an undefined character code and is ignored. The message `You owe me` is displayed.

NOTE

Be careful when embedding character codes in the text string. It is easy to forget that the character following the `$` is part of the code. For example, if you want to display `Hello` on the top line and `Nate` on the bottom line, send the following command:

```
display.settext("Hello$Nate")
```

The above command displays `Hello` on the top line and `ate` on the bottom line. The correct syntax for the command is as follows:

```
display.settext("Hello$NNate")
```

Returning a text message

The `display.gettext()` function returns the displayed message (*text*) and has the following options:

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, columnStart)
text = display.gettext(embellished, row, columnStart, columnEnd)
```

Where:

<i>embellished</i>	Returns text as a simple character string (<i>false</i>) or includes character codes (<i>true</i>)
<i>row</i>	The row to read text from (1 or 2); if not included, text from both rows is read
<i>columnStart</i>	Starting column for reading text
<i>columnEnd</i>	Ending column for reading text

Sending the command without the *row* parameter returns both lines of the display. The *\$N* character code is included to show where the top line ends and the bottom line begins. The *\$N* character code is returned even if *embellished* is set to *false*.

With *embellished* set to *true*, all other character codes that were used in the creation of each message line are returned with the message. With *embellished* set to *false*, only the message is returned.

Sending the command without the *columnStart* parameter defaults to column 1. Sending the command without the *columnEnd* argument defaults to the last column (column 20 for row 1, column 32 for row 2).

Input prompting

You can use display messaging with virtual front panel controls to make a user script interactive. In an interactive script, input prompts are displayed so that the operator can perform a prescribed action using the virtual front panel controls. While displaying an input prompt, the test pauses and waits for the operator to perform the prescribed action.

Menu

You can present a user-defined menu on the display. The menu consists of the menu name on the top line and a selectable list of menu items on the bottom line. To define a menu, use the `display.menu(menu, items)` function, where:

menu The name of the menu; use a string of up to 20 characters (including spaces)
items A string is made up of one or more menu items; each item must be separated by white space

When the `display.menu()` function is sent, script execution waits for the operator to select one of the menu items. Rotate the navigation wheel to place the blinking cursor on a menu item. Items that do not fit in the display area are displayed by rotating the navigation wheel to the right. With the cursor on the menu item, press the navigation wheel (or the **ENTER** key) to select it.

Pressing the EXIT (LOCAL) key does not abort the script while the menu is displayed, but it returns *nil*. The script can be aborted by calling the `exit()` function when *nil* is returned.

The following programming example illustrates how to present the operator with the choice of two menu items: *Test1* or *Test2*. If *Test1* is selected, the message *Running Test1* is displayed. If *Test2* is selected, the message *Running Test2* is displayed.

```
display.clear()
menu = display.menu("Sample Menu", "Test1 Test2")
if menu == "Test1" then
    display.settext("Running Test1")
else
    display.settext("Running Test2")
end
```


Parameter value prompting

You can use the `display.inputvalue()` and `display.prompt()` functions to create an editable input field on the user screen at the present cursor position.

The `display.inputvalue()` function uses the user screen at the present cursor position. Once the command is finished, it returns the user screen to its previous state. The `display.prompt()` function creates a new edit screen and does not use the user screen.

Each of these functions can be used in the following ways:

```
display.inputvalue(format)
display.inputvalue(format, default)
display.inputvalue(format, default, min)
display.inputvalue(format, default, min, max)
display.prompt(format, units, help)
display.prompt(format, units, help, default)
display.prompt(format, units, help, default, min)
display.prompt(format, units, help, default, min, max)
```

Where:

<i>format</i>	String that creates an editable input field on the user screen at the present cursor position (examples: +0.00 00, +00, 0.00000E+0) Value field: + = Include for positive/negative value entry; omitting the + prevents negative value entry 0 = Defines the digit positions for the value (up to six zeros (0)) Exponent field (optional): E = include for exponent entry + = Include for positive/negative exponent entry; omitting the + prevents negative value entry 0 = Defines the digit positions for the exponent
<i>default</i>	Option to set a default value for the parameter, which is displayed when the command is sent
<i>min</i>	Option to specify minimum limits for the input field ■ When NOT using the “+” sign for the value field, the minimum limit cannot be set to less than zero ■ When using the “+” sign, the minimum limit can be set to less than zero (for example, -2)
<i>max</i>	Option to specify maximum limits for the input field
<i>units</i>	Text string to identify the units for the value (8 characters maximum), for example: Units text is “V” for volts and “A” for amperes
<i>help</i>	Informational text string to display on the bottom line (32 characters maximum)

Both the `display.inputvalue()` and `display.prompt()` functions display the editable input field, but the `display.inputvalue()` function does not include the text strings for *units* and *help*.

After one of the above functions is executed, command execution pauses and waits for the operator to input the source level. The program continues after the operator enters the value by pressing the navigation wheel or the ENTER key.

The following programming example illustrates how to prompt the operator to enter a source voltage value for SMU A:

```
display.clear()
value = display.prompt("0.00", "V", "Enter source voltage")
display.screen = display.SMUA
smua.source.levelv = value
```

The script pauses after displaying the prompt message and waits for the operator to enter the voltage level. The display then toggles to the source-measure display for SMU A and sets the source level to `value`.

NOTE

If the operator presses EXIT(LOCAL) instead of entering a source value, `value` is set to `nil`.

The second line of the above code can be replaced using the other input field function:

```
value = display.inputvalue("0.00")
```

The only difference is that the display prompt does not include the “V” units designator and the Enter source value message.

Indicators

To determine which virtual front-panel display indicators are turned on, use the `display.getannunciators()` function. For example, send the following commands.

```
annun = display.getannunciators()
print(annun)
```

The 16-bit binary equivalent of the returned value is a bitmap. Each bit corresponds to an indicator. If the bit is set to 1, the indicator is turned on. If the bit is set to 0, the indicator is turned off.

The following table identifies the bit position for each indicator. The table also includes the weighted value of each bit. The returned value is the sum of all the weighted values for the bits that are set.

Bit identification for indicators

Bit	B16	B15	B14	B13	B12	B11	B10	B9
Annunciator	REL	REAR	SRQ	LSTN	TALK	REM	ERR	EDIT
Weighted value*	32768	16384	8192	4096	2048	1024	512	256
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Bit	B8	B7	B6	B5	B4	B3	B2	B1
Annunciator	SMPL	STAR	TRIG	ARM	AUTO	4W	MATH	FILT
Weighted value*	128	64	32	16	8	4	2	1
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

* The weighted values are for bits that are set to 1. Bits set to 0 have no value.

Not all the indicators shown in above table may be used by the 2606B.

For example, assume the returned bitmap value is 34061. The binary equivalent of this value is as follows:

```
1000010100001101
```

For the above binary number, the following bits are set to 1: 16, 11, 9, 4, 3, and 1. Using the table, the following indicators are on: REL, REM, EDIT, AUTO, 4W, and FILT.

Local lockout

You can use the virtual front-panel EXIT (LOCAL) key to cancel remote operation and return control to the virtual front panel. However, this key can be locked out to prevent a test from being interrupted. When locked, this key becomes a NO-OP (no operation). Configure the following attribute to lock or unlock the EXIT (LOCAL) key:

```
display.locallockout = lockout
```

Where *lockout* is set to one of the following values:

0 or `display.UNLOCK`

1 or `display.LOCK`

For example, to lock out the EXIT (LOCAL) key:

```
display.locallockout = display.LOCK
```

Load test menu

The LOAD TEST menu lists tests (USER, FACTORY, and SCRIPTS) that can be run from the virtual front panel.

Factory tests are preloaded and saved in nonvolatile memory at the factory. They are available in the FACTORY TESTS submenu.

If you load named scripts into the runtime environment, they can be selected from the front-panel SCRIPTS menu.

User tests

User tests can be added to or deleted from the USER TESTS submenu.

Adding USER TESTS menu entries

You can use the following function in either of two ways to add an entry into the USER TESTS menu:

```
display.loadmenu.add(displayname, code)  
display.loadmenu.add(displayname, code, memory)
```

Where:

<i>displayname</i>	The name string that is added to the USER TESTS menu.
<i>code</i>	The code that is run from the USER TESTS menu when the RUN button is pressed. It can include any valid Lua code.
<i>memory</i>	A value that specifies if the <i>code</i> and <i>displayname</i> parameters are saved in nonvolatile memory. Set to one of the following values: 0 or <code>display.DONT_SAVE</code> 1 or <code>display.SAVE</code> (this is the default setting)

Scripts, functions, and variables that are used in the *code* are not saved when `display.SAVE` is used. Functions and variables need to be saved with the script. If the script is not saved in nonvolatile memory, it is lost when the 2606B is turned off. See Example 1 below.

Example 1:

Assume a script with a function named `DUT1` has been loaded into the 2606B, and the script has not been saved in nonvolatile memory.

Now assume you want to add a test named `Test` to the USER TESTS menu. You want the test to run the function named `DUT1` and sound the beeper. The following programming example illustrates how to add `Test` to the menu, define the *code*, and then save *displayname* and *code* in nonvolatile memory:

```
display.loadmenu.add("Test", "DUT1() beeper.beep(2, 500)", display.SAVE)
```

When `Test` is run from the virtual front-panel USER TESTS menu, the function named `DUT1` executes and the beeper beeps for two seconds.

Now assume you turn the 2606B power off and then on again. Because the script was not saved in nonvolatile memory, the function named `DUT1` is lost. When `Test` is again run from the virtual front panel, the beeper beeps, but `DUT1` does not execute because it is no longer in the runtime environment.

Example 2:

The following command adds an entry called `Part1` to the virtual front-panel USER TESTS submenu for the code `testpart([[Part1]], 5.0)` and saves it in nonvolatile memory:

```
display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)", display.SAVE)
```

Deleting USER TESTS menu entries

You can use the following function to delete an entry from the virtual front-panel USER TESTS menu:

```
display.loadmenu.delete(displayname)
```

Where:

displayname Name to delete from the menu.

The following programming example removes the entry named `Part1` from the virtual front-panel USER TESTS menu:

```
display.loadmenu.delete("Part1")
```

Running a test from the virtual front panel

To run a user, factory, or script test from the virtual front panel:

1. Press the **LOAD** key to display the LOAD TEST menu.
2. Select the **USER**, **FACTORY**, or **SCRIPTS** menu item.
3. Position the blinking cursor on the test to be run and press **ENTER** or the navigation wheel.
4. Press the **RUN** key to run the test.

Key-press codes

You can use key codes to remotely simulate pressing a virtual front-panel key or the navigation wheel. There are also key codes to simulate rotating the navigation wheel to the left or right (one click at a time).

Sending key codes

Use the `display.sendkey()` function to remotely simulate pressing a front-panel key or the navigation wheel. The following programming examples illustrate how to simulate pressing the MENU key in two different ways:

```
display.sendkey(display.KEY_MENU)
display.sendkey(68)
```

Capturing key-press codes

A history of the key code for the last pressed virtual front-panel key is maintained by the 2606B. When the instrument is turned on (or when transitioning from local to remote operation), the key code is set to 0 (`display.KEY_NONE`).

When a virtual front-panel key is pressed, the key code value for that key can be captured and returned. There are two functions associated with the capture of key-press codes:

`display.getlastkey()` and `display.waitkey()`.

display.getlastkey()

The `display.getlastkey()` function immediately returns the key code for the last pressed key. The following programming example illustrates how to display the last key pressed:

```
key = display.getlastkey()
print(key)
```

The above code returns the key code value (see the following table). A value of 0 (`display.KEY_NONE`) indicates that the key code history had been cleared.

Key codes

Value	Key list	Value	Key list
0	<code>display.KEY_NONE</code>	83	<code>display.KEY_MEASB</code>
65	<code>display.KEY_RANGEUP</code>	84	<code>display.KEY_DIGITSB</code>
67	<code>display.KEY_RELB</code>	85	<code>display.KEY_RECALL</code>
68	<code>display.KEY_MENU</code>	86	<code>display.KEY_MEASA</code>
69	<code>display.KEY_MODEA</code>	87	<code>display.KEY_DIGITSA</code>
70	<code>display.KEY_RELA</code>	90	<code>display.KEY_LIMITB</code>
71	<code>display.KEY_RUN</code>	91	<code>display.KEY_SPEEDB</code>
72	<code>display.KEY_DISPLAY</code>	92	<code>display.KEY_TRIG</code>
73	<code>display.KEY_AUTO</code>	93	<code>display.KEY_LIMITA</code>
75	<code>display.KEY_EXIT</code>	94	<code>display.KEY_SPEEDA</code>
77	<code>display.KEY_FILTERA</code>	95	<code>display.KEY_LOAD</code>
78	<code>display.KEY_STORE</code>	97	<code>display.WHEEL_ENTER</code>
79	<code>display.KEY_SRCA</code>	103	<code>display.KEY_RIGHT</code>
80	<code>display.KEY_CONFIG</code>	104	<code>display.KEY_LEFT</code>
81	<code>display.KEY_RANGEDOWN</code>	107	<code>display.WHEEL_LEFT</code>
82	<code>display.KEY_ENTER</code>	114	<code>display.WHEEL_RIGHT</code>

NOTE

The OUTPUT ON/OFF control for a source-measure unit (SMU) cannot be tracked by this function.

display.waitkey()

The `display.waitkey()` function captures the key code value for the next key press:

```
key = display.waitkey()
```

After sending the `display.waitkey()` function, the script pauses and waits for the operator to press a virtual front-panel key. For example, if the MENU key is pressed, the function returns the value 68, which is the key code for that key. The key code values are the same as listed in [display.getlastkey\(\)](#) (on page 15-68).

The following programming example illustrates how to prompt the user to press the EXIT (LOCAL) key to abort the script, or any other key to continue it:

```
display.clear()
display.setcursor(1, 1)
display.settext("Press EXIT to Abort")
display.setcursor(2, 1)
display.settext("or any key to continue")
key = display.waitkey()
display.clear()
display.setcursor(1, 1)
if key == 75 then
    display.settext("Test Aborted")
    exit()
else
    display.settext("Test Continuing")
end
```

The above code captures the key that is pressed by the operator. The key code value for the EXIT (LOCAL) key is 75. If the EXIT (LOCAL) key is pressed, the script aborts. If any other key is pressed, the script continues.

In this section:

Limits.....	4-1
Fundamental circuit configurations.....	4-2
DUT test connections.....	4-3
Saved setups	4-13
USB storage overview.....	4-16
Displayed error and status messages	4-18

Limits

When sourcing voltage, the 2606B can be set to limit current or power. Conversely, when sourcing current, the 2606B can be set to limit voltage or power. In steady-state conditions, the 2606B output will not exceed the limit. The maximum limit is the same as the maximum values listed in the specifications.

The limit circuit limits in either polarity regardless of the polarity of the source or limit value. The accuracy of the limit opposite in polarity from the source is diminished. The maximum limits are based on the source range. For more information, see [Limit principles](#) (on page 8-1).

The limit operation of the instrument changes dependent on the source mode (current or voltage), load, and the configured limits (current, voltage, and power). As the names imply, the current limit restricts the current for sourced voltage and the voltage limit restricts the voltage for a sourced current. The power limit, however, restricts power by lowering the present limit in effect (voltage or current) as needed to restrict the SMU from exceeding the specified power limit. For additional details on using limits, including load considerations when specifying both a current (or a voltage) limit and a power limit, see [Operating boundaries](#) (on page 8-3).

The following topics describe how to set the limits.

Virtual front-panel limit

To set the limit from the virtual front panel:

1. Select the **LIMIT** key for the channel.
2. Select **CURRENT**, **VOLTAGE**, or **POWER** as needed.
3. Select the **ENTER** key.
4. Use the navigation wheel to set the new value.
5. Press the **ENTER** key.
6. Press the **EXIT (LOCAL)** key to return to the main display.

Remote limit

The table below summarizes basic commands to program a limit. For a more complete description of these commands, refer to the [TSP command reference](#) (on page 15-1).

Limit commands

Command	Description
<code>smuX.source.limiti = limit</code>	Set current limit.
<code>smuX.source.limitv = limit</code>	Set voltage limit.
<code>smuX.source.limitp = limit</code>	Set power limit.
<code>compliance = smuX.source.compliance</code>	Test if in limit (<code>true</code> = in limit; <code>false</code> = not in limit).

To set the limit, send the command with the limit value as the parameter. The following programming example illustrates how to set the current, voltage, and power limit to 50 mA, 4 V, and 1 W respectively:

```
smua.source.limiti = 50e-3
smua.source.limitv = 4
smua.source.limitp = 1
```

The following programming example illustrates how to print the limit state:

```
print(smua.source.compliance)
```

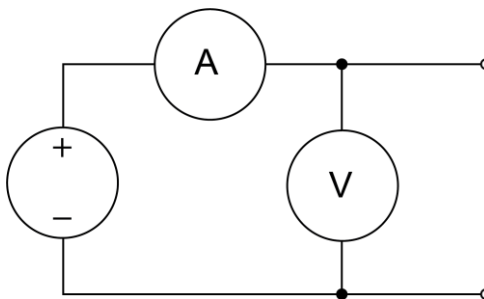
A returned value of `true` indicates one of three things:

- If the instrument is configured as a current source, the voltage limit has been reached
- If the instrument is configured as a voltage source, the current limit has been reached

Fundamental circuit configurations

The fundamental source-measure configurations for the 2606B are shown in the following figures. When sourcing voltage, you can measure current or voltage, as shown in the following figure.

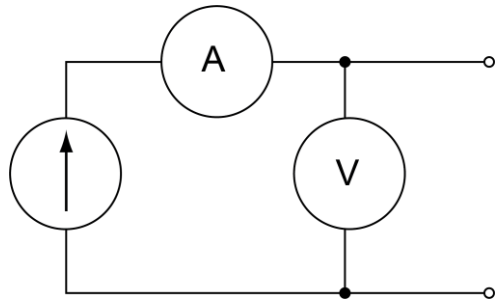
Figure 24: Fundamental source-measure configurations: Source V



A	Current meter
+	Voltage source
-	
V	Voltage meter

When sourcing current, you can measure voltage or current, as shown in the following figure.

Figure 25: Fundamental source-measure configuration: Source I



A	Current meter
↑	Current source
V	Voltage meter

See [Basic circuit configurations](#) (on page 5-5) for detailed information.

DUT test connections

WARNING

Hazardous voltages may be present in the test system. To prevent injury or death, remove power from the instrument or test system and discharge any energy storage components (for example, capacitors or cables) before changing any connections that might allow contact with an uninsulated conductor.

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.

Input/output connectors

The Keithley Instruments 2606B System SourceMeter® instrument uses screw terminal connectors for input and output connections to devices under test (DUTs).

You can remove a screw terminal connector from the rear panel by loosening the two captive retaining screws and pulling it off the rear panel. Each screw in the terminal connector cable assembly (used with the SMU connector) can accommodate from 24 AWG (0.2 mm²) to 12 AWG (2.5 mm²) conductors.

Basic connection sequence:

1. With the output off and the connector uninstalled from the 2606B rear panel, make the wire connections from a connector to the DUT.
2. Reinstall the connector onto the rear panel.
3. If using a screw terminal connector, tighten the two captive screws.

WARNING

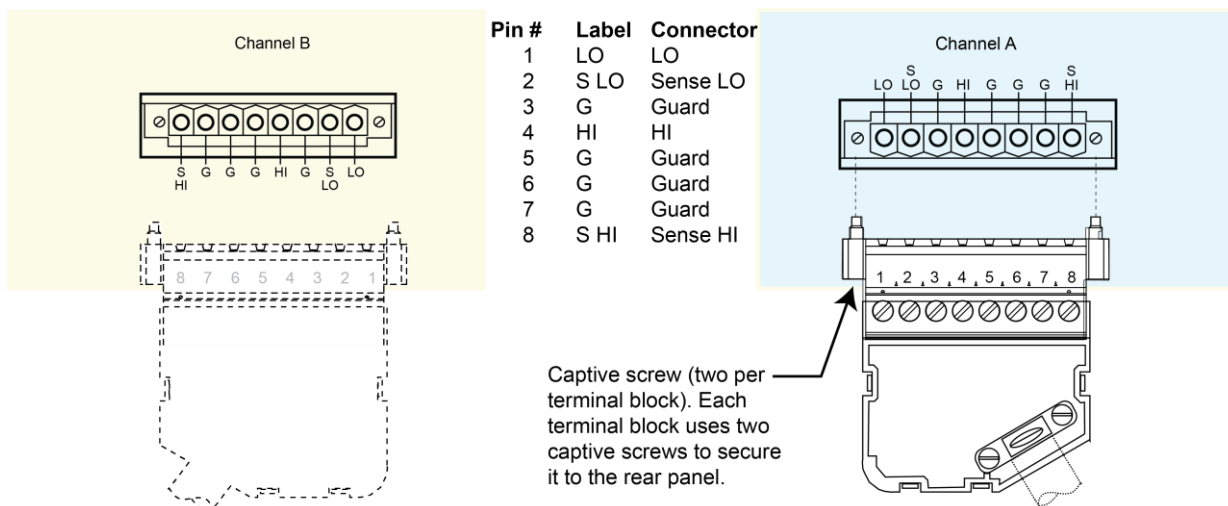
Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the 2606B while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of a System SourceMeter® instrument before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.

Maximum floating (common mode) voltage for a SMU is 250 V. Exceeding this level could damage the instrument and create a shock hazard. See [Floating a SMU](#) (on page 4-12) later in this section for details on floating the SMUs.

The input/output connectors of the System SourceMeter® instrument are rated for connection to circuits rated Measurement Category O only, with transients rated less than 1500 V_{PEAK}. Do not connect the 2606B terminals to CAT II, CAT III, or CAT IV circuits. Connections of the input/output connectors to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.

To prevent electric shock or damage to the System SourceMeter® instrument, when connecting to a source with a greater current capability than the 2606B, a user-supplied fuse, rated at no more than 20 A SLO-BLO, should be installed in-line with the 2606B input/output connectors.

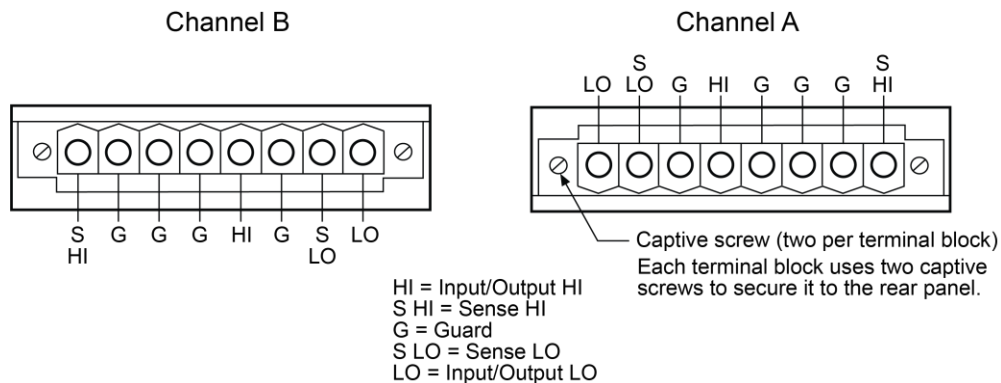
Figure 26: Input/output connectors



Input/output LO and chassis ground

The SMU input/output LOs are available at the rear-panel terminal blocks. Input/output LOs are not connected between channels and are electrically isolated from chassis ground.

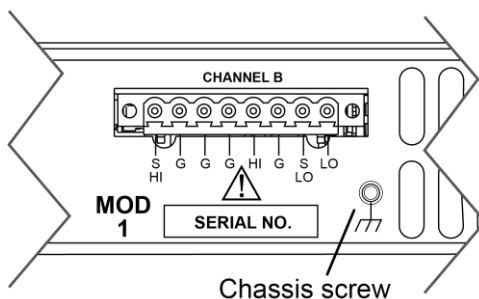
Figure 27: Input/output LO and chassis ground terminals



The chassis screw is on the rear panel of the instrument and is connected to the metal chassis of the 2606B. It is in the MOD 1 section of the rear panel, as shown in the following figure, but can be used for either module.

NOTE

The chassis should never be used as a ground point for signal connections. High frequencies (more than 1 MHz) present on the chassis of the 2606B may result in higher noise at the output. The chassis should only be used as a safety shield. Use the chassis screw for connections to the chassis of the 2606B.

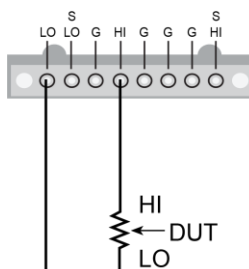
Figure 28: 2606B chassis screw

2-wire local sensing connections

You can use 2-wire local sensing measurements, shown in the following figure, for the following source-measure conditions:

- Sourcing and measuring current.
- Sourcing and measuring voltage in high impedance (more than 1 kΩ) test circuits.

When using 2-wire local sensing connections, make sure to properly configure the 2606B [Sense mode selection](#) (on page 5-11).

Figure 29: Two-wire resistance connections

4-wire remote sensing connections

When sourcing or measuring voltage in a low-impedance test circuit, there can be errors associated with lead resistance. Voltage source and measure accuracy are optimized by using 4-wire remote sense connections. When sourcing voltage, 4-wire remote sensing ensures that the programmed voltage is delivered to the DUT. When measuring voltage, only the voltage drop across the DUT is measured.

NOTE

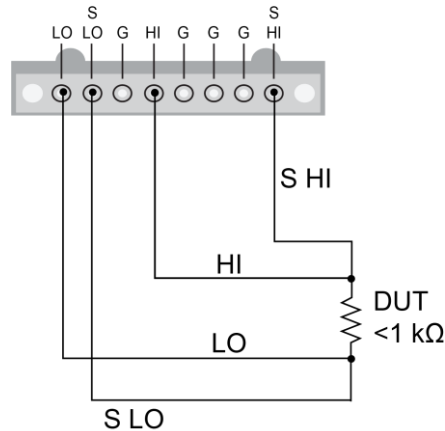
When sourcing voltage in remote sense, make sure the sense leads are connected to the DUT. If a sense lead becomes disconnected, an erroneous voltage will be sensed, and the 2606B may increase the output voltage to compensate. You can use contact check to verify that the sense leads are connected. Refer to [Contact check measurements](#) (on page 5-21).

Use 4-wire remote sensing for the following source-measure conditions:

- Sourcing or measuring voltage in low impedance ($<1\text{ k}\Omega$) test circuits.
- Enforcing voltage compliance limit directly at the DUT.

When using 4-wire local sensing connections, make sure to properly configure the 2606B [Sense mode selection](#) (on page 5-11).

Figure 30: Four-wire connections (remote sensing)



Contact check connections

The contact check function prevents measurement errors due to excessive resistance in the source or sense leads. See [Contact check measurements](#) (on page 5-21) for operation.

Contact check requires both source and sense connections. Refer to [4-wire remote sensing connections](#) (on page 4-6) for the connection scheme.

Guarding and shielding

You can optimize source-measure performance and safety with the effective use of guarding and shielding (noise and safety shields).

Safety shield

WARNING

A safety shield must be used whenever hazardous voltages ($>30 V_{\text{RMS}}$, $42 V_{\text{PEAK}}$) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the 2606B 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 device under test (DUT) test circuit. A metal safety shield must be connected to a known protective earth (safety ground). See [Test fixtures](#) (on page 4-11) for important safety information on the use of a metal or a nonconductive enclosure.

Safety shielding and hazardous voltages

The maximum output voltage for a 2606B channel is 20 V, which is considered a nonhazardous level. However, using multiple voltage sources in a series configuration or [floating a SMU](#) (on page 4-12) can cause test circuit voltage to exceed 42 V. For example, if a 2606B has LO driven to 100 V by a Model 2612B, the test circuit voltage could exceed 42 V. See [TSP advanced features](#) (on page 14-57) for information on using multiple System SourceMeter® instruments.

Use #18 AWG wire or larger for connections to safety earth ground and chassis.

Guarding

A driven guard is always enabled and provides a buffered voltage that is at the same level as the input/output HI voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between HI and LO. Without guarding, leakage and capacitance in the external high-impedance test circuit could be high enough to adversely affect the performance of the 2606B.

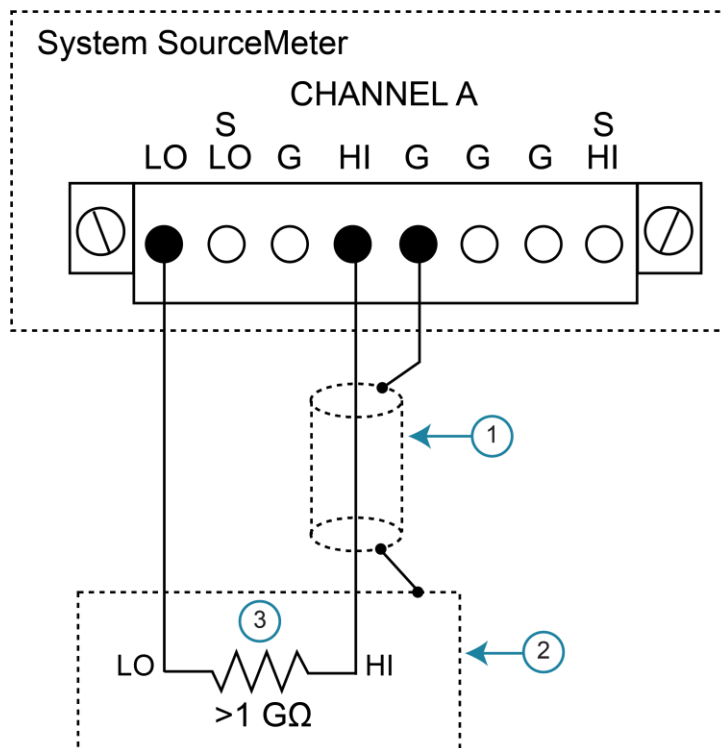
Guarding is recommended when test circuit impedance is $>1 \text{ G}\Omega$.

NOTE

See [Guard](#) (on page 8-12) for details on the principles of guarding.

The guard shield can be the shield of a coaxial cable or an insulated foil that surrounds the conductor.

Figure 31: Connections for guarding



- | |
|---------------------------|
| 1 Guard shield |
| 2 Metal guard shield |
| 3 Device under test (DUT) |

Noise shield

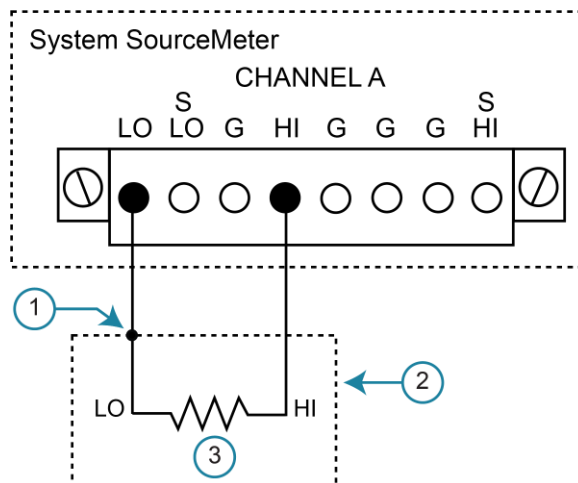
Use a noise shield (see the following figure) to prevent unwanted signals from being introduced into the test circuit. Low-level signals may benefit from effective shielding. The metal noise shield surrounds the test circuit and should be connected to LO, as shown.

⚠ WARNING

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.

⚠ WARNING

Connections to LO on the 2606B are not necessarily at 0 V. Hazardous voltages could exist between LO and chassis ground. Make sure that high-voltage precautions are taken throughout the test system. Alternatively, limit hazardous levels by adding external protection to limit the voltage between LO and chassis. Failure to make sure high-voltage precautions are used throughout the test system or a failure to limit hazardous levels could result in severe personal injury or death from electric shock.

Figure 32: Noise shield

1 Noise shield connected to LO

2 Noise shield

3 Device under test (DUT)

Using shielding and guarding together

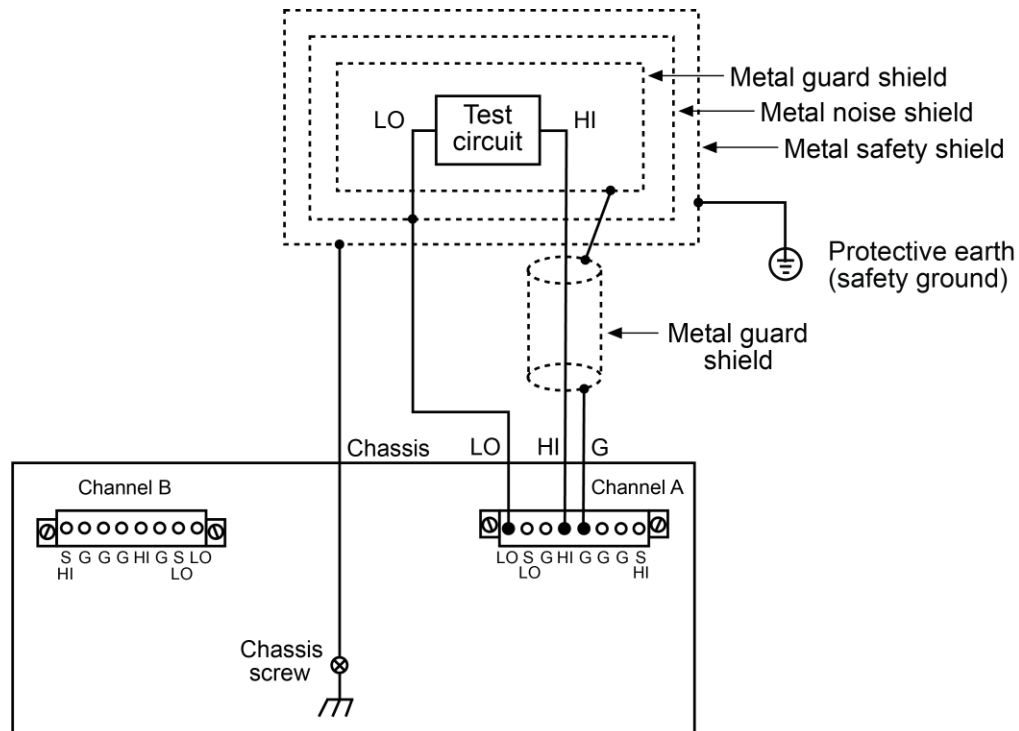
The following figures show connections for a test system that uses a noise shield, a safety shield, and guarding. The guard shields are connected to the driven guard (labeled G or GUARD, depending on your model) of the SMU. The noise shield is connected to LO. The safety shield is connected to the chassis and to protective earth (safety ground).

The following figures show connections for a test system that uses a noise shield, a safety shield, and guarding. The guard shields are connected to the driven guard (labeled G) of the module. The noise shield is connected to LO. The safety shield is connected to the chassis and to protective earth (safety ground).

⚠ WARNING

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.

Figure 33: Connections for noise shield, safety shield, and guarding



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 properly used, the output of the 2606B will turn off when the lid of the test fixture is opened. The test circuit is mounted inside the test fixture. When hazardous voltages ($>30\text{ V}_{\text{RMS}}$, $42\text{ V}_{\text{PEAK}}$) will be present, the test fixture must meet the following safety requirements.

⚠ 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).

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. Input/output connectors mounted on a metal test fixture 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 2606B digital I/O port provides an output enable line. When properly used with a test fixture, the output of the 2606B turns off when the lid of the test fixture opens.

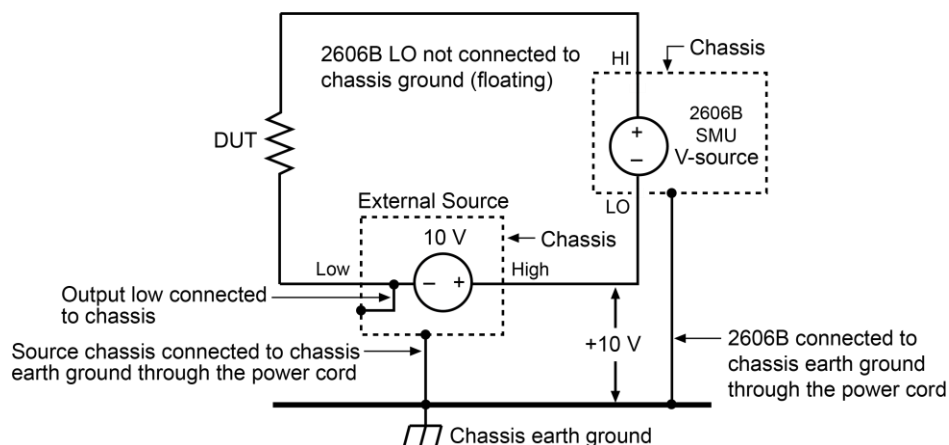
See [Digital I/O](#) (on page 7-37) for information on the digital I/O port.

Floating a SMU

Using an external source in the test system may require that a 2606B source-measure unit (SMU) float off chassis earth ground. An example of such a test system is shown below, which includes an external voltage source. Notice that output low of the external voltage source is connected to chassis earth ground.

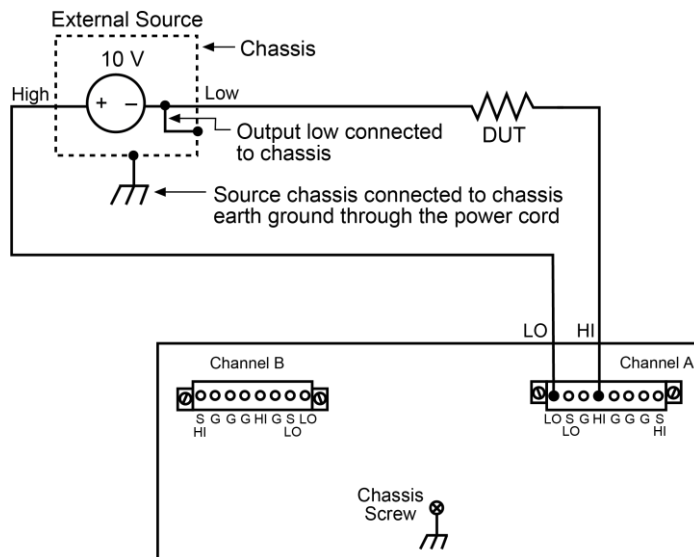
For the test circuit shown below, the 2606B must float off chassis earth ground. As shown, LO of the 2606B is floating +10 V above chassis earth ground. If LO of the 2606B was instead connected to chassis ground, the external voltage source would be shorted to the chassis ground.

Figure 34: Floating a SMU



The 2606B connections for the floating configuration are shown below.

Figure 35: Connections for floating a SMU



⚠ 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 (see [Safety shield](#) (on page 4-8)).

Saved setups

You can restore the 2606B to one of six nonvolatile-memory setup configurations (five user setups and one factory default), or to a setup stored on an external USB flash drive. As shipped from the factory, the 2606B powers up with the factory default settings, which cannot be overwritten. The default settings are also in the five user setup locations, but may be overwritten. The factory default settings are listed in the command descriptions in the [TSP command reference](#) (on page 15-1).

You can also change the setup configuration that is used when the instrument powers up.

Saving user setups

You can save the present 2606B setup to internal nonvolatile memory or a USB flash drive.

To save a user setup to nonvolatile memory from the virtual front panel:

1. Configure the 2606B to the settings that you want to save.
2. Press the **MENU** key.
3. Select **SETUP** and then press the **ENTER** key.
4. Select the **SAVE** menu item and then press the **ENTER** key.
5. Select **INTERNAL** and then press the **ENTER** key.
6. Select the user number (1 through 5) and press the **ENTER** key.

To save a user setup to an external USB flash drive from the virtual front panel:

1. Configure the 2606B to the settings that you want to save.
2. Insert the USB flash drive into the USB port on the front panel of the 2606B.
3. Press the **MENU** key.
4. Select **SETUP** and then press the **ENTER** key.
5. Select **SAVE** and then press the **ENTER** key.
6. Select **USB1**. The file name `setup000.set` is displayed.
7. Turn the navigation wheel to change the last three digits of the file name and then press the **ENTER** key.

To save and recall user setups using remote commands:

Use the `setup.save()` and `setup.recall()` functions to save and recall user setups. The following example saves the present setup as setup 1, and then recalls setup 1.

```
-- Save the present setup to nonvolatile memory.  
setup.save(1)  
-- Recall the saved user setup from nonvolatile memory.  
setup.recall(1)
```

Recalling a saved setup using the front panel

You can recall setups from internal nonvolatile memory or a USB flash drive.

To recall a saved setup from the virtual front panel:

1. Press the **MENU** key to access the main menu.
2. Select **SETUP**, and then press the **ENTER** key.
3. Select the **RECALL** menu item, and then press the **ENTER** key.
4. Select one of the following:
 - **INTERNAL**
 - **USB1**
5. For **INTERNAL** only, do one of the following:
 - Select **FACTORY** to restore factory defaults, then press the **ENTER** key.
 - Select the user number (1 through 5), then press the **ENTER** key.
6. **USB1** only: Select the appropriate file and then press the **ENTER** key.

Start-up configuration

The start-up configuration allows you to apply specific settings every time the instrument powers up. The settings are defined in a user setup. You can also select the factory default setup.

To select the power-on setup from the virtual front panel:

1. Press the **MENU** key to access the main menu.
2. Select **SETUP**, and then press the **ENTER** key.
3. Select **POWERON**, and then press the **ENTER** key.
4. Select the configuration to use.
5. Press the **ENTER** key.
6. Use the **EXIT (LOCAL)** key to return to the main display.

To select the power-on setup using remote commands:

Use the `setup.poweron` attribute to select which setup to use when the instrument powers up. To set the `setup.poweron` configuration attribute:

```
setup.poweron = n
```

Where *n* is:

- 0 (*RST or `reset()` factory defaults)
- 1 to 5 (user setup 1 to 5)

Restoring the factory default setups using remote commands

Use one of the reset functions to return the 2606B to the original factory defaults. An example of each type of reset is shown in the following program examples.

Restore all factory defaults of all nodes on the TSP-Link® network:

```
reset()
```

Restore all factory defaults (note that you cannot use `*rst` in a script):

```
*rst
```

Restore all factory defaults:

```
setup.recall(0)
```

Restore channel A defaults:

```
smua.reset()
```

Reset only the local TSP-Link node:

```
localnode.reset()
```

USB storage overview

The 2606B includes a USB port on the front panel of each module. To store scripts and to transfer files from the instrument to the host computer, you need a USB flash drive.

For information about saving reading buffers to a USB flash drive, see [Saving reading buffers](#) (on page 6-5).

For information about storing and loading scripts to and from a USB flash drive, see [Save a user script](#) (on page 14-11).

For information about file I/O, see [File I/O](#) (on page 13-7).

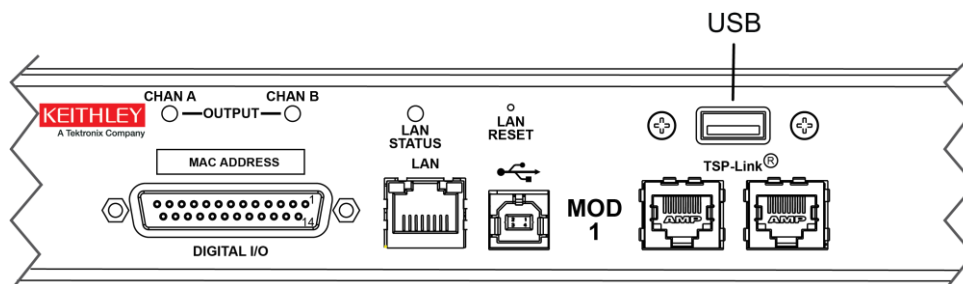
For information about saving user setups, see [Saved setups](#) (on page 4-13).

Connecting the USB flash drive

The 2606B supports flash drives that comply with USB 2.0 standards, 1.0, and 1.1 standards. You can save data to the USB flash drive from the virtual front panel, or you can create a script to save data to the USB flash drive.

To connect the USB flash drive, plug the USB flash drive into the USB port on the instrument's front panel (see the figure below).

Figure 36: USB-A location on Module 1



File system navigation

The 2606B can use commands from the Lua `fs` library to navigate and list files that are available on a flash drive. These Lua commands are in the `fs` command group in the instrument.

The `fs` commands make the file system of any given node available to the entire TSP-Link® system. For example, you can use the command `node[5].fs.readdir(". ")` to read the contents of the current working directory on node 5.

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 generally need to use a double backslash (\\) when you are creating scripts or sending commands to the instrument.

The instrument supports the following Lua `fs` commands:

[fs.chdir\(\)](#) (on page 15-103)

[fs.cwd\(\)](#) (on page 15-104)

[fs.is_dir\(\)](#) (on page 15-104)

[fs.is_file\(\)](#) (on page 15-105)

[fs.mkdir\(\)](#) (on page 15-106)

[fs.readdir\(\)](#) (on page 15-107)

[fs.rmdir\(\)](#) (on page 15-107)

The following Lua `fs` commands are not supported:

`fs.chmod()`

`fs.chown()`

`fs.stat()`

Displayed error and status messages

During operation and programming, virtual front-panel messages may be briefly displayed. Typical messages are either status or error notifications (refer to the [Error summary list](#) (on page 12-2) for a complete list of these messages and their meanings).

NOTE

Status and error messages are held in a queue. For information about retrieving messages from queues, refer to [Queues](#) (on page 10-2). For information about error messages, refer to the [Troubleshooting guide](#) (on page 12-1).

Sourcing and measuring

In this section:

Basic operation	5-1
Basic circuit configurations.....	5-5
Operation considerations for the ADC.....	5-10
Sense mode selection.....	5-11
Triggering in local mode.....	5-12
Configuring for measure-only tests using the MODE key.....	5-14
Voltmeter and ammeter measurements	5-15
Ohms measurements.....	5-15
Power measurements	5-19
Contact check measurements.....	5-21
Multiple SMU connections.....	5-24
Output-off modes	5-25
Range	5-29
Digits	5-32
Speed.....	5-33
Sweep operation	5-34
Relative offset	5-47
Filters	5-49
High-capacitance mode	5-52

Basic operation

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 2606B 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 2606B before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

NOTE

Before you begin any of the following virtual front-panel procedures, make sure that you exit out of the menu structure. Press the **EXIT (LOCAL)** key as many times as needed to return to the main display.

Source-measure capabilities

From the virtual front panel, you can configure the instrument to perform the following source-measure operations:

- **Source voltage:** Measure and display current, voltage, resistance, or power
- **Source current:** Measure and display voltage, current, resistance, or power
- **Measure resistance:** Display resistance calculated from voltage and current components of measurement (can optionally specify source voltage or source current value)
- **Measure power:** Display power calculated from voltage and current components of measurement (can optionally specify source voltage or source current value)
- **Measure only (V or I):** Display voltage or current measurement

Virtual front-panel source-measure procedure

Use the following procedure to perform the basic source-measure operations of the 2606B SMU using the virtual front panel. The following procedure assumes that the 2606B is already connected to the device under test (DUT), as explained in [DUT test connections](#) (on page 4-3).

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 2606B 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 2606B before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Step 1: Select and set the source level

To select the source and edit the source value:

1. Press the **SRC** key as needed to select the voltage source or current source, as indicated by the units in the source field on the display. The flashing digit (cursor) indicates which value is presently selected for editing.
2. Move the cursor to the digit to change, then press the navigation wheel to enter the EDIT mode.
3. Use the **RANGE** keys to select a range that accommodates the value you want to set. For best accuracy, use the lowest possible source range.
4. Enter the source value.
5. Press the **ENTER** key or the navigation wheel to complete editing.

Step 2: Set the compliance limit

To edit the compliance limit value:

1. Select the **CONFIG** key.
2. Select the **LIMIT** key.
3. Select **CURRENT** or **VOLTAGE**.
4. Select the **ENTER** key or the navigation wheel.
5. Move the cursor to the digit to change, then press the navigation wheel to enter the EDIT mode, as indicated by the EDIT indicator.
6. Enter the limit value, then press the **ENTER** key or the navigation wheel to complete editing.

Step 3: Select the measurement function and range

To select the measurement function and range:

1. Select the **DISPLAY** key to place it in single-channel-display mode.
2. Select the measurement function by pressing the **MEAS** key.
3. Set the measurement range with the **RANGE** keys, or enable **AUTO** range. When setting the range, consider the following points:
 - When measuring the source (such as when sourcing V and measuring V), you cannot select the measurement range using the RANGE keys. The selected source range determines the measurement range.
 - When not measuring the source (such as when sourcing V but measuring I), you can select the measurement range selection manually or the instrument can set it automatically. When using manual ranging, use the lowest possible range for best accuracy. When autorange is enabled, the 2606B automatically goes to the most sensitive range to make the measurement.

Step 4: Turn the output on

To turn the output on, select the **OUTPUT ON/OFF** switch on the virtual front panel. The OUTPUT indicator light turns on.

Step 5: Observe readings on the display.

Press the **TRIG** key if necessary to trigger the instrument to begin making readings. The readings are on the top line, and source and limit values are on the bottom line.

Step 6: Turn the output off

When finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Remote source-measure commands

Basic source-measurement procedures can also be performed through a remote interface. To do this, send the appropriate commands. The following table summarizes basic source-measure commands. See [Introduction to TSP operation](#) (on page 13-1) for more information on using these commands.

Basic source-measure commands

Command*	Description
<code>smuX.measure.autorangei = smuX.AUTORANGE_ON</code>	Enable current measure autorange.
<code>smuX.measure.autorangev = smuX.AUTORANGE_ON</code>	Enable voltage measure autorange.
<code>smuX.measure.autorangei = smuX.AUTORANGE_OFF</code>	Disable current measure autorange.
<code>smuX.measure.autorangev = smuX.AUTORANGE_OFF</code>	Disable voltage measure autorange.
<code>smuX.measure.rangei = rangeval</code>	Set current measure range.
<code>smuX.measure.rangev = rangeval</code>	Set voltage measure range.
<code>reading = smuX.measure.i()</code>	Request a current reading.
<code>reading = smuX.measure.v()</code>	Request a voltage reading.
<code>iReading, vReading = smuX.measure.iv()</code>	Request a current and voltage reading.
<code>reading = smuX.measure.r()</code>	Request a resistance reading.
<code>reading = smuX.measure.p()</code>	Request a power reading.
<code>smuX.source.autorangei = smuX.AUTORANGE_ON</code>	Enable current source autorange.
<code>smuX.source.autorangev = smuX.AUTORANGE_ON</code>	Enable voltage source autorange.
<code>smuX.source.autorangei = smuX.AUTORANGE_OFF</code>	Disable current source autorange.
<code>smuX.source.autorangev = smuX.AUTORANGE_OFF</code>	Disable voltage source autorange.
<code>smuX.source.func = smuX.OUTPUT_DCVOLTS</code>	Select voltage source function.
<code>smuX.source.func = smuX.OUTPUT_DCAMPS</code>	Select current source function.
<code>smuX.source.leveli = sourceval</code>	Set current source value.
<code>smuX.source.levelv = sourceval</code>	Set voltage source value.
<code>smuX.source.limiti = level</code>	Set current limit.
<code>smuX.source.limitv = level</code>	Set voltage limit.
<code>smuX.source.limitp = level</code>	Set power limit.
<code>smuX.source.output = smuX.OUTPUT_ON</code>	Turn on source output.
<code>smuX.source.output = smuX.OUTPUT_OFF</code>	Turn off source output.
<code>smuX.source.rangei = rangeval</code>	Set current source range.
<code>smuX.source.rangev = rangeval</code>	Set voltage source range.
<code>smuX.sense = smuX.SENSE_LOCAL</code>	Select local sense (2-wire).
<code>smuX.sense = smuX.SENSE_REMOTE</code>	Select remote sense (4-wire).

Requesting readings

You can request readings by including the appropriate measurement command as the argument for the `print()` command. The following programming example illustrates how to request a channel A current reading:

```
print(smuA.measure.i())
```

Source-measure programming example

The following SMU programming example illustrates the setup and command sequence of a basic source-measure procedure with the following parameters:

- Source function and range: Voltage, autorange
- Source output level: 5 V
- Current compliance limit: 10 mA
- Measure function and range: Current, 10 mA

```
-- Restore 2606B defaults.
smua.reset()
-- Select voltage source function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set source range to autorange.
smua.source.autorangev = smua.AUTORANGE_ON
-- Set voltage source to 5 V.
smua.source.levelv = 5
-- Set current limit to 10 mA.
smua.source.limiti = 10e-3
-- Set current range to 10 mA.
smua.measure.rangei = 10e-3
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Print and place the current reading in the reading buffer.
print(smua.measure.i(smua.nvbuffer1))
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
```

Basic circuit configurations

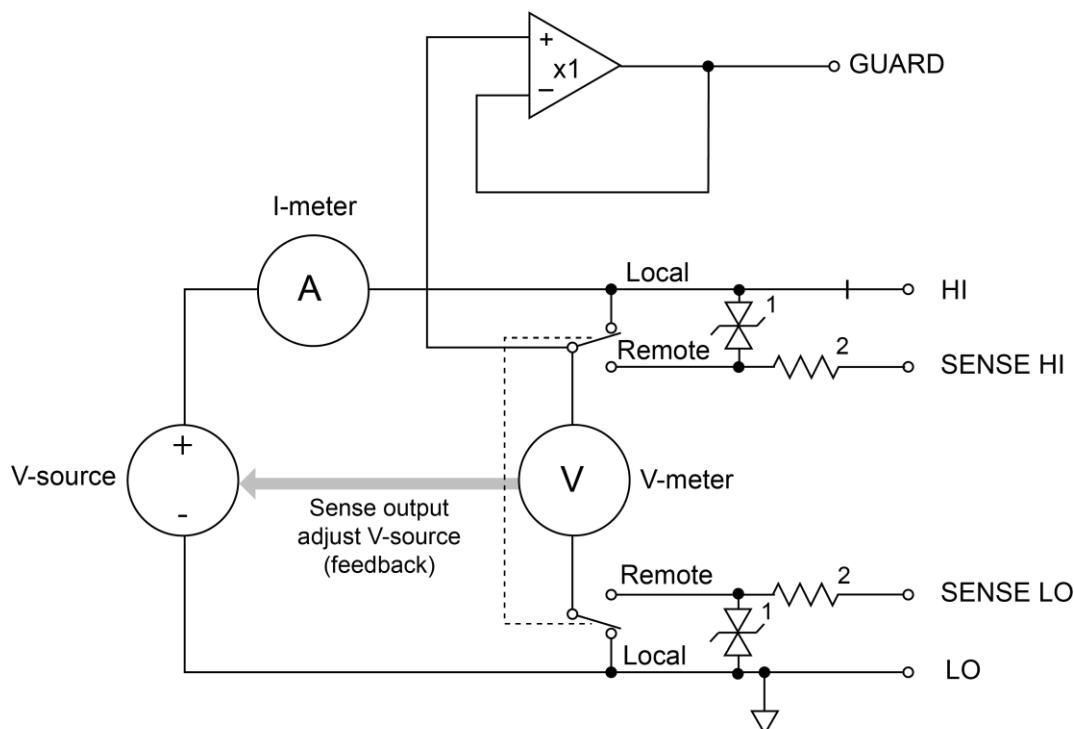
The following topics describe the source, measure, and contact check circuit configurations.

Source V

When configured to source voltage (V-source) as shown in the figure below, the 2606B functions as a low-impedance voltage source with current limit capability and can measure current (I-meter) or voltage (V-meter).

Sense circuitry is used to monitor the output voltage continuously and make adjustments to the V-source as needed. The V-meter senses the voltage at the HI / LO terminals (2-wire local sense) or at the device-under-test (DUT) (4-wire remote sense using the sense terminals) and compares it to the programmed voltage level. If the sensed level and the programmed value are not the same, the V-source is adjusted accordingly. Remote sense eliminates the effect of voltage drops in the test leads, ensuring that the exact programmed voltage appears at the DUT. With 4-wire sensing enabled, both remote sense leads must be connected to prevent incorrect operation. Use contact check to verify that the sense leads are connected. Refer to [Contact check measurements](#) (on page 5-21) for more information.

Figure 37: Source V configuration



- NOTES: 1. This represents a protection circuit that is very high impedance until the voltage across it exceeds approximately 3 V. Above 3 V, the protection turns on and allows current to flow through it.
 2. Approximately 13 kΩ.

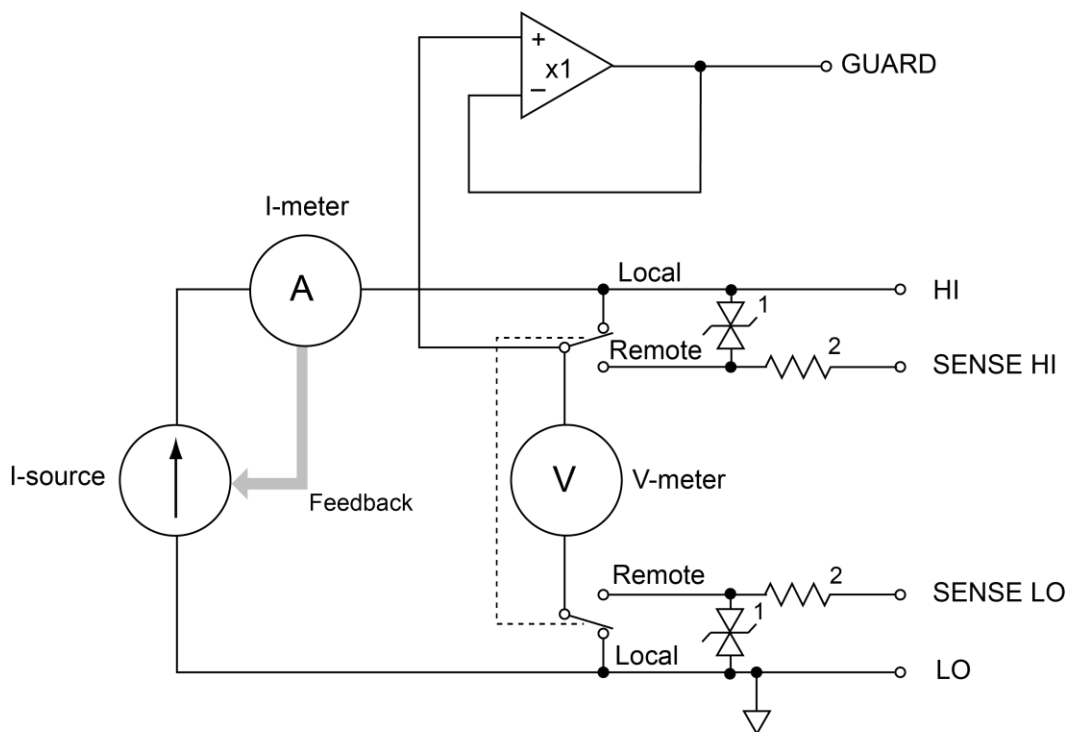
Source I

When the instrument is configured to source current (I-source), as shown in the figure below, the instrument functions as a high-impedance current source with voltage limit capability and can measure current (I-meter) or voltage (V-meter).

For 2-wire local sensing, voltage is measured at the HI / LO terminals of the instrument. For 4-wire remote sensing, voltage is measured directly at the device-under-test (DUT) using the sense terminals. This eliminates any voltage drops that may be in the test leads or connections between the instrument and the DUT.

The current source does not require or use the sense leads to enhance current source accuracy. However, if the instrument is in 4-wire remote sense mode, the instrument may reach limit levels if the sense leads are disconnected. With 4-wire remote sensing selected, the sense leads must be connected or incorrect operation results.

Figure 38: Source I configuration



- NOTES: 1. This represents a protection circuit that is very high impedance until the voltage across it exceeds approximately 3 V. Above 3 V, the protection turns on and allows current to flow through it.
2. Approximately 13 kΩ.

Source I measure I, source V measure V

The System SourceMeter® instrument can measure the same function that it is sourcing. For example, when sourcing a voltage, you can measure voltage. Conversely, if you are sourcing current, you can measure the output current. For these operations, the measure range is the same as the source range.

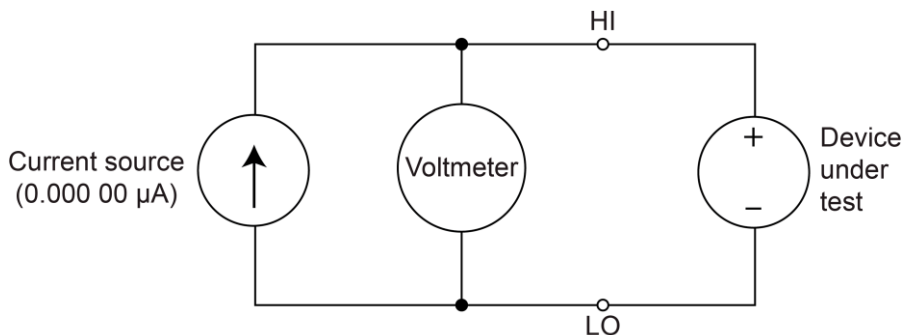
This feature is valuable when operating with the source in compliance. When in compliance, the programmed source value is not reached, so measuring the source lets you measure the actual output level.

Measure only (voltage or current)

The figures below show the configurations for using the instrument exclusively as a voltmeter or ammeter.

As shown in the following figure, to configure the instrument to measure voltage only, set it to source 0 A and measure voltage.

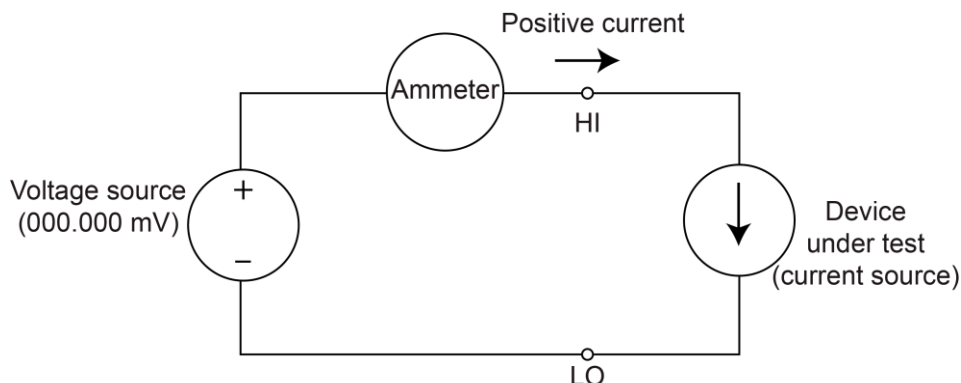
Figure 39: 2606B measure voltage only



CAUTION

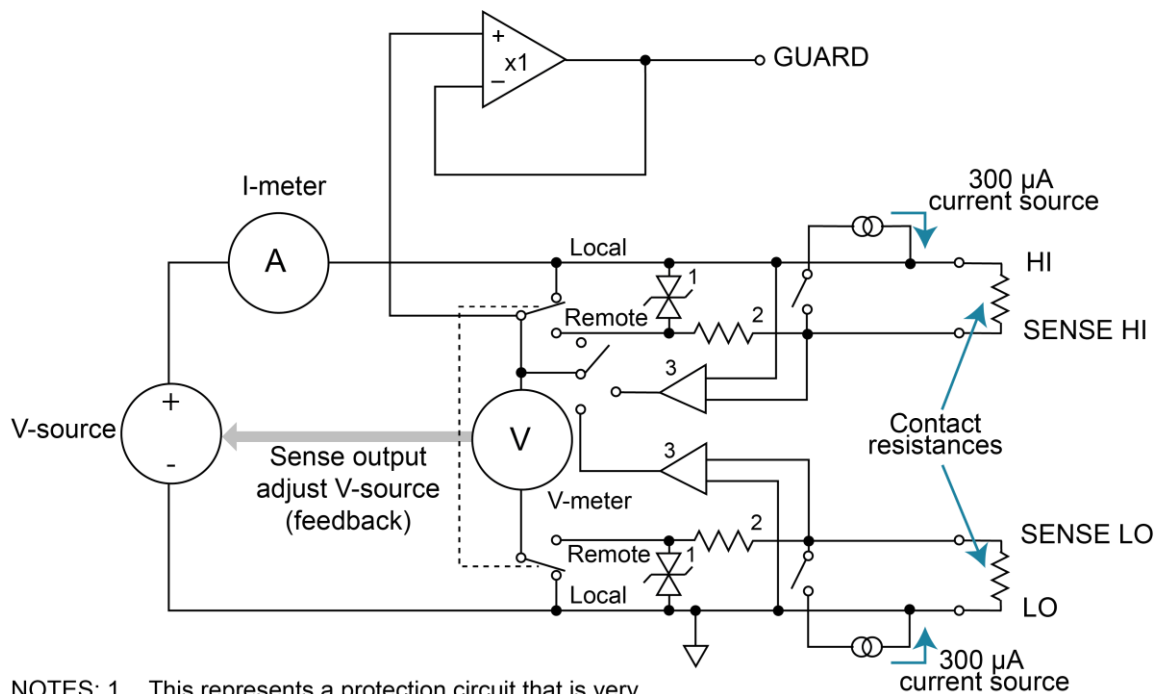
Set the voltage limit to a level that is higher than the measured voltage. If the voltage limit is set to a level that is lower than the measured voltage, excessive current flows into the instrument. This current could damage the instrument. Also, when connecting an external energy source to the instrument when it is configured as a current source, set the output off state to the high-impedance mode. See [Output-off states](#) (on page 5-25) for more information on the output-off states. Refer to [Limits](#) (on page 4-1) for details on compliance limits.

In the following figure, the instrument uses a 2-wire local sensing configuration and is set to measure current only by setting it to source 0 V and measure current. Note that to obtain positive (+) readings, conventional current must flow from HI to LO.

Figure 40: 2606B measure current only

Contact check

When a contact check measurement is made, two small current sources switch between the HI and SENSE HI terminals and the LO and SENSE LO terminals. By controlling the switches illustrated in the following figure, the current from these sources flows through the test leads and through the contact resistance, as shown. To accurately measure the resulting contact resistance, the differential amplifier outputs are measured once with the current sources connected, and again with the current sources disconnected. This allows for compensation of various offset voltages that can occur.

Figure 41: Contact check circuit

- NOTES: 1. This represents a protection circuit that is very high impedance until the voltage across it exceeds approximately 3 V. Above 3 V, the protection turns on and allows current to flow through it.
 2. Approximately 13 kΩ.
 3. High impedance differential amplifier.

Operation considerations for the ADC

The following paragraphs discuss autozero and NPLC caching with the [analog-to-digital converter \(ADC\)](#) (on page 8-14).

Autozero

The ADC of the 2606B uses a ratiometric analog to digital (A/D) conversion technique. To ensure reading accuracy, the instrument must periodically obtain fresh measurements of its internal ground and voltage reference. Separate reference and zero measurements are used for each aperture.

As summarized in the table below, there are different settings for autozero. By default, the instrument is set to AUTO, which automatically checks these reference measurements whenever a signal measurement is made. If the reference measurements are out of date when a signal measurement is made, the instrument automatically makes two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes longer than normal.

This extra time can cause problems in sweeps and other test sequences in which measurement timing is critical. To avoid the extra time for the reference measurements in these situations, you can select OFF. This setting disables the automatic reference measurements. Note that with automatic reference measurements disabled, the instrument may gradually drift out of specification.

To minimize the drift, make a reference and zero measurement immediately before a critical test sequence. You can use the ONCE setting to force a refresh of the reference and zero measurements used for the current aperture setting.

Autozero settings

Autozero setting	Description
OFF	Turns automatic reference measurements off.
ONCE	After immediately making one reference and one zero measurement, turns automatic reference measurements off.
AUTO	Automatically makes new acquisitions when the 2606B determines reference and zero values are out-of-date.

Setting autozero from the virtual front panel

To change autozero from the virtual front panel:

1. Press the **CONFIG** key.
2. Press the **MEAS** key.
3. Turn the navigation wheel to select **AUTO-ZERO**, and then press the **ENTER** key or the navigation wheel.
4. Turn the navigation wheel to select the mode (**OFF**, **ONCE**, or **AUTO**), and then press the **ENTER** key or the navigation wheel.
5. Press the **EXIT (LOCAL)** key to return to the previous display.

Remote command autozero

To set autozero from a remote interface:

Use the autozero command with the appropriate option shown in the following table to set autozero through a remote interface (see [smuX.measure.autozero](#) (on page 15-212)). For example, send the following command to activate channel A automatic reference measurements:

```
smua.measure.autozero = smua.AUTOZERO_AUTO
```

Autozero commands and options

Command	Description
<code>smuX.measure.autozero = smuX.AUTOZERO_OFF</code>	Disable autozero. Old NPLC cache values are used when autozero is disabled (see NPLC caching (on page 5-11)).
<code>smuX.measure.autozero = smuX.AUTOZERO_ONCE</code>	After immediately taking one reference and one zero measurement, turns automatic reference measurements off.
<code>smuX.measure.autozero = smuX.AUTOZERO_AUTO</code>	Automatically takes new acquisitions when the 2606B determines reference and zero values are out-of-date.

NPLC caching

NPLC caching speeds up operation by caching A/D reference and zero values for up to the ten most recent measurement aperture settings. Whenever the integration rate is changed using the SPEED key, or a user setup is recalled, the NPLC cache is checked. If the integration rate is already stored in the cache, the stored reference and zero values are recalled and used. If the integration rate is not already stored in the cache, a reference and zero value is acquired and stored in the cache when the next measurement is made. If there are already ten NPLC values stored, the oldest one is overwritten by the newest one. When autozero is off, NPLC values stored in the cache are used, regardless of age.

Sense mode selection

You can set the sense mode to use [2-wire local sensing connections](#) (on page 4-6) or [4-wire remote sensing connections](#) (on page 4-6). The default sense setting is 2-wire local.

Front-panel sense mode selection

To check or change the voltage sense mode from the virtual front panel:

1. Press the **CONFIG** key.
2. Press the **SRC** or **MEAS** key. You can access and set the 2606B sense mode from either the V-SOURCE or the V-MEAS menu items.
3. If you pressed the SRC key: Select **V-SOURCE** > **SENSE-MODE**, and then press the **ENTER** key or the navigation wheel.
If you pressed the MEAS key: Select **V-MEAS** > **SENSE-MODE**, and then press the **ENTER** key or the navigation wheel.
4. Select **2-WIRE** or **4-WIRE** as needed, and then press the **ENTER** key or the navigation wheel.

Selecting the sense from the remote interface

To select the remote sense from the remote interface:

Set the `smuX.sense` attribute to control the sense state by remote. The programming example below illustrates how to configure the 2606B for 4-wire remote sensing:

```
smua.sense = smua.SENSE_REMOTE
```

The following table summarizes the commands to select the sense mode. See [Remote source-measure commands](#) (on page 5-4) and [TSP command reference](#) (on page 15-1) for details on using these commands.

Commands to select sense mode

Command	Description
<code>smuX.source.output = smuX.OUTPUT_OFF</code>	Turn off the source-measure unit (SMU) output.
<code>smuX.sense = smuX.SENSE_LOCAL</code>	Select local (2-wire) sense.
<code>smuX.sense = smuX.SENSE_REMOTE</code>	Select remote (4-wire) sense.

Triggering in local mode

You do not need to change any trigger settings to use the basic source and measurement procedures described in the following topics.

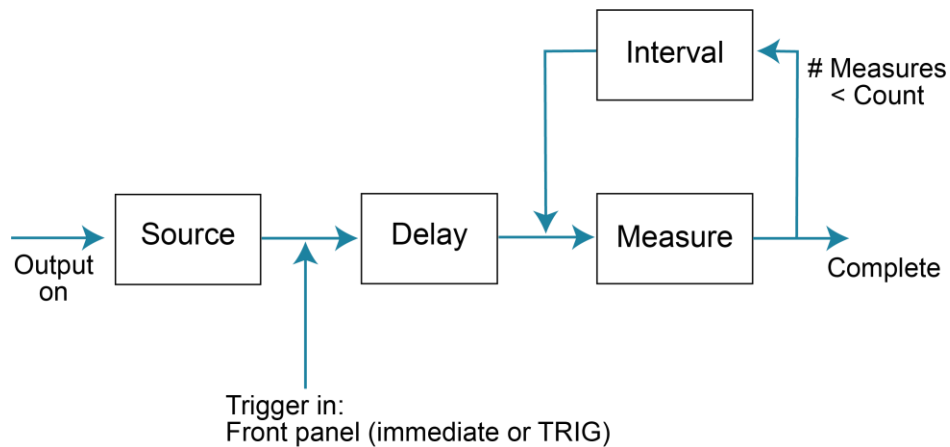
NOTE

Press the **MENU** key, and then select **SETUP > RECALL > INTERNAL > FACTORY** to reset the factory default conditions.

The following figure shows the general sequence for SMU measurement triggering. The basic sequence is as follows:

- When the output is turned on, the programmed source value is immediately applied to the device under test (DUT).
- Through the virtual front panel only: If the immediate trigger source is selected, a measurement is triggered immediately. However, if the manual trigger source is selected, the virtual front-panel TRIG key must be pressed.
- The instrument waits for the programmed delay period (if any).
- The instrument makes one measurement.
- If the number of measurements is less than the programmed trigger count, it cycles to make another measurement (the measurement cycle is repeated indefinitely if the infinite trigger count is selected).
- For multiple measurements, the instrument waits for the programmed trigger interval (if any) before making the next measurement.

Figure 42: Local triggering



Configuring trigger attributes in local mode

From the virtual front panel, press the **CONFIG** key, and then select **TRIG**. The following menu items are available:

- **TRIGGER-IN:** Use these options to select the trigger-in source:
 - **IMMEDIATE:** Triggering occurs immediately and the instrument starts to make measurements when it is ready (for example, after the source output is turned on).
 - **MANUAL:** The virtual front-panel TRIG key must be pressed to trigger the instrument to make readings.
- **COUNT:** Sets the trigger count (number of measurements) as follows:
 - **FINITE:** The instrument goes through measurement cycles for the programmed trigger count (1 to 99999).
 - **INFINITE:** The instrument goes through measurement cycles indefinitely until halted.
- **INTERVAL:** Sets the time interval between measurements (0 s to 999.999 s) when the count is greater than 1.
- **DELAY:** Sets the delay period between the trigger and the start of measurement (0 s to 999.999 s).

Front-panel triggering example

This example uses the virtual front panel to configure the trigger parameters to meet the following requirements:

- Manual triggering (TRIG key)
- Infinite trigger count (cycle indefinitely through measurement cycles)
- Interval (time between measurements): 1 s
- Delay (time from trigger to measurement): 2 s

To configure the trigger parameters:

1. Press the **CONFIG** key, and then the **TRIG** key.
2. Select **TRIGGER-IN**, and then press the **ENTER** key or the navigation wheel.
3. Select **MANUAL**, and then press the **ENTER** key or the navigation wheel.
4. Select **COUNT**, then select **INFINITE**, and then press the **ENTER** key or the navigation wheel.
5. Select **INTERVAL**, set the interval to 1 s, and then press the **ENTER** key or the navigation wheel.
6. Choose **DELAY**, set the delay to 2 s, and then press the **ENTER** key or the navigation wheel.
7. Use the **EXIT (LOCAL)** key to return to the normal display.
8. Press the **OUTPUT ON/OFF** control to turn the output on.
9. Press the **TRIG** key. A 2 s delay occurs before the first measurement. The instrument cycles through measurements indefinitely with a 1 s interval between measurements.
10. Press the **OUTPUT ON/OFF** control again to stop making readings.

Configuring for measure-only tests using the MODE key

In addition to using the 2606B for conventional source-measure operations, you can also use it like a meter to measure current, voltage, resistance, or power.

To configure the 2606B as a voltage meter, current meter, ohmmeter, or wattmeter:

1. Press the **MODE** key.
2. Turn the navigation wheel to select the type of meter from the menu (**I-METER**, **V-METER**, **OHM-METER**, or **WATT-METER**).
3. Press the **ENTER** key to complete the configuration of the 2606B as the selected meter.

To manually configure the settings, refer to the following topics:

- [Voltmeter and ammeter measurements](#) (on page 5-15)
- [Ohms measurements](#) (on page 5-15)
- [Power measurements](#) (on page 5-19)

Voltmeter and ammeter measurements

You can make voltmeter and ammeter measurements without using the MODE key, such as when configuring measure-only tests over the remote interface.

To use the 2606B to measure voltage or current:

1. Select the source-measure functions:
 - **Voltmeter:** Press the **SRC** key to select the current source and press the **MEAS** key to select the voltage measurement function.
 - **Ammeter:** Press the **SRC** key to select the voltage source and press the **MEAS** key to select the current measurement function.
2. Set source and compliance levels. To edit the source level, use the procedure provided in [Step 1: Select and set the source level](#) (on page 5-2); to edit the compliance level, use the procedure provided in [Step 2: Set the compliance limit](#) (on page 5-3):
 - Select the lowest source range and set the source level to zero.
 - Set the compliance level to a value that is higher than the expected measurement.

CAUTION

When using the 2606B as a voltmeter, the voltage compliance limit must be set higher than the voltage that is being measured. Failure to do this could result in excessive current flow into the 2606B, incorrect measurements, and possible damage to the instrument.

3. Use the **RANGE** keys to select a fixed measurement range that accommodates the expected reading. Use the lowest possible range for best accuracy. You can also select autorange, which automatically sets the 2606B to the most sensitive range.
4. Connect the voltage or current to be measured. Make sure to use 2-wire connections from the 2606B to the device under test (DUT). Refer to [DUT test connections](#) (on page 4-3).
5. Press the **OUTPUT ON/OFF** control to turn the output on.
6. View the displayed reading (press the **TRIG** key if necessary).
7. When finished, press the **OUTPUT ON/OFF** control to turn the output off.

Ohms measurements

Resistance readings are calculated from the measured current and measured voltage as follows:

$$R = V/I$$

Where:

- R is the calculated resistance
- V is the measured voltage
- I is the measured current

Ohms ranging

The virtual front-panel ohms function does not use ranging. The instrument formats a calculated resistance reading (V/I) to best fit the display. There may be leading zeros if the ohms reading is less than 1 mΩ.

Basic ohms measurement procedure

When you use the MODE key to select ohms measurement, the 2606B is automatically configured as a current source with a level of 1 mA. To change the source function, source value, or compliance value (in other words, customize the standard ohmmeter configuration of the MODE key), then use the following steps to make ohms measurements. The following procedure assumes that the 2606B is already connected to the device under test (see [DUT test connections](#) (on page 4-3)).

To make an ohms measurement:

1. Press the **DISPLAY** key to select single-channel display mode. See [Display mode](#) (on page 3-34).
2. Press the **SRC** key to select the source function.
3. Set the output source (current or voltage, dependent on which function is selected) to a value based on the expected resistance. See [Step 1: Select and set the source level](#) (on page 5-2).
4. Press the **LIMIT** key to edit the voltage or current limit. When programming a voltage limit, set the voltage limit above the maximum expected voltage across the resistor under test. When programming a current limit, set the current limit at or above the maximum expected current through the resistor under test. See [Step 2: Set the compliance limit](#) (on page 5-3).
5. Press the **MEAS** key to display voltage or current.
6. Make sure that AUTO measurement range is on (press the **AUTO** key if needed).
7. Press the **MEAS** key as many times as needed to display ohms.
8. Press the **OUTPUT ON/OFF** control to turn the output on.
9. View the displayed reading (press the **TRIG** key if necessary). When finished, press the **OUTPUT ON/OFF** control again to turn the output off.

Remote ohms command

Use the `smuX.measure.r()` function to get a resistance reading.

The programming example below illustrates how to get a resistance reading from SMU A:

```
reading = smua.measure.r()
```

See [Remote source-measure commands](#) (on page 5-4) for more commands to set up source and measure functions, and [Introduction to TSP operation](#) (on page 13-1) for more details.

Ohms programming example

The following programming example illustrates the setup and command sequence of a typical ohms measurement procedure with the following parameters:

- **Source function:** Current, 10 mA range, 10 mA output
- **Voltage measure range:** Autorange
- **Voltage compliance:** 10 V
- **Sense mode:** 4-wire

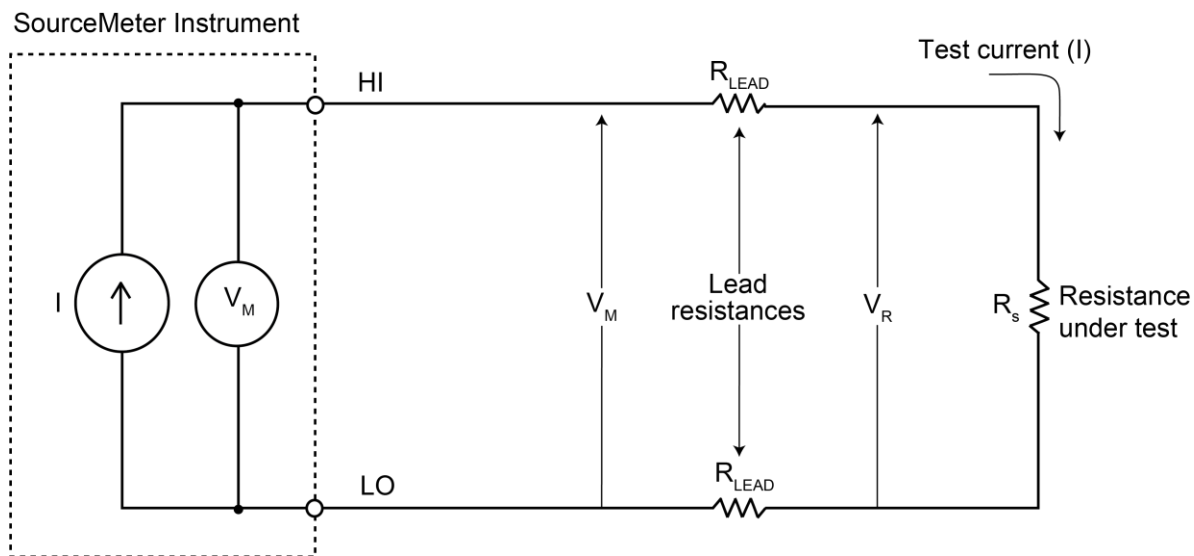
```
-- Restore 2606B defaults.
smua.reset()
-- Select the current source function.
smua.source.func = smua.OUTPUT_DCAMPS
-- Set the source range to 10 mA.
smua.source.rangei = 10e-3
-- Set the current source to 10 mA.
smua.source.leveli = 10e-3
-- Set the voltage limit to 10 V.
smua.source.limitv = 10
-- Enable 4-wire ohms.
smua.sense = smua.SENSE_REMOTE
-- Set the voltage range to auto.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Retrieve a resistance reading.
print(smua.measure.r())
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
```

Ohms sensing

Ohms measurements can be made using either 2-wire or 4-wire sensing. See [DUT test connections](#) (on page 4-3) for information on connections and sensing methods.

The 2-wire sensing method has the advantage of requiring only two test leads. However, as shown in the following figure (2-wire resistance sensing), test lead resistance can seriously affect the accuracy of 2-wire resistance measurements, particularly with lower resistance values.

Figure 43: Two-wire resistance sensing



I = Current sourced

V_M = Voltage measured

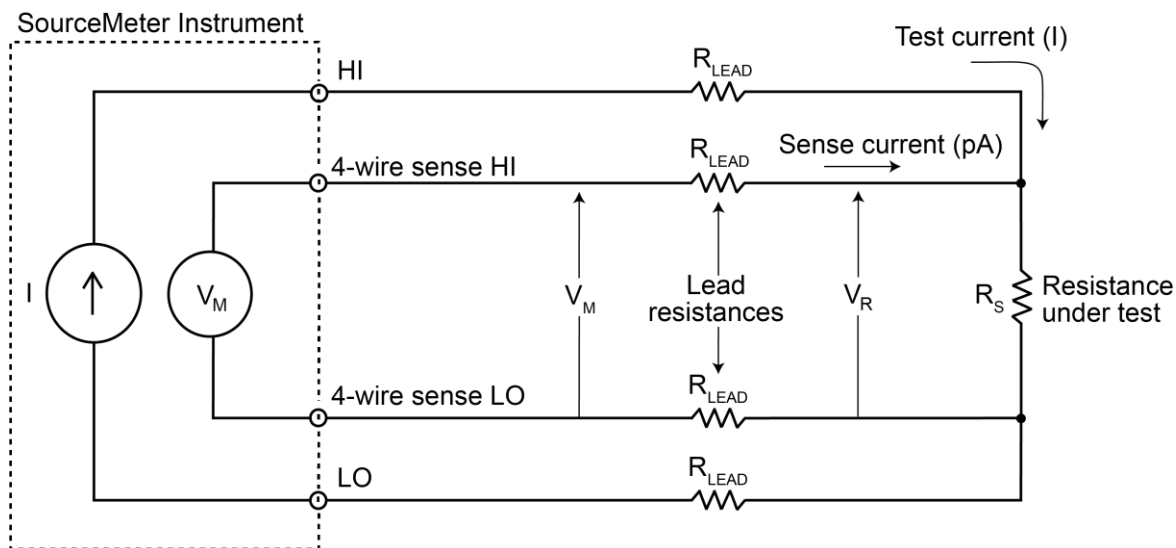
V_R = Voltage across resistor

$$\text{Measured resistance} = \frac{V_M}{I} = R_s + (2 \times R_{\text{LEAD}})$$

$$\text{Actual resistance} = \frac{V_R}{I} = R_s$$

The 4-wire sensing method, as shown in the following figure (4-wire resistance sensing), minimizes or eliminates the effects of lead resistance by measuring the voltage across the resistor under test with a second set of test leads. Because of the high input impedance of the voltmeter, the current through the sense leads is negligible, and the measured voltage is essentially the same as the voltage across the resistor under test.

Figure 44: Four-wire resistance sensing



I = Current sourced by SourceMeter
 V_M = Voltage measured by SourceMeter
 V_R = Voltage across resistor

Because sense current is negligible, $V_M = V_R$
 and measured resistance = $\frac{V_M}{I} = \frac{V_R}{I} = R_s$

Power measurements

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

Basic power measurement procedure

If you need to customize the standard wattmeter configuration of the MODE key, perform the following steps to make power measurements. The following procedure assumes that the 2606B is already connected to the device under test (DUT) as explained in [DUT test connections](#) (on page 4-3).

WARNING

Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the 2606B while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the 2606B before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.

To make power measurements from the virtual front panel:

1. Press the **DISPLAY** key to select single-channel display mode. See [Display mode](#) (on page 3-34).
2. Set source function and value. Press the **SRC** key to select the voltage or current source function as required.
3. Set the output voltage or current to an appropriate value. See Step 1 of [Front-panel source-measure procedure](#) (on page 5-2).
4. Press the **LIMIT** key and set the voltage or current limit high enough for the expected voltage or current across the DUT to be measured. See Step 2 of [Front-panel source-measure procedure](#) (on page 5-2).
5. Press the **MEAS** key as many times as needed to display power.
6. Press the **OUTPUT ON/OFF** control to turn the output on.
7. View the displayed reading (press the **TRIG** key if necessary).
8. When finished, press the **OUTPUT ON/OFF** control again to turn the output off.

Power measurements using the remote interface

The following paragraphs summarize basic power measurement commands using the remote interface and also give a programming example for a typical power measurement situation.

Remote power reading command

The programming example below illustrates how to get a power reading from SMU A:

```
reading = smua.measure.p()
```

See [Remote source-measure commands](#) (on page 5-4) for more commands necessary to set up source and measure functions and also [Introduction to TSP operation](#) (on page 13-1).

Power measurement programming example

The following programming example illustrates the setup and command sequence for a typical power measurement procedure with the following parameters:

- **Source function:** Voltage, source autorange, 5 V output
- **Current measure function and range:** Current, autorange
- **Current compliance:** 50 mA

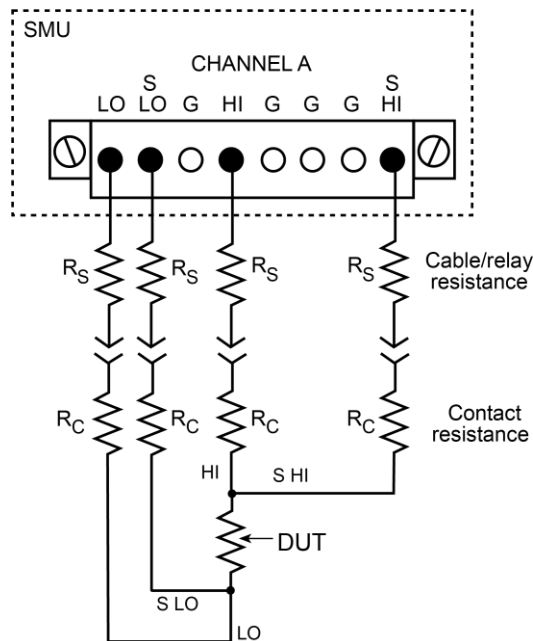
```
-- Restore 2606B defaults.
smua.reset()
-- Select voltage source function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Enable source autoranging.
smua.source.autorangev = smua.AUTORANGE_ON
-- Set voltage source to 5 V.
smua.source.levelv = 5
-- Set current limit to 50 mA.
smua.source.limiti = 50e-3
-- Set current range to autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Turn on the output.
smua.source.output = smua.OUTPUT_ON
-- Retrieve a power reading.
print(smua.measure.p())
-- Turn off the output.
smua.source.output = smua.OUTPUT_OFF
```

Contact check measurements

The contact check function prevents measurements that may be in error due to excessive resistance in the force or sense leads when making remotely sensed (Kelvin) measurements. Potential sources for this resistance include poor contact at the device under test (DUT), failing relay contacts on a switching card, and wires that are too long or thin. To use contact check, the current limit must be at least 1 mA (this allows enough current to flow when performing the test), and the source-measure unit (SMU) must not be in High-Z output-off mode.

The contact check function also detects an open circuit that may occur when a four-point probe is misplaced or misaligned. This relationship is shown schematically in the figure below, where R_C is the resistance of the mechanical contact at the DUT, and R_S is the series resistance of relays and cables.

Figure 45: Contact check measurements



When the source is off, `smuX.source.offmode` is set to `smuX.OUTPUT_ZERO`, and the effective current limit is less than 1 mA, contact check operations result in error code 5066, `source.offlimiti` too low for contact check. In the zero off mode, `smuX.source.offlimiti` is ignored and the effective current limit depends on what the channel is sourcing when it is turned off. If the channel is sourcing:

- Voltage: The current limit is determined by `smuX.source.limiti`.
- Current: The current limit is determined by the greater of `smuX.source.leveli` or 10 percent of `smuX.source.rangei`.

Contact check commands

The following table summarizes the basic contact check commands. For a more complete description of these commands, refer to the [TSP command reference](#) (on page 15-1). For connection information, refer to [Contact check connections](#) (on page 5-23).

Basic contact check commands

Command	Description
<code>flag = smuX.contact.check()</code>	Determine if contact resistance is lower than threshold.
<code>rhi, rlo = smuX.contact.r()</code>	Measure the aggregate contact resistance.
<code>smuX.contact.speed = speed_opt</code>	Set <code>speed_opt</code> to one of the following: 0 or <code>smuX.CONTACT_FAST</code> 1 or <code>smuX.CONTACT_MEDIUM</code> 2 or <code>smuX.CONTACT_SLOW</code>
<code>smuX.contact.threshold = rvalue</code>	Set resistance threshold for the contact check function.

Contact check programming example

The following programming example illustrates the setup and command sequence for a typical contact check measurement. These commands set the contact check speed to fast and the threshold to 100 Ω . Then, a contact check measurement against the threshold is made. If it fails, a more accurate contact check measurement is made, and the test is aborted. Otherwise, the output is turned on, and the test continues.

```
-- Restore defaults.
smua.reset()
-- Set contact check speed to fast.
smua.contact.speed = smua.CONTACT_FAST
-- Set the contact check threshold to 100 ohms.
smua.contact.threshold = 100
-- Check contacts against threshold.
if not smua.contact.check() then
    -- Set speed to slow.
    smua.contact.speed = smua.CONTACT_SLOW
    -- Get aggregate resistance readings.
    rhi, rlo = smua.contact.r()
    -- Return contact resistances to the host.
    print(rhi, rlo)
    -- Terminate execution.
    exit()
end
-- Turn output on and continue.
smua.source.output = smua.OUTPUT_ON
```


Multiple SMU connections

WARNING

Connecting two 2606B channels in parallel or in series may result in voltages or power levels that exceed the safety mechanisms. This increases the risk of instrument damage and the possibility of personal injury or death due to electric shock. The user assumes all of the associated risks of combining the outputs of two or more 2606B channels.

WARNING

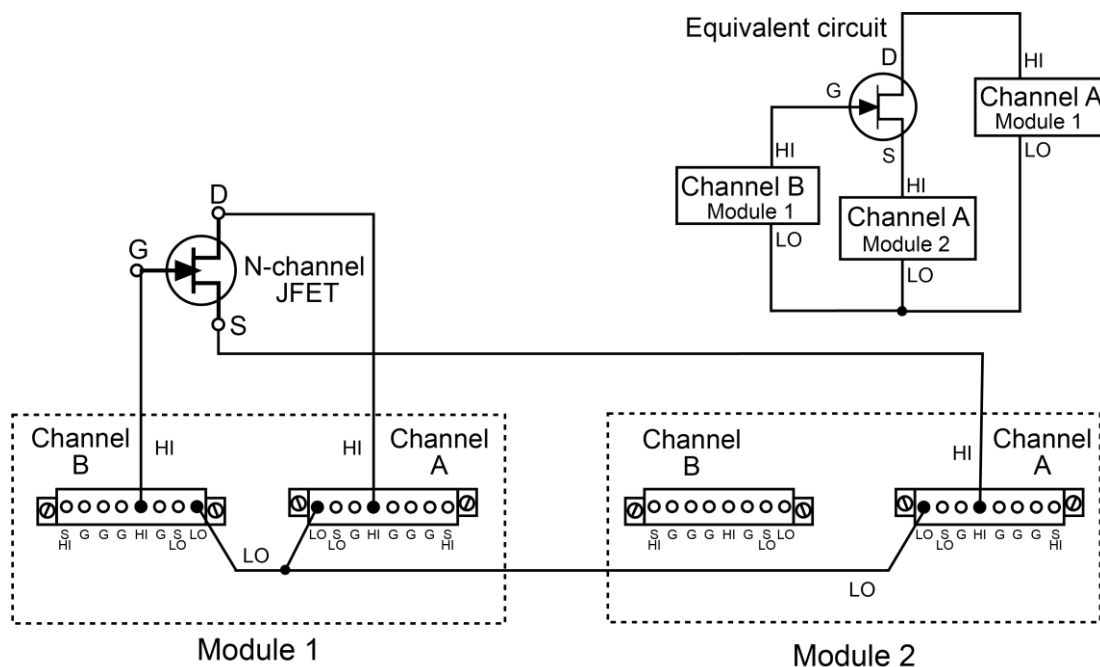
Connections to LO on the 2606B are not necessarily at 0 V. Hazardous voltages could exist between LO and chassis ground. Make sure that high-voltage precautions are taken throughout the test system. Alternatively, limit hazardous levels by adding external protection to limit the voltage between LO and chassis. Failure to make sure high-voltage precautions are used throughout the test system or a failure to limit hazardous levels could result in severe personal injury or death from electric shock.

CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the 2606B to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other 2606B instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

The following figures illustrate using three channels to test the same 3-terminal N-channel JFET device. The third channel is connected to the source (S) terminal of the JFET. This allows the source terminal to be biased above signal LO. Setting this SMU to output 0 V effectively connects the source terminal of the JFET to signal LO.

Figure 46: Three channels connected to a 3-terminal device



Output-off modes

CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the 2606B to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other 2606B instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

Turning off the 2606B output may not completely isolate the instrument from the external circuit. You can use the output-off mode to place the 2606B in a noninteractive state during idle periods. The available output-off modes are normal, high-impedance, and zero.

Normal output-off mode

The normal output-off mode is the default output-off mode setting. When the source-measure unit (SMU) is in the normal output-off mode, you can select either the current or the voltage output-off function (see [Output-off function](#) (on page 5-26)). You can also specify current and voltage output-off limits ([Output-off limits \(compliance\)](#) (on page 5-28)).

When the output is turned off, the output goes to either 0 V or 0 A, depending on the selected output-off function. Voltage is the default output-off function.

High-impedance output-off mode

For the high-impedance output-off mode (HI-Z), the output relay opens when the output is turned off. This disconnects external circuitry from the input/output of the source-measure unit (SMU). To prevent excessive wear on the output relay, do not use this output-off mode for tests that turn the output off and on frequently.

Zero output-off mode

When the zero output-off mode is selected, the programmed source remains on the display, but internally, the voltage source is selected and is set to 0 V. Measurements are made and displayed.

When the selected source is voltage, the current compliance setting remains the same as the output-on value and compliance detection remains active.

When the selected source is current, the current compliance setting is the programmed current source value or 10 percent full-scale of the present current range, whichever is greater.

You can use the 2606B as a current meter when it is in zero output-off mode because it outputs 0 V but measures current.

To configure the output-off mode from the virtual front panel:

1. Press the **CONFIG** key.
2. Press the **OUTPUT ON/OFF** control.
3. Select **OFF-STATE**.
4. Select **MODE**.
5. Select the output-off mode: **HI-Z** (high-impedance), **NORMAL**, or **ZERO**.
6. Press the **EXIT** key to return to the normal display.

*To select the normal output-off mode over a remote interface:**

```
smua.source.offmode = smua.OUTPUT_NORMAL
```

*To select the high-impedance output-off mode over a remote interface:**

```
smua.source.offmode = smua.OUTPUT_HIGH_Z
```

*To select the zero output-off mode over a remote interface:**

```
smua.source.offmode = smua.OUTPUT_ZERO
```

* smuX can be smua for channel A or smub for channel B

Output-off function

This setting is used only when the output is turned off and the 2606B is set to the normal output-off mode (`smuX.source.offmode = smuX.OUTPUT_NORMAL`).

You can set the output-off function to **CURRENT** or **VOLTAGE** through the **CONFIG** menu on the virtual front panel, or by using the `smuX.source.offfunc` attribute from a remote interface. **VOLTAGE** is the default output-off function.

When the output is turned off and the selected output-off function is **VOLTAGE** (`smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`):

- The source-measure unit (SMU) sources 0 V.
- The current limit is set by the `smuX.source.offlimiti` attribute (default 1 mA).

When the output is turned off and the selected output-off function is **CURRENT** (`smuX.source.offfunc = smuX.OUTPUT_DCAMPS`):

- The SMU sources 0 A.
- The voltage limit is set by the `smuX.source.offlimitv` attribute (default 40 V).

When the output-off function is set to either voltage or current, the SMU may source or sink a very small amount of power. In most cases, this source or sink power level is insignificant.

Selecting the output-off function

NOTE

This setting is used only when the output is turned off and the source-measure unit (SMU) is in **NORMAL** output-off mode.

To configure the output-off function from the virtual front panel:

1. Press the **CONFIG** key.
2. Press the **OUTPUT ON/OFF** control.
3. Select **OFF-STATE** and then select **FUNCTION**.
4. Select **CURRENT** or **VOLTAGE**.
5. Press the **EXIT** key to return to the normal display.

To configure the output-off function remotely:

To set 0 V output with current limit set by the `smuX.source.offlimiti` attribute:

Channel A: `smua.source.offfunc = smua.OUTPUT_DCVOLTS`

Channel B: `smub.source.offfunc = smub.OUTPUT_DCVOLTS`

To set 0 A output with voltage limit set by the `smuX.source.offlimitv` attribute:

Channel A: `smua.source.offfunc = smua.OUTPUT_DCAMPS`

Channel B: `smub.source.offfunc = smub.OUTPUT_DCAMPS`

Output-off limits (compliance)

You can set output-off limits (compliance) for the current and voltage output-off functions using the CONFIG menu on the 2606B virtual front panel, or by setting the `smuX.source.offlimitY` attribute from a remote interface. The output-off limits only apply when the output-off mode is normal.

Setting output-off limits

Setting the output-off limit for CURRENT (`smuX.source.offlimiti`) specifies the current limit for the voltage source; setting the output-off limit for VOLTAGE (`smuX.source.offlimitv`) specifies the voltage limit for the current source.

To configure the output-off limits from the virtual front panel:

1. Press the **CONFIG** key.
2. Press the **OUTPUT ON/OFF** control.
3. Select **OFF-STATE** and then select **LIMIT**.
4. Select **CURRENT** or **VOLTAGE**.
5. Set the limit value and then press the **ENTER** key or the navigation wheel.
6. Press the **EXIT** key to return to the normal display.

To set the current limit in NORMAL output-off mode remotely:*

```
smuX.source.offlimiti = iValue
```

To set the voltage limit in NORMAL output-off mode remotely:*

```
smuX.source.offlimitv = vValue
```

* `smuX` can be `smua` for channel A or `smub` for channel B

Remote programming output-off states quick reference

The content of the following table is a quick reference of commands for programming output-off states from a remote interface.

Output-off state programming quick reference

Command*	Description
<code>smuX.source.offmode = smua.OUTPUT_NORMAL</code>	Selects normal output-off mode.
<code>smuX.source.offmode = smua.OUTPUT_HIGH_Z</code>	Selects high-impedance output-off mode.
<code>smuX.source.offmode = smua.OUTPUT_ZERO</code>	Selects zero output-off mode.
<code>smuX.source.offfunc = smua.OUTPUT_DCVOLTS</code>	Sets 0 V output with current limit specified by the <code>smua.source.offlimiti</code> attribute.
<code>smuX.source.offfunc = smua.OUTPUT_DCAMPS</code>	Sets 0 A output with voltage limit specified by the <code>smua.source.offlimitv</code> attribute.
<code>smuX.source.offlimiti = iValue</code>	Sets current limit in normal output-off mode.
<code>smuX.source.offlimitv = vValue</code>	Sets voltage limit in normal output-off mode.

* `smuX` can be `smua` for channel A or `smub` for channel B

Range

The selected measurement range affects the accuracy of the measurements and the maximum signal that can be measured. If the range is changed, the virtual front-panel display may contain dashes instead of a reading (for example, `-- . ---- mA`). This indicates that no measurement was made using the range that is presently selected. To update the displayed reading, trigger a measurement (if in local control, press the **TRIG** key).

Refer to the specifications for available ranges.

Maximum source values and readings

The full-scale output for each voltage and current source range is 101 percent of the selected range, but the full-scale measurement is 102 percent of the range. For example, ± 1.01 A is the full-scale source value for the 1 A range, and ± 102 mA is the full-scale reading for the 100 mA measurement range. Input levels that exceed the maximum levels cause the overflow message to be displayed. The instrument autoranges at 100 percent of the range.

Measure autodelay

The measure delay is a specific delay that is applied before each measurement is made. This delay is disabled by default (measurements are made immediately). You can change the default delay by setting the [smuX.measure.delay](#) (on page 15-216) attribute either to a specific value or to an autodelay setting (set `smuX.measure.delay = smuX.DELAY_AUTO`). If the measure delay is set to the autodelay setting, a range-dependent delay is applied each time the instrument makes a current measurement. This delay also happens for the measurement that is made after changing current ranges during an autoranged measurement.

You can increase or decrease the autodelay by changing the delay factor (for example, to reduce the delay across all ranges by half, set `smuX.measure.delayfactor = 0.5`). For additional information, refer to [smuX.measure.delayfactor](#) (on page 15-217).

Ranging limitations

If the source and measure functions are different (such as source V and measure I, or source I and measure V), you can set source and measure ranges separately. If both source and measure functions are the same, the measure range is locked to the source range.

With the 20 V range selected for the voltage source, the highest current measurement range is 1 A. With the 3 A range selected for the current source, the highest voltage measurement range is 6 V. Refer to [Operating boundaries](#) (on page 8-3) for power derating information.

Manual ranging

To select a range from the virtual front panel, use the range keys,  and  :

- To set the source range, press the **SRC** key, and then use the **RANGE** keys to set the range.
- To set the measure range, select the single-channel display mode, and then use the **RANGE** keys to set the range.

If the instrument displays the overflow message on a particular range, select a higher range until an on-range reading is displayed. To ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow.

Autorangeing

To use automatic source ranging from the virtual front panel, press **SRC** then the **AUTO** range key.

To use automatic measure ranging, press the **MEAS** key followed by the **AUTO** range key. The **AUTO** indicator turns on when source or measure autoranging is selected.

When autorange is selected, the instrument automatically sets the best range to source or measure the applied signal. The instrument increases the range to 100 percent of the present range.

NOTE

When you change a source value, source autoranging is automatically turned off and remains off until you re-enable it.

Low range limits

The low range limit sets the lowest range that the 2606B uses when autoranging is enabled. This feature is useful for minimizing autorange settling times when measurements require numerous range changes.

To individually set low range limits for Source V, Source I, Measure V, and Measure I from the virtual front panel:

1. Press the **CONFIG** key, then press either the **SRC** key (for source) or the **MEAS** key (for measure).
2. Select voltage or current source, or measure, as appropriate, and then press the **ENTER** key or the navigation wheel.
3. Select **LOWRANGE**, and then press the **ENTER** key or the navigation wheel.
4. Set the low range to the appropriate setting, and then press the **ENTER** key or the navigation wheel.
5. Press the **EXIT (LOCAL)** key twice to return to the main display.

Range considerations

The source range and measure range settings can interact depending on the source function. Additionally, the output state (on or off) can affect how the range is set. The following table describes these interactions.

If...	Then...	Notes
The source function is the same as the measurement function (for example, sourcing voltage and measuring voltage)	The measurement range is locked to be the same as the source range.	The setting for the voltage measure range is retained and used when the source function is changed to current. 2606B example: <pre>smua.source.func = smua.OUTPUT_DCVOLTS smua.source.rangev = 1 smua.measure.rangev = 10 -- Prints 1, the source range print(smua.measure.rangev) smua.source.func = smua.OUTPUT_DCAMPS -- Prints 10, the measure range print(smua.measure.rangev)</pre>
A source or measurement range for a function is explicitly set	Autoranging for that function is disabled.	Autoranging is controlled separately for each source and measurement function: Source voltage, source current, measure voltage, and measure current. Autoranging is enabled for all measurement functions by default.
Source autoranging is enabled	The output level controls the range.	Querying the range after the level is set returns the range the instrument chose as appropriate.
You send a source level that is out of range while autorange is off	The instrument does not return an error until the output is turned on.	While the output is turned off, the display shows a series of question marks. For example, ??? . ???mA is displayed for the 100 mA range.
Measure autoranging is enabled	The measure range changes only when a measurement is made.	Querying the range after the measurement is made returns the range that the instrument chose.

Range commands

The following tables summarize commands necessary to control measure and source ranges. See the [TSP command reference](#) (on page 15-1) for more details about these commands.

Measure range commands

Command*	Description
<code>smuX.measure.autorangei = smuX.AUTORANGE_ON</code>	Enable current measure autorange.
<code>smuX.measure.autorangei = smuX.AUTORANGE_OFF</code>	Disable current measure autorange.
<code>smuX.measure.autorangev = smuX.AUTORANGE_ON</code>	Enable voltage measure autorange.
<code>smuX.measure.autorangev = smuX.AUTORANGE_OFF</code>	Disable voltage measure autorange.
<code>smuX.measure.lowrangei = lowrange</code>	Set lowest I measure range for autorange.
<code>smuX.measure.lowrangev = lowrange</code>	Set lowest V measure range for autorange.
<code>smuX.measure.rangei = rangeval</code>	Select manual current measure range.
<code>smuX.measure.rangev = rangeval</code>	Select manual voltage measure range.

* smuX can be smua for channel A or smub for channel B

Source range and limit commands

Command*	Description
<code>smuX.source.autorangei = smuX.AUTORANGE_ON</code>	Enable current source autorange.
<code>smuX.source.autorangei = smuX.AUTORANGE_OFF</code>	Disable current source autorange.
<code>smuX.source.autorangev = smuX.AUTORANGE_ON</code>	Enable voltage source autorange.
<code>smuX.source.autorangev = smuX.AUTORANGE_OFF</code>	Disable voltage source autorange.
<code>smuX.source.limiti = level</code>	Set voltage source current limit (compliance).
<code>smuX.source.limitv = level</code>	Set current source voltage limit (compliance).
<code>smuX.source.limitp = level</code>	Set source power limit (compliance).
<code>smuX.source.lowrangei = lowrange</code>	Set lowest I source range for autorange.
<code>smuX.source.lowrangev = lowrange</code>	Set lowest V source range for autorange.
<code>smuX.source.rangei = rangeval</code>	Select manual current source range.
<code>smuX.source.rangev = rangeval</code>	Select manual voltage source range.

* `smuX` can be `smua` for channel A or `smub` for channel B

Range programming example

The programming example below illustrates how to control both source and measure ranges. The 2606B is set up as follows:

- Voltage source range: Autorange
- Current measure range: 10 mA
- Voltage source current limit: 10 mA

```
-- Restore 2606B defaults.
smua.reset()
-- Set V source range to autorange.
smua.source.autorangev = smua.AUTORANGE_ON
-- Select 10 mA measure range.
smua.measure.rangei = 10e-3
-- Set limit level to 10 mA.
smua.source.limiti = 10e-3
```

Digits

The virtual front-panel display resolution of the measured reading depends on the DIGITS setting. The default display resolution setting is 6.5 digits. The DIGITS setting selects display resolution for all measurement functions.

The DIGITS setting has no effect on the format of readings returned by a `print()` command over a remote interface. To adjust the format of remote interface readings, see [format.asciiprecision](#) (on page 15-100).

The number of displayed digits does not affect accuracy or speed. Accuracy and speed are controlled by the SPEED setting (see [Speed](#) (on page 5-33)).

Setting display resolution from the virtual front panel

To set the display resolution, press the **DIGITS** key until the correct number of digits is displayed. Available display resolutions are 4.5, 5.5, and 6.5 digits.

Setting display resolution from a remote interface

The following table summarizes use of the `display.smuX.digits` command. See the [TSP command reference](#) (on page 15-1) for more information.

Digits commands

Command*	Description
<code>display.smuX.digits = display.DIGITS_4_5</code>	Set the display to 4.5 digits.
<code>display.smuX.digits = display.DIGITS_5_5</code>	Set the display to 5.5 digits.
<code>display.smuX.digits = display.DIGITS_6_5</code>	Set the display to 6.5 digits.

* `smuX` can be `smua` for channel A or `smub` for channel B

Digits programming example

```
-- Select 5.5 digits.  
display.smua.digits = display.DIGITS_5_5
```

Speed

The **SPEED** key on the virtual front panel sets the integration time, or measurement aperture, of the analog to digital (A/D) converter (period the input signal is measured). The integration time affects the usable digits, the amount of reading noise, and the reading rate of the instrument. The integration time is specified in parameters based on the number of power line cycles (NPLC), where 1 PLC for 60 Hz is 16.67 ms (1/60) and 1 PLC for 50 Hz is 20 ms (1/50).

In general, the fastest integration time (0.001 PLC) results in the fastest reading rate, but also causes increased reading noise and fewer usable digits. The slowest integration time (25 PLC) provides the best common-mode and normal-mode noise rejection but has the slowest reading rate. Settings between the fastest and slowest integration times are a compromise between speed and noise. The default power-on speed setting is NORMAL (1 PLC).

NOTE

The **SPEED** setting affects all measurement functions. After setting speed, display resolution can be changed using the **DIGITS** key.

Setting the speed from the virtual front panel

Press the **SPEED** key (or use the CONFIG menu) to display the following menu items:

- **FAST:** Sets the measurement speed to 0.01 PLC (fast performance, but accuracy is reduced).
- **MED:** Sets the measurement speed to 0.10 PLC (speed and accuracy are balanced).
- **NORMAL:** Sets the measurement speed to 1.00 PLC (speed and accuracy are balanced).
- **HI-ACCURACY:** Sets the measurement speed to 10.00 PLC (high accuracy, but speed is reduced).
- **OTHER:** Sets the measurement speed to any value from 0.001 PLC to 25 PLC.

Setting the speed using the remote interface

The following table shows the command that controls speed. See the [TSP command reference](#) (on page 15-1) for more information.

Speed command

Command*	Description
<code>smuX.measure.nplc = nplc</code>	Sets the speed of the A/D converter (nplc = 0.001 to 25).

* `smuX` can be `smua` for channel A or `smub` for channel B

Speed programming example

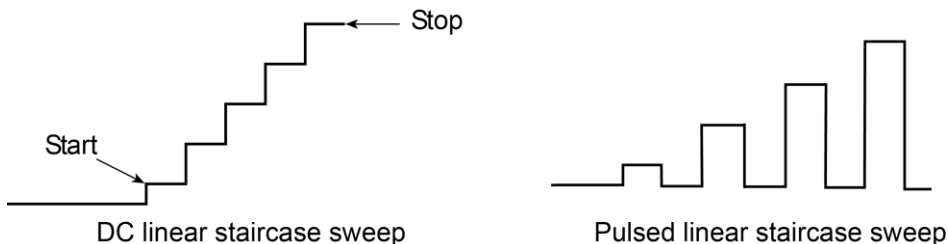
Use the NPLC command to set the speed of the integrating analog-to-digital converter (ADC). The programming example below illustrates how to set the speed to 10 PLC:

```
-- Set NPLC to 10.
smua.measure.nplc = 10
```

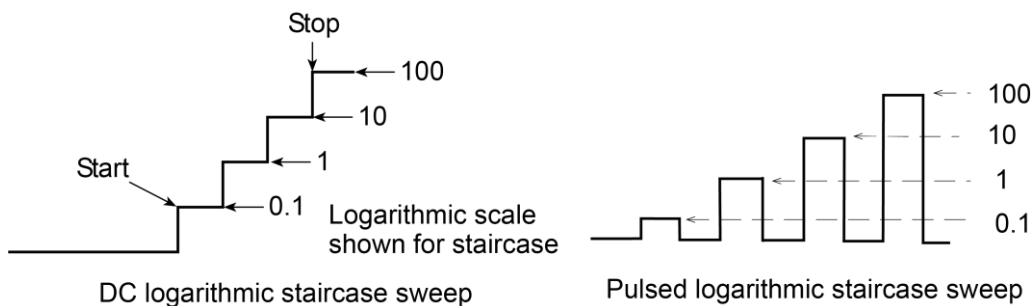
Sweep operation

The 2606B can generate dc and pulsed sweeps to perform source-only sweeps, source-and-measure sweeps, or measure-only sweeps. The following information describes the sweep types of dc and pulsed linear staircase, dc and pulsed logarithmic staircase, and dc and pulsed list.

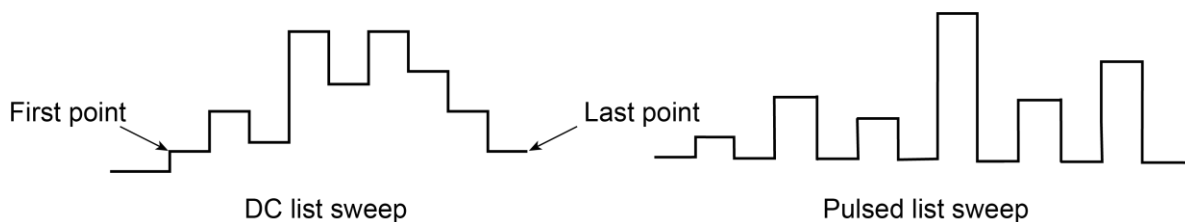
DC and pulsed linear staircase sweeps: With this type of sweep, the voltage or current increases or decreases in fixed steps, beginning with a start voltage or current and ending with a stop voltage or current. The figure below shows an increasing linear staircase sweep and a pulsed staircase sweep. Pulsed linear staircase sweeps function the same way that dc linear staircase sweeps function, except that pulsed linear staircase sweeps return to the idle level between pulses.

Figure 47: DC and pulsed linear staircase sweeps

DC and pulsed logarithmic staircase sweeps: In this type of sweep, the current or voltage increases or decreases 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 and a pulsed logarithmic staircase sweep. Pulsed logarithmic staircase sweeps function the same way that dc logarithmic staircase sweeps function, except that pulsed logarithmic staircase sweeps return to the idle level between pulses.

Figure 48: DC and pulsed logarithmic staircase sweeps

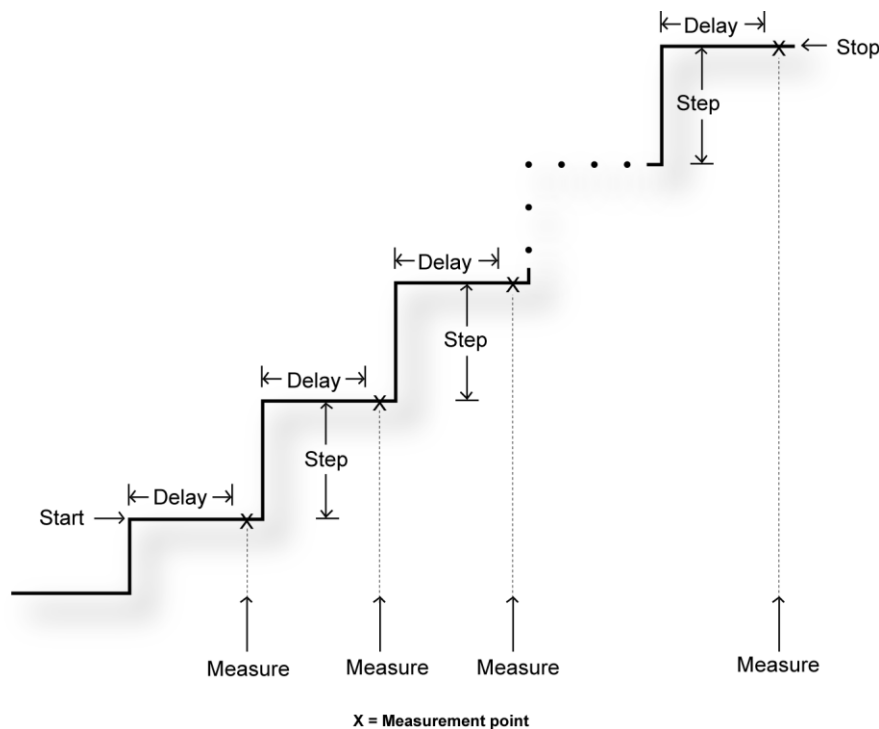
DC and pulsed list sweeps: The list sweep allows you to program arbitrary sweep steps anywhere within the output voltage or current range of the 2606B. The following figure shows a list sweep with arbitrary steps and a pulsed list sweep. Pulsed list sweeps function the same way that dc list sweeps function, except that pulsed list sweeps return to the idle level between pulses.

Figure 49: DC and pulsed list sweeps

Linear staircase sweeps

As shown below, this sweep type steps from a start voltage or current value to an ending (stop) value. When enabled, a measurement is made at each point after the source and measurement settling time.

Figure 50: Linear staircase sweep



A linear staircase sweep is configured using a start level, a stop level, and the total number of points, including the start and stop points. The step size is determined by the start and stop levels, and the number of sweep points:

$$\text{step} = (\text{stop} - \text{start}) / (\text{points} - 1)$$

NOTE

The number of sweep steps actually performed is determined by the trigger count. Refer to [Triggering](#) (on page 7-1) for more information.

The sweep can be either positive-going or negative-going, depending on the relative values of the start and stop parameters. When the sweep starts, the output goes to the start source level. The output then changes in equal steps until the stop level is reached. If the trigger count is greater than the number of points specified, the SMU starts over at the beginning value.

To configure a linear staircase sweep, use the `smuX.trigger.source.linearY()` command. This function configures the source values the SMU outputs when performing a linear sweep. After configuring the sweep, you must also enable the source action by setting the following attribute:*

```
smuX.trigger.source.action
```

* `smuX` can be `smua` for channel A or `smub` for channel B

Example:

```
-- Configure a sweep from 0 to 10 V in 1 V steps.
smua.trigger.source.linearv(0, 10, 11)
-- Enable the source action.
smua.trigger.source.action = smua.ENABLE
```

For more information, see [smuX.trigger.source.linearY\(\)](#) (on page 15-265).

Logarithmic staircase sweeps

This type of sweep is similar to the linear staircase sweep. The steps, however, are done on a logarithmic scale.

Like a linear staircase sweep, logarithmic sweeps are configured using a start level, a stop level, and the number of points. The step size is determined by the start and stop levels, and the number of sweep points. However, in a logarithmic sweep, the step size increases or decreases exponentially. To create an increasing logarithmic sweep, set the stop value to be greater than the start value. To create a decreasing logarithmic sweep, set the stop value to be less than the start value. When enabled, a measurement is made at each step after source and measurement settling time. An asymptote can also be used to control the inflection of a sweep.

NOTE

The number of sweep steps actually performed is determined by the trigger count. See [Triggering](#) (on page 7-1) for more information.

The formula for a logarithmic sweep is:

$$v_i = A + kb^i$$

Where:

- v_i = The source value at source point i
- i = The index of points in the sweep (ranges from 0 to $N - 1$), where N is the number of points in the sweep
- k = The initial source value as an offset from the asymptote
- b = The step size ratio
- A = The asymptote value

The asymptote is used to change the inflection of the sweep curve and allow it to sweep through zero. The following figures depict the effect of the asymptote on the inflection of the sweep curve.

Figure 51: Increasing logarithmic sweep

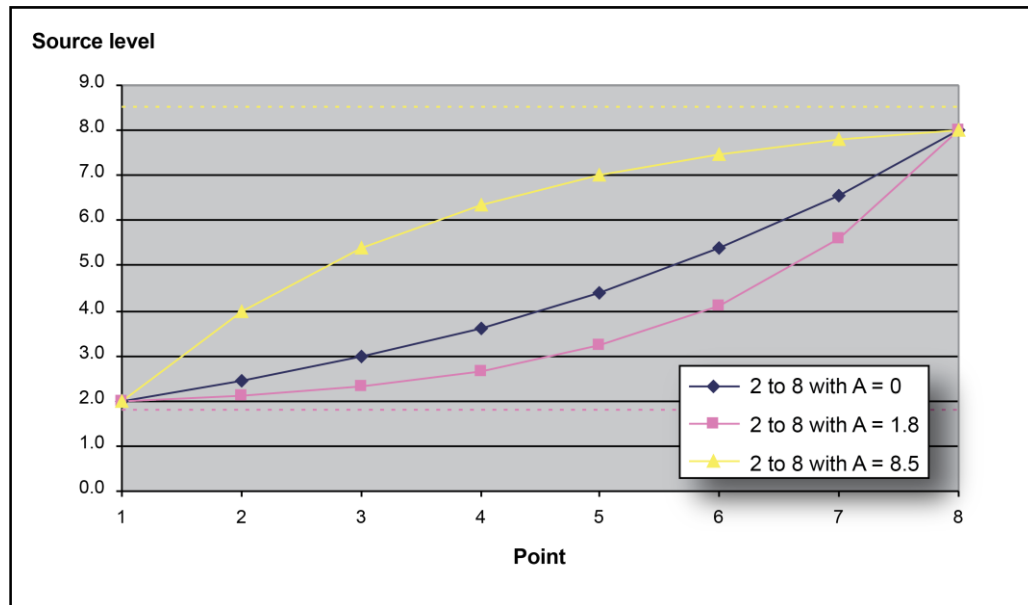
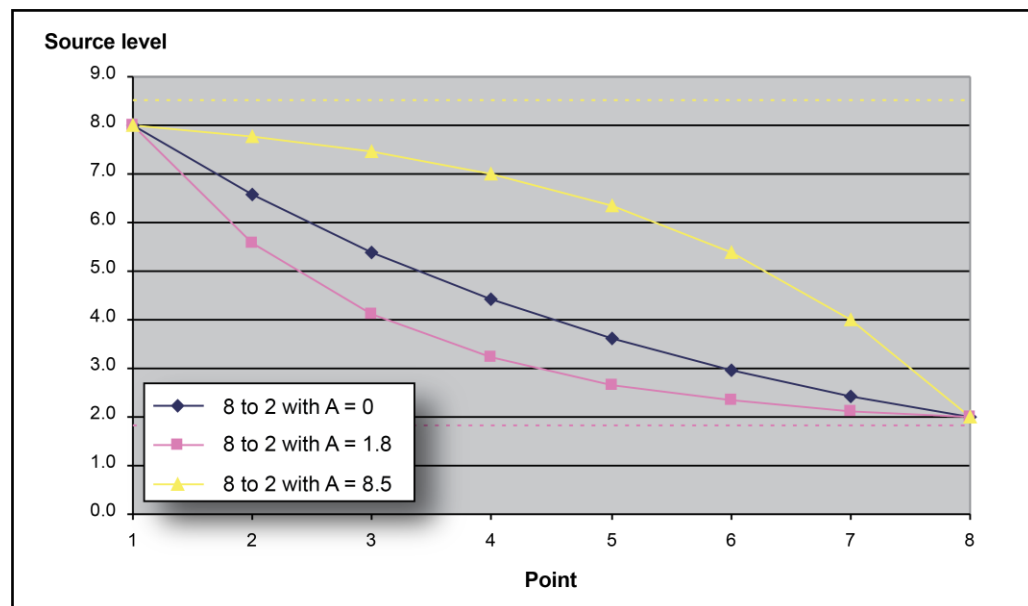


Figure 52: Decreasing logarithmic sweep



Solving for k and b provides the following formulas:

$$k = V_{\text{start}} - A$$

$$b = 10^{\left(\frac{\log_{10}(V_{\text{end}} - A) - \log_{10}(V_{\text{start}} - A)}{N - 1} \right)}$$

Where:

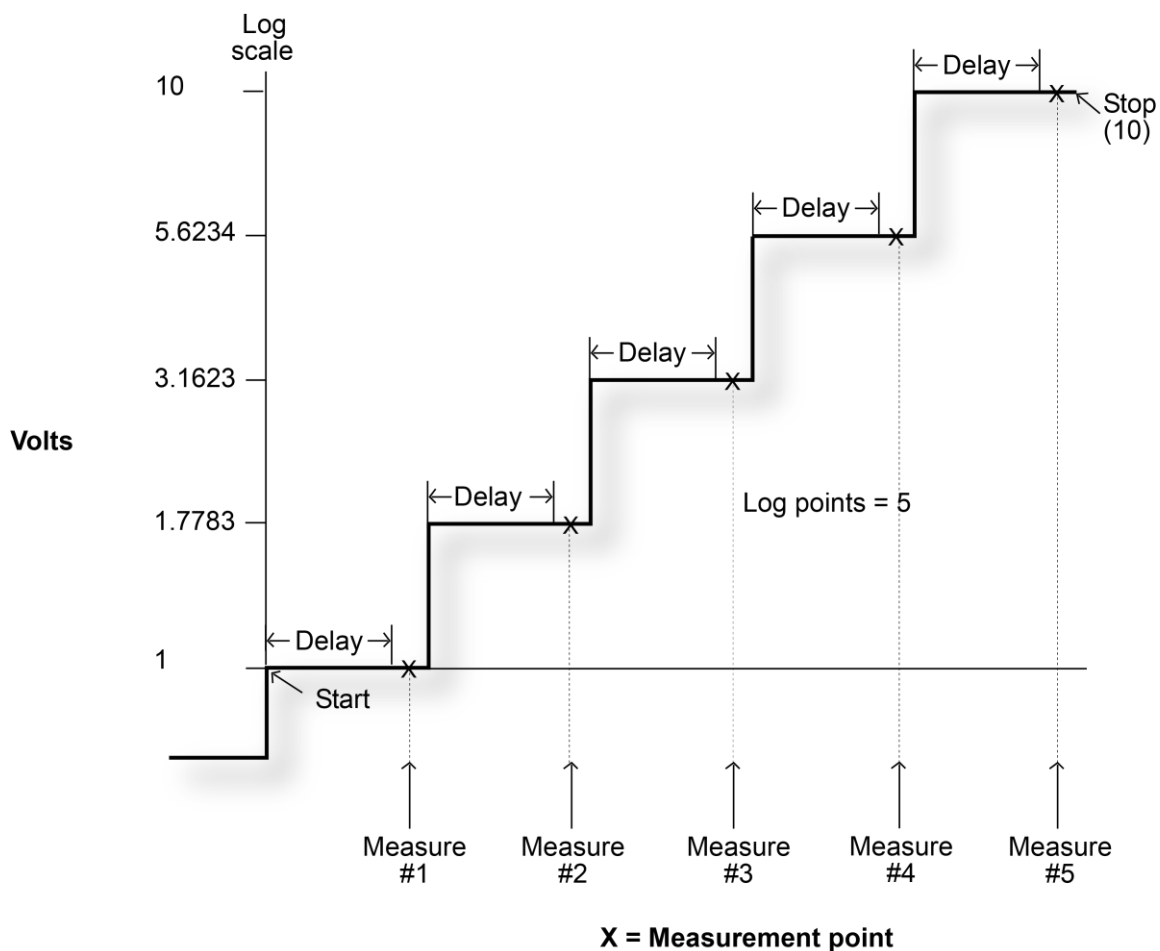
V_{end} = The source value at the end point
 V_{start} = The source value at the start point
 N = The number of points in the sweep
 A = The asymptote value

NOTE

The number of points in a sweep is one greater than the number of steps in the sweep.

The following figure is an example of a five-point logarithmic sweep from 1 V to 10 V.

Figure 53: Logarithmic staircase sweep (1 V to 10 V, five steps)



In this example:

$$A = 0, V_{\text{start}} = 1, V_{\text{end}} = 10, N = 5$$

Using the formula above, $k = 1$

Step size (b) for the sweep in the above figure is calculated as follows:

$$\begin{aligned} \text{Log step size} &= \frac{\log_{10}(\text{stop}) - \log_{10}(\text{start})}{\text{Points} - 1} \\ &= \frac{\log_{10}(10) - \log_{10}(1)}{5 - 1} \\ &= \frac{1 - 0}{4} \\ &= 0.25 \end{aligned}$$

Therefore, $b = 10^{(\log \text{ step size})} = 1.7783$

The log steps for this sweep are listed in the table below.

Logarithmic sweep points

Source point (N)	Source level (V)	Step number (i)
1	1	0
2	1.7783	1
3	3.1623	2
4	5.6234	3
5	10	4

When this sweep starts, the output goes to the start level (1 V) and sweeps through the symmetrical log points.

To configure a logarithmic staircase sweep, use the `smuX.trigger.source.logY()` function. This function configures the source values the source-measure unit (SMU) outputs when performing a logarithmic sweep. After configuring the sweep, you must also enable the source action by setting the `smuX.trigger.source.action` attribute.

Example:

```
-- Configure a sweep from 1 to 10 V in 10 steps with an asymptote of 0 V.
smua.trigger.source.logv(1, 10, 11, 0)
-- Enable the source action.
smua.trigger.source.action = smua.ENABLE
```

For more information, see [smuX.trigger.source.logY\(\)](#) (on page 15-267).

List sweeps

Use a list sweep to configure a sweep with arbitrary steps. When enabled, a measurement is made at each point after source and measurement settling time.

To configure a list sweep, use the `smuX.trigger.source.listY()` function. This function configures the source values that the source-measure unit (SMU) outputs when performing a list sweep. After configuring the sweep, you must also enable the source action by setting the `smuX.trigger.source.action` attribute.

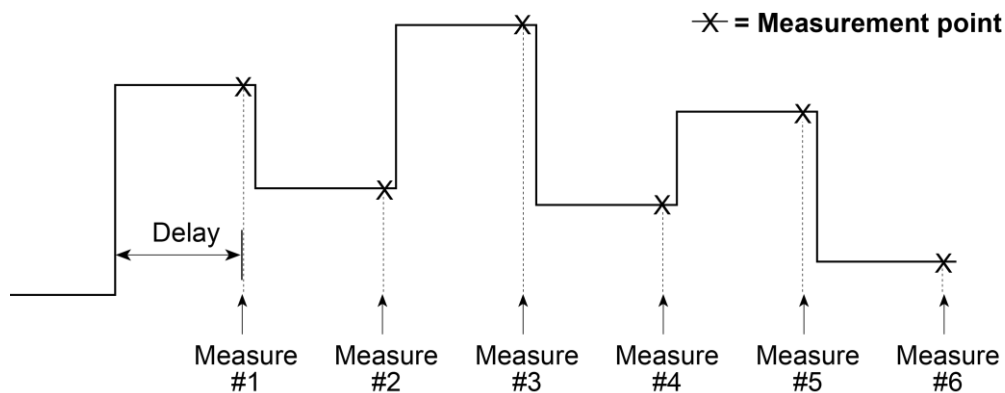
Example:

```
-- Sweep through 3 V, 1 V, 4 V, 5 V, and 2 V.  
smua.trigger.source.listv({3, 1, 4, 5, 2})  
-- Enable the source action.  
smua.trigger.source.action = smua.ENABLE
```

When the sweep is started, the output level goes to the first point in the sweep. The sweep continues through the steps in the order that they were programmed.

The following figure shows a different example of a list sweep with six measurement points. When the sweep starts, the current or voltage goes to the first point in the sweep. The instrument cycles through the sweep points in the programmed order.

Figure 54: List sweep example



Pulse mode sweeps

To create a pulse sweep for any of the sweep types, configure the end pulse action.

To configure a pulse sweep for source-measure unit (SMU) A, send:

```
smua.trigger.endpulse.action = smua.SOURCE_IDLE
```

To configure a dc sweep for SMU A, send:

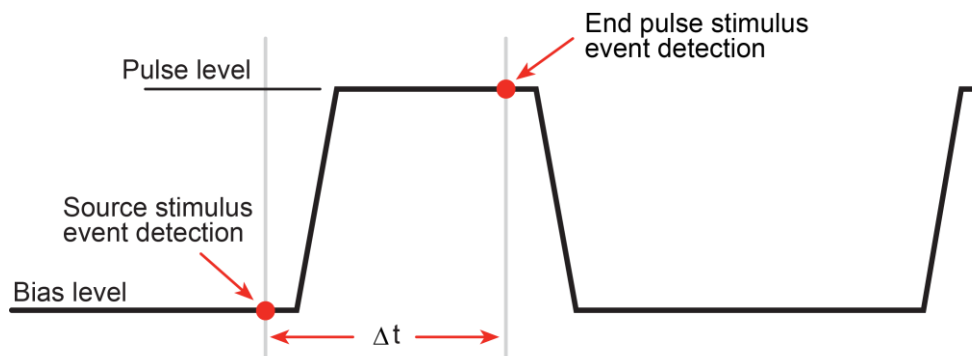
```
smua.trigger.endpulse.action = smua.SOURCE_HOLD
```

Timers must be used to configure the pulse width and period. Refer to [Using timers to perform pulsed sweeps](#) (on page 7-15) for details.

The pulse width is managed by controlling the duration between the source stimulus event and the end pulse stimulus event. A latency exists between these stimulus events and their resulting source level transitions. This trigger latency can vary based on factors such as the source range and the electrical characteristics of the device under test (DUT).

The figure below shows the source and end pulse stimulus events in relationship to the pulse (see [Triggering](#) (on page 7-1) for information on stimulus events). Any change in Δt results in a corresponding change in the pulse width.

Figure 55: Pulse width control



Pulse duty cycle

Duty cycle is the percentage of time during the pulse period that the output is on. It is calculated as follows:

$$\text{Duty cycle} = \text{Pulse width} / (\text{Pulse width} + \text{Off time})$$

For example, if the pulse width is 10 ms and the off time is 90 ms, the duty cycle is calculated as follows:

$$\begin{aligned}\text{Duty cycle} &= 10 \text{ ms} / (10 \text{ ms} + 90 \text{ ms}) \\ &= 10 \text{ ms} / 100 \text{ ms} \\ &= 0.10 \\ &= 10 \text{ percent}\end{aligned}$$

See [Maximum duty cycle equation](#) (on page 8-2) for additional information on calculating the maximum duty cycle for a SMU.

Configuring and running sweeps

Use the following topics to configure and run a sweep.

Configuring compliance limits remotely

You can configure voltage and current limits using the `smuX.trigger.source.limitY` attribute, which sets the sweep source limits. For example, to set the SMU A sweep limit to 10 V, send the command:

```
smua.trigger.source.limitv = 10
```

Configuring end sweep actions remotely

Use the end sweep action to configure the source action at the end of the sweep. The source-measure unit (SMU) can be programmed to return to the idle source level or hold the last value of the sweep. Configure the end sweep action by setting the `smuX.trigger.endsweep.action` attribute. For example, execute the following command to configure SMU A to return the source to the idle source level at the end of a sweep:

```
smua.trigger.endsweep.action = smua.SOURCE_IDLE
```

Configuring measurements during a sweep

You can make measurements during a sweep using the `smuX.trigger.measure.Y()` function. When sweeps are run, measurements are stored in the specified reading buffer for later recall. You can specify which reading buffer stores the readings. For example, to store the voltage readings made during the sweep, send the commands:

```
smua.trigger.measure.v(vbuffername)
smua.trigger.measure.action = smua.ENABLE
```

To recall sweep data using the virtual front panel:

1. Press the **RECALL** key.
2. Select **DATA** or **STATISTICS**.
3. **If you selected DATA:** Select the buffer and use the navigation wheel or cursor keys to choose reading numbers to display.
4. **If you selected STATISTICS:** Select the buffer and use the navigation wheel or cursor keys to choose **MEAN**, **STD DEV**, **SAMPLE SIZE**, **MINIMUM**, **MAXIMUM**, or **PK-PK**.

NOTE

Recalling readings from the reading buffer using the virtual front panel can be done only if one of the dedicated reading buffers is used to store the sweep data.

To recall sweep data using remote commands:

Use the `printbuffer()` function to request buffer readings.

See [Reading buffers](#) (on page 6-1) for details about recalling data from the buffer.

Source and measurement delays

Whenever the source-measure unit (SMU) outputs a source value in a sweep, it also applies the programmed source delay. The default source delay is zero (0) seconds. Set an additional source delay using the `smuX.source.delay` attribute.

Whenever the SMU makes a measurement in a sweep, it also applies any configured measurement delays. Use the `smuX.measure.delay` attribute to program a specific measurement delay. The default measurement delay varies by measure range.

Initiating and running sweeps

To run a sweep, you must configure the number of sweep points to output and the number of sweeps to perform. See [Triggering](#) (on page 7-1) for more information.

Examples:

To start a sweep, use the `smuX.trigger.initiate()` function. Sweeps are overlapped operations, so you can use the `waitcomplete()` function to suspend further operation until the sweep is complete.

To sweep 15 source points:

```
smua.trigger.count = 15
```

To perform eight sweeps:

```
smua.trigger.arm.count = 8
```

Aborting a sweep

You can use the `smuX.abort()` function to terminate all overlapped operations on a source-measure unit (SMU), including sweeps. It returns the SMU to the idle state of the remote trigger model. See [Triggering](#) (on page 7-1) for more information.

Sweeping using factory scripts

Factory script functions that perform linear staircase, logarithmic staircase, and list sweeps are described in [Factory scripts](#) (on page 13-19). You can use the factory script functions to execute simple sweeps or use them as examples for programming your own custom sweeps.

To run a sweep from the virtual front panel:

1. Press the **LOAD** key, and then select **FACTORY**.
2. Select the name of the test to run.
3. Press the **RUN** key. Follow the display prompts to complete the test.

See [Factory scripts](#) (on page 13-19) for more information about using factory scripts.

Press the **RECALL** key to access sweep data stored in dedicated reading buffer 1. See [Reading buffers](#) (on page 6-1) for more details about the buffer.

Sweep programming examples

The following topics provide procedures for programming and running a sweep. Each of these procedures includes commands for a typical sweep example. The following table summarizes parameters for each of these examples.

NOTE

You can retrieve the source code for the factory scripts by using the [scriptVar.list\(\)](#) (on page 15-185) or [scriptVar.source](#) (on page 15-188) commands.

Sweep example parameters

Sweep type	Parameters for sweep examples
Linear staircase sweep	Start current: 1 mA Stop current: 10 mA Settling time: 0.1 s Number of points: 10
Pulse current sweep	Bias current: 1 mA On current: 10 mA Pulse on time: 10 ms Pulse off time: 50 ms Number of points: 10
List sweep	Points: 3 V, 1 V, 4 V, 5 V, 2 V Settling time 0.1 s Number of points: 5

Linear staircase sweep example

The programming example below illustrates a staircase sweep.

<pre>-- Restore 2606B defaults. smua.reset() -- Set compliance to 1 V. smua.source.limitv = 1</pre>	1. Configure source functions. Restores defaults and sets the compliance limit to 1 V.
<pre>-- Linear staircase sweep -- 1 mA to 10 mA, 0.1 second delay, -- 10 points. SweepILinMeasureV(smua, 1e-3, 10e-3, 0.1, 10)</pre>	2. Configure and execute the sweep. Configures a linear staircase current sweep from 1 mA to 10 mA with 10 points and a 0.1 second settling time.
<pre>printbuffer(1, 10, smua.nvbuffer1.readings)</pre>	3. Request readings. Requests readings from buffer 1.

Pulse current sweep example

The programming example below illustrates a pulse sweep.

<pre>-- Restore 2606B defaults. smua.reset() -- Set compliance to 10 V. smua.source.limitv = 10</pre>	1. Configure source functions. Restores defaults and sets the compliance to 10 V.
<pre>-- Pulse current sweep, 1 mA bias, -- 10 mA level, 10 ms pulse on, -- 50 ms pulse off, 10 cycles. PulseIMeasureV(smua, 1e-3, 10e-3, 20e-3, 50e-3, 10)</pre>	2. Configure and execute the sweep. Configures a 10 mA pulse current sweep with a 10 ms pulse on time, a 50 ms pulse off time, and 10 pulse-measure cycles.
<pre>printbuffer(1, 10, smua.nvbuffer1.readings)</pre>	3. Request readings. Requests readings from buffer 1.

List sweep example

The programming example below illustrates a list sweep.

<pre>-- Restore 2606B defaults. smua.reset() -- Set compliance to 10 mA. smua.source.limitsi = 10e-3</pre>	1. Configure source functions. Restores defaults and set the compliance to 10 mA.
<pre>-- Define voltage list. vlist = {3, 1, 4, 5, 2} -- List sweep, channel A, 3 V, 1 V, 4 V, -- 5 V, 2 V steps, 0.1 s delay, 5 points. SweepVListMeasureI(smua, vlist, 0.1, 5)</pre>	2. Configure and execute the sweep. Configures a list sweep with 3 V, 1 V, 4 V, 5 V, and 2 V points using a 0.1 second settling time.
<pre>printbuffer(1, 5, smua.nvbuffer1.readings)</pre>	3. Request readings. Requests readings from buffer 1.

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:

$$\text{Displayed value} = \text{Actual measured value} - \text{Relative offset value}$$

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 0.5 A is set as a relative offset value on the 1 A range, the relative offset value is also 0.5 A on the lower current ranges. Selecting a range that cannot accommodate the relative offset value does not cause an overflow condition, but it also does not increase the maximum allowable input for that range. For example, on the 1 A range, the 2606B still overflows for an input of more than 1.02 A.

When relative offset is enabled, the REL indicator turns on. Changing measurement functions changes the relative offset value to the established relative offset value and state for that measurement function.

Enabling and disabling relative offset from the front panel

To enable and use the relative offset feature, press the **REL** key on the virtual front panel. The reading (which becomes the relative offset value) is subtracted from itself, causing the SMU to display a zero value. The reading is stored for use with subsequent measurements. Press the **REL** key a second time to disable the relative offset.

Defining a relative offset value from the front panel

You can establish a relative offset value for the selected measurement function.

To establish a relative offset value from the virtual front panel:

1. Press the **CONFIG** key and then the **REL** key.
2. Select the measurement function (**CURRENT**, **VOLTAGE**, **OHMS**, or **WATTS**).
3. Press **ENTER** or the navigation wheel. The present relative offset value is displayed.
4. Set the relative offset value.
5. With the relative offset value displayed, press the **ENTER** key or the navigation wheel, and then press the **EXIT (LOCAL)** key to back out of the menu structure.

Relative offset commands

Relative offset commands are summarized in the following table.

Command*	Description
To set relative offset values:	
<code>smuX.measure.rel.leveli = relval</code>	Set current relative offset value
<code>smuX.measure.rel.levelp = relval</code>	Set power relative offset value
<code>smuX.measure.rel.levelr = relval</code>	Set resistance relative offset value
<code>smuX.measure.rel.levelv = relval</code>	Set voltage relative offset value
To enable or disable relative offset:	
<code>smuX.measure.rel.enablei = smuX.REL_OFF</code>	Disable current relative offset
<code>smuX.measure.rel.enablep = smuX.REL_OFF</code>	Disable power relative offset
<code>smuX.measure.rel.enabler = smuX.REL_OFF</code>	Disable resistance relative offset
<code>smuX.measure.rel.enablev = smuX.REL_OFF</code>	Disable voltage relative offset
<code>smuX.measure.rel.enablei = smuX.REL_ON</code>	Enable current relative offset
<code>smuX.measure.rel.enablep = smuX.REL_ON</code>	Enable power relative offset
<code>smuX.measure.rel.enabler = smuX.REL_ON</code>	Enable resistance relative offset
<code>smuX.measure.rel.enablev = smuX.REL_ON</code>	Enable voltage relative offset

* `smuX` can be `smua` for channel A or `smub` for channel B

Relative offset programming example

The programming example below performs a current measurement, uses it as the relative offset value, and enables current relative offset:

```
-- Measure and set present current value as the relative offset.
smua.measure.rel.leveli = smua.measure.i()
-- Enable current relative offset.
smua.measure.rel.enablei = smua.REL_ON
```

Filters

The filter feature lets you set the filter response to stabilize noisy measurements. The 2606B uses a digital filter, which is based on reading conversions. The displayed, stored, or transmitted reading is calculated using one or more reading conversions (from 1 to 100).

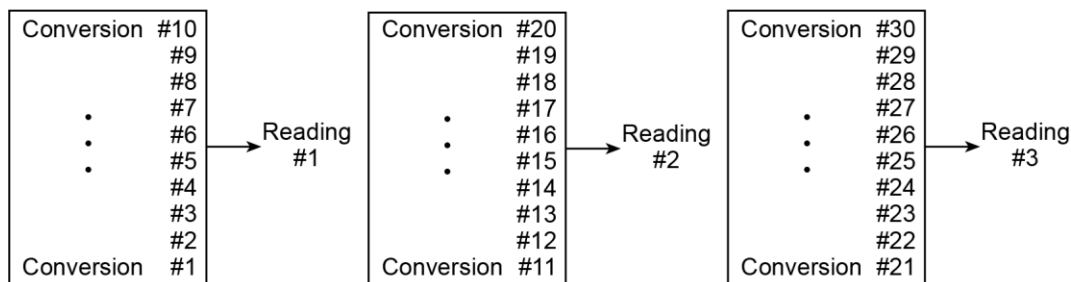
Filter types

The 2606B provides two averaging filters and a median filter. The power-on default is the repeating filter.

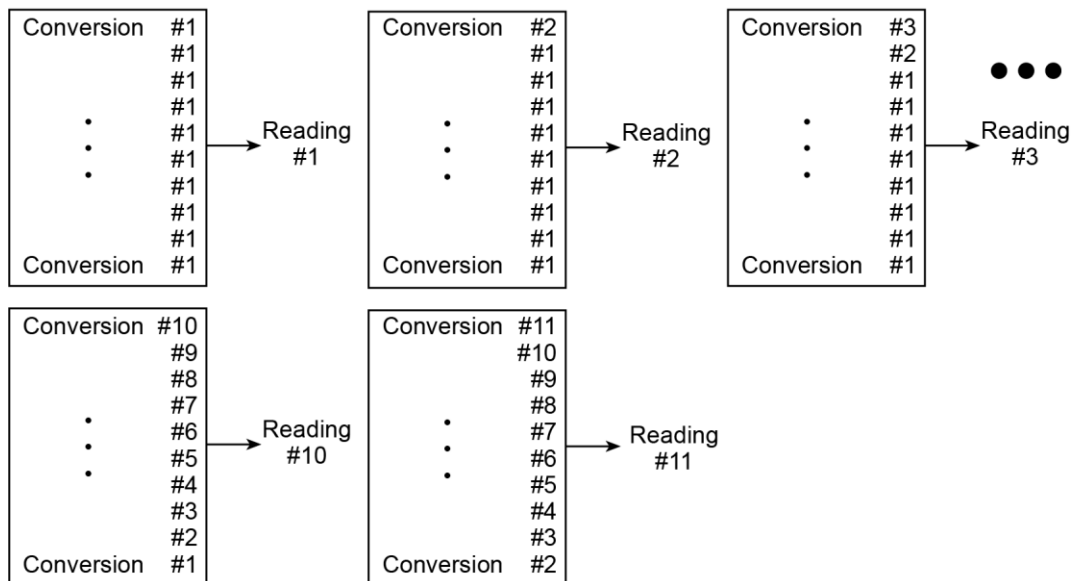
The averaging filters are repeating and moving, as shown in the following figure. For the repeating filter, the stack (filter count) is filled, and the conversions are averaged to yield a reading. The stack is then cleared, and the process starts over.

Figure 56: Repeating and moving average filters

Repeating filter, readings = 10



Moving filter, readings = 10

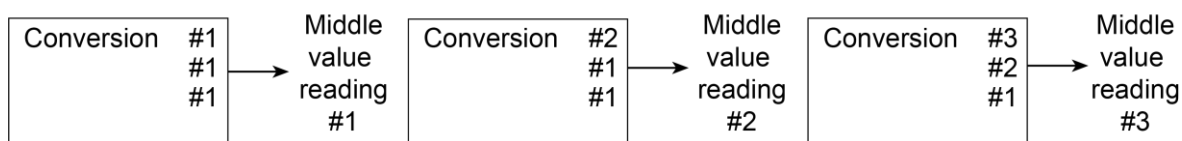


The moving filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the measurement conversions are averaged, yielding a reading. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The stack is averaged again, yielding a new reading.

The median filter is used to pass the reading that is nearest to the middle from a group of readings that are arranged according to size. The median filter uses a first-in, first-out stack similar to the moving filter. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The median is then redetermined.

Figure 57: Median filter

Median filter, readings = 3



When a moving filter or a median filter is first enabled, the stack is empty. The first reading conversion is placed in the stack and is then copied to the other stack locations to fill it. Thus, the first filtered reading is the same as the first reading conversion. The normal moving filter process continues. A true average or median reading is only yielded when the stack is filled with new reading conversions (no copies in the stack). For example, in the figure for the moving filter, it makes ten filtered readings to fill the stack with new reading conversions. The first nine filtered readings are calculated using copied reading conversions.

Response time

The filter parameters have speed and accuracy trade-offs for the time needed to display, store, or output a filtered reading. These affect the number of reading conversions for speed versus accuracy and response to input signal changes.

The filter type and count affect the overall reading speed. The moving average filter is much faster than the repeat average filter because the instrument does not have to refill the filter stack for each reading. Also, the number of readings averaged affects reading speed; as the number of readings averaged increases, the reading speed decreases.

Enabling the filter from the front panel

The filter is enabled by pressing the **FILTER** key on the virtual front panel. The **FILT** indicator is on while the filter is enabled. Pressing **FILTER** a second time disables the filter.

Configuring the filter from the front panel

Filter type and count are configured from the filter configuration menu on the virtual front panel. The same filter configuration is used for all measurement functions.

To configure the filter:

1. Press the **CONFIG** key and then the **FILTER** key.
2. Select **TYPE**, and then select the filter type: **AVERAGE** or **MEDIAN**.
 - **AVERAGE**: Use this menu item to select an averaging filter, then select the averaging filter type: **MOVING** or **REPEAT**.
 - **MEDIAN**: Use this menu item to select a median filter. The **MOVING** filter type is the only option.
3. Select **COUNT**, and then specify the filter count (1 to 100 readings).

Setting the filter using a remote interface

The following table summarizes the filter commands. See the [TSP command reference](#) (on page 15-1) for details about commands.

Filter commands

Command*	Description
<code>smuX.measure.filter.count = count</code>	Set filter count (1 to 100)
<code>smuX.measure.filter.enable = smuX.FILTER_ON</code>	Enable filter
<code>smuX.measure.filter.enable = smuX.FILTER_OFF</code>	Disable filter
<code>smuX.measure.filter.type = smuX.FILTER_MEDIAN</code>	Select median filter type
<code>smuX.measure.filter.type = smuX.FILTER_MOVING_AVG</code>	Select moving average filter type
<code>smuX.measure.filter.type = smuX.FILTER_REPEAT_AVG</code>	Select repeating average filter type

* `smuX` can be `smua` for channel A or `smub` for channel B

Filter programming example

The programming example below illustrates how to set the following filter options:

- **Filter type:** Moving average
- **Filter count:** 10
- **Filter state:** Enabled

```
-- Set the program count to 10.
smua.measure.filter.count = 10
-- Set the moving average filter type.
smua.measure.filter.type = smua.FILTER_MOVING_AVG
-- Enable the filter.
smua.measure.filter.enable = smua.FILTER_ON
```

High-capacitance mode

The Keithley Instruments 2606B has a high-capacitance mode.

Because the source-measure unit (SMU) can measure low current, issues can arise when driving a capacitive load. The pole formed by the load capacitance and the current range resistor can cause a phase shift in the SMU voltage control loop. This shift can lead to overshoot, ringing, and instability. Due to the large dynamic range of current measurement and wide range of internal resistors, the operating conditions for a given capacitive load can vary.

Based on the type, some test applications may require capacitors larger than 10 nF. While running test scripts, it may not be possible to disconnect the capacitor from the integrated circuit and extract accurate data. For this purpose, you can use the high-capacitance mode to minimize overshoot, ringing, and instability.

This section provides the details that you need to estimate performance based on load capacitance and measurement conditions.

Understanding high-capacitance mode

The source-measure unit (SMU) in the 2606B drives 10 nF of capacitance in normal operation. Typically, an internal capacitor across the current measuring element provides phase lead to compensate for the phase lag caused by the load capacitance on the output. This internal capacitance across the range resistance limits the speed for a specific measurement range.

The SMU in the 2606B implements frequency compensation to achieve the highest throughput possible for a 10 nF or less load. In addition, you must consider the settling time, voltage range, measure delay, the quality of the capacitor, the current measure range resistor, and the load resistor.

In normal operation, the SMU in the 2606B can drive capacitive loads as large as 10 nF. In high-capacitance mode, the SMU can drive a maximum of 50 μ F of capacitance.

NOTE

When high-capacitance mode is enabled, a minimum load capacitance of 100 nF is recommended. In absence of this minimum load capacitance, overshoot and ringing may occur.

Highest throughput is achieved by using normal operation. In high-capacitance mode, the speed of the 2606B SMU is reduced to compensate for the larger load capacitance. Stability is achieved by inserting an internal capacitance across the current measuring element of the SMU. This internal capacitor limits the speed for the source and measurement ranges. Therefore, when optimizing the speed of your test configuration in high-capacitance mode, you must consider the settling time, voltage, and current ranges, measure delay, quality of the load capacitor, and load resistance.

Source settling times

Each 2606B source-measure unit (SMU) can drive up to 50 μF of a capacitance in high-capacitance mode. To accomplish this, the speed of the 2606B SMU is reduced. Source settling times increase when high-capacitance mode is enabled. The following table compares examples of the differences in source settling times between normal and high-capacitance modes.

Source settling times		
Range	Normal mode	High-capacitance mode
100 mV	50 μs	200 μs
1 V	50 μs	200 μs
6 V	100 μs	200 μs
20 V	150 μs	7,000 μs

In high-capacitance mode, the frequency compensation capacitance across the measure range resistors increases. This increase causes longer settling times on some current measure ranges. The same range elements that are used to measure current are used to source current. As a result, the current limit response times respond in a similar manner. Examples of the differences in source settling times are shown in the following table.

Current measure and source settling times		
Current measure range	Normal mode (typical)	High capacitance mode (typical)
1 A to 3 A	80 μs	120 μs ($R_{\text{LOAD}} > 2 \Omega$)
100 mA	100 μs	100 μs
10 mA	80 μs	100 μs
1 mA	100 μs	3,000 μs
100 μA	150 μs	3,000 μs
10 μA	500 μs	230,000 μs
1 μA	2,000 μs	230,000 μs

When high-capacitance mode is enabled, the amount of time to change the current measure range increases for each SMU. The current measure range and the current limit range are locked together. Setting the current limit automatically updates the measure range.

Adjusting the voltage source

When driving large capacitive loads with high-capacitance mode enabled, the response time may be lengthened by the current limit. For example, see the table titled "Current measure and source settling times" in [Understanding source settling times](#) (on page 5-53). If a 1 μF capacitor charges to 10 V in 10 μs with a 1 A limit and the limit is set to 100 nA, the charging time is 100 seconds, as shown in the following equation.

$$i = C \frac{\Delta V}{\Delta t}$$

The total response times while in high-capacitance mode are a combination of the time spent charging the capacitor (current limit) or the response time, whichever is greater. There is a direct relationship between the current limit and the charging time. As the current limit decreases, the amount of time required to charge the capacitor increases.

Understanding the capacitor

Based on the capacitor dielectric absorption, the settling time may change and the values in the "Current measure and source settling times" table in [Understanding source settling times](#) (on page 5-53) may differ.

NOTE

Tantalum or electrolytic capacitors are well known for long dielectric absorption settling times. Film capacitors and ceramics perform better, with NPO/COG dielectric ceramics yielding the best settling response.

Charging the capacitor and making readings

To charge and read a capacitor in high-capacitance mode:

1. Set the current limit to a value that is higher than the value that is used for the measurement (for example, if measuring at 10 μ A, the initial current limit can be set to 1 A).
2. After the capacitor charges, lower the current limit and measure range to obtain the current measurement.

Enabling high-capacitance mode

Before enabling high-capacitance mode, note the following:

- It is important to read [High-capacitance mode](#) (on page 5-52) to understand the impact of high-capacitance mode.
- 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.
- Test the DUT for the appropriate current limit and output voltages.
- 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.

CAUTION

Failure to test the DUT for the appropriate current limit and output voltages can result in damage to or destruction of the DUT.

To enable high-capacitance mode from the virtual front panel:

1. Press the **CONFIG** key.
2. Press the **SRC** key and then select **HIGHC-MODE**.
3. Select **SRC-ENABLE** and then press the navigation wheel (or the **ENTER** key).
4. Select **ENABLE** and then press the navigation wheel (or the **ENTER** key).
5. Press the **EXIT (LOCAL)** key to back out of the menu structure.

To enable high-capacitance mode using a remote interface:

Turning on high-capacitance mode has the following effects on the SMU* settings:

- `smuX.measure.autorangei` is set to `smuX.AUTORANGE_FOLLOW_LIMIT` and cannot be changed.
- Current ranges below 1 μA are not accessible.
- If `smuX.source.limiti` is less than 1 μA , it is raised to 1 μA .
- If `smuX.source.rangei` is less than 1 μA , it is raised to 1 μA .
- If `smuX.source.lowrangei` is less than 1 μA , it is raised to 1 μA .
- If `smuX.measure.lowrangei` is less than 1 μA , it is raised to 1 μA .

* `smuX` can be `smua` for channel A or `smub` for channel B

Measuring current using high-capacitance mode

The following inputs are required to test leakage using the factory leakage script, as shown in the following script example.

- **SMU:** Sets the 2606B source-measure unit to use
- **levelv:** Sets the output voltage level
- **limiti:** Sets the current limit for discharging or charging the capacitor
- **sourcedelay:** Solves the following equation to determine the amount of time before making a current reading:

$$i = C \frac{\Delta V}{\Delta t}$$

Where: i is the `limiti` setting (current limit)

- **measurei:** Sets the current measure range
- **measuredelay:** Defines the delay to wait after lowering the current limit before making the measurement

Script example

Use the `smuX.source.highc` attribute to set and control the options for high-capacitance mode.

The programming examples and figure below illustrate how to enable high-capacitance mode on SMU A.

1. To enable high-capacitance mode, send:

```
-- Enables high-capacitance mode.
smua.source.highc = smua.ENABLE
```

2. To run the `i_leakage_measure()` function in the `KIHighC` factory script, send:

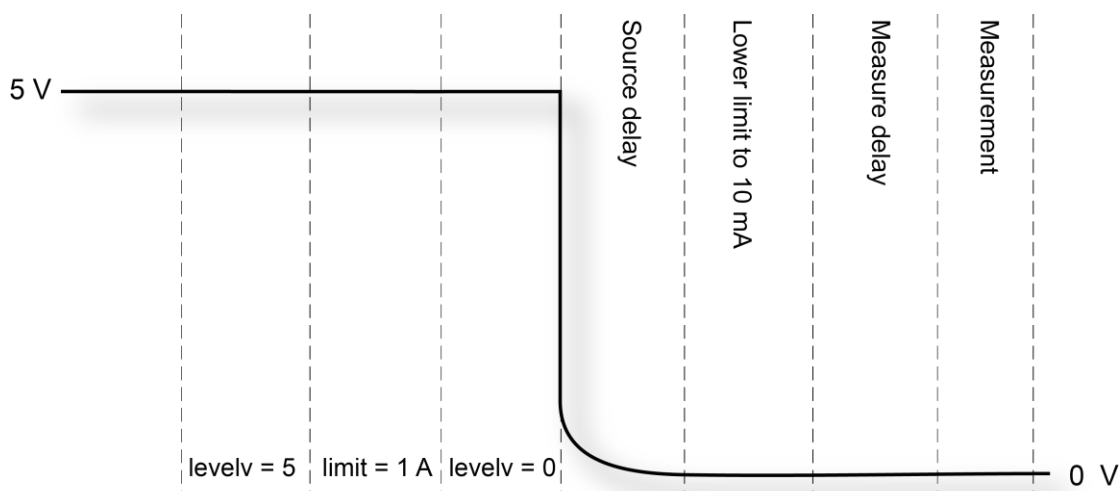
```
-- Charges the capacitor.
smua.source.levelv = 5
smua.source.output = smua.OUTPUT_ON
delay(1)
imeas = i_leakage_measure(smua, 0, 1, 300e-3, 10e-6, 100e-3)
-- The parameters in the i_leakage_measure() function represent
-- the following:
-- smu = smua
-- levelv = 0 V
-- limiti = 1 A
-- sourcedelay = 300 ms
-- measurei = 10 uA range
-- measuredelay = 100 ms
```

NOTE

Adjust the voltage level and source delays based on:

- The value and type of capacitor
- The magnitude of the voltage step
- The current measure range

Figure 58: Enabling high-capacitance mode



Reading buffers

In this section:

Overview	6-1
Front-panel reading buffer control	6-1
Remote reading buffer programming	6-6
Buffer examples	6-13

Overview

Reading buffers capture measurements, ranges, instrument status, and output state of the Keithley Instruments 2606B. The 2606B has two default reading buffers for each channel. In addition to the default buffers, you can create user-defined reading buffers. You can use the reading buffers to acquire readings.

You can access reading buffers from the virtual front panel or over the remote command interface.

The default reading buffers can store more than 60,000 readings if you enable the options for timestamps and source values. To store 140,000 readings internally, you can disable the timestamps and source values.

You can save reading buffers to internal nonvolatile memory in the instrument or to a USB flash drive.

Once you save the reading buffers to a USB flash drive, insert the USB flash drive into the USB port on your computer to view the data in any compatible data analysis application or to transfer the data from the USB flash drive to your computer.

Front-panel reading buffer control

The dedicated reading buffers can be configured, stored, and recalled when in local mode operation. Use the virtual front panel to navigate and configure the reading buffers options and to save and recall stored readings.

Reading buffer options

The following list outlines the menu structure and menu items associated with virtual front-panel reading buffer control. This section provides a description for each reading buffer option. Use the procedure in [Configuring reading buffers](#) (on page 6-3) as a guideline to configure these reading buffer options.

CHANA-BUFF: Configures Channel A buffer.

- DEST: Sets data storage destination (buffer 1, buffer 2, or none).
- BUFFER1: Configure Buffer 1.
 - CLEAR: Clear buffer (YES or NO).
 - ELEMENTS: Enable (ON) or disable (OFF) data storage elements.
 - SRC-VAL: Enable or disable source values.
 - TSTAMP: Enable or disable timestamps.
- BUFFER2: Configure Buffer 2.
 - CLEAR: Clear buffer (YES or NO).
 - ELEMENTS: Enable (ON) or disable (OFF) data storage elements.
 - SRC-VAL: Enable or disable source values.
 - TSTAMP: Enable or disable timestamps.

CHANB-BUFF: Configures Channel B buffer.

- DEST: Sets data storage destination (buffer 1, buffer 2, or none).
- BUFFER1: Configure Buffer 1.
 - CLEAR: Clear buffer (YES or NO).
 - ELEMENTS: Enable (ON) or disable (OFF) data storage elements.
 - SRC-VAL: Enable or disable source values.
 - TSTAMP: Enable or disable timestamps.
- BUFFER2: Configure Buffer 2.
 - CLEAR: Clear buffer (YES or NO).
 - ELEMENTS: Enable (ON) or disable (OFF) data storage elements.
 - SRC-VAL: Enable or disable source values.
 - TSTAMP: Enable or disable timestamps.

Configuring reading buffers

To configure reading buffers from the virtual front panel:

NOTE

Enabling or disabling the source value or the timestamp is optional.

1. Press the **CONFIG** key.
2. Press the **STORE** key and then choose one of the following:
 - CHANA-BUFF
 - CHANB-BUFF
3. To select a storage destination, select the **DEST** option, and then choose one of the following:
 - CHANA-BUFF1 for Channel A or CHANB-BUFF1 for Channel B
 - CHANA-BUFF2 for Channel A or CHANB-BUFF2 for Channel B
 - NONE
4. Select **BUFFER1** or **BUFFER2**.
5. Clear the buffer by turning the navigation wheel to select **CLEAR > YES**.
6. Turn the navigation wheel to highlight **ELEMENTS**, and then press the navigation wheel (or the **ENTER** key).

NOTE

You must clear the reading buffer before you can enable or disable the source value or the timestamp options.

7. Configure the timestamp elements of the reading buffer:
 - a. Turn the navigation wheel to highlight **TSTAMP**.
 - b. Press the navigation wheel (or the **ENTER** key).
 - c. Select **OFF** or **ON** and then press the navigation wheel (or the **ENTER** key).
8. Configure the source value elements of the reading buffer:
 - a. Turn the navigation wheel to highlight **SRC-VAL**.
 - b. Press the navigation wheel (or the **ENTER** key).
 - c. Select **OFF** or **ON**.
9. Press the **EXIT (LOCAL)** key to return to the main menu.

Appending or overwriting existing reading buffers

When storing data to a reading buffer that already holds data, the new data can be appended to the reading buffer data, or it can overwrite the old data.

To configure the instrument to append or overwrite measurements the next time data is acquired:

1. Press the **CONFIG** key.
2. Select **STORE**, and then select **STORAGE-MODE**. The Storage Mode menu is shown.
3. Select one of the following:
 - **APPEND**
 - **OVERWRITE**
4. Press the **EXIT (LOCAL)** key to return to the main menu.

Storage operation

Use this option to initiate a storage operation and to configure the number of readings to acquire during the storage operation. The reading count can be more than 60,000 if timestamps and source values are enabled. The count can be more than 140,000 if timestamps and source values are disabled.

NOTE

To store the maximum number of readings in a reading buffer, disable the source values and timestamps for that reading buffer.

To specify the number of readings and initiate the storage operation:

1. From the virtual front panel, press the **STORE** key, and then select **TAKE_READINGS**.
2. Use the navigation wheel to select the number of readings.
3. Press the navigation wheel to switch to edit mode.
4. Turn the navigation wheel to change the numeric value.
5. Press the navigation wheel to save the numeric value.
6. Press the **ENTER** key to save the count.
7. Press the **OUTPUT ON/OFF** control to start making readings.

NOTE

If the output-off mode is ZERO or the output is already on, the instrument starts acquiring readings when the ENTER key is pressed. Otherwise, the instrument starts acquiring readings when the output is turned on.

Saving reading buffers

You can save the dedicated reading buffers to nonvolatile memory or you can save them to a USB flash drive.

The instrument restores the dedicated reading buffers from internal nonvolatile memory when the instrument is turned off and back on.

NOTE

You can also save reading buffer data to a .csv file using the web interface.

Saving the reading buffers to nonvolatile memory

After the measurements are complete, you can save the reading buffer data to the nonvolatile memory in the instrument.

To save the reading buffer data:

1. From the virtual front panel, press the **STORE** key, and then select **SAVE**.
2. Select **INTERNAL** to save to internal nonvolatile memory.
3. Select one of the following:
 - **SMUA_BUFFER1**
 - **SMUA_BUFFER2**
 - **SMUB_BUFFER1**
 - **SMUB_BUFFER2**
4. The virtual front panel displays *Saving...* This may take awhile.
5. Press the **EXIT (LOCAL)** key to return to the main menu.

Saving the reading buffer to a USB flash drive

After the measurements are complete, you can save the reading buffer data to a USB flash drive.

To save the reading buffer data to a USB flash drive:

1. Insert the USB flash drive into the USB port.
2. Press the **STORE** key and use the navigation wheel to select **SAVE**.
3. Select **USB1**.
4. Select one of the following file formats:
 - **CSV**
 - **XML**
5. Use the navigation wheel to select the reading buffer.
6. Use the navigation wheel to change the file name.
7. Press the navigation wheel or the **ENTER** key to save the file.
8. Press the **EXIT (LOCAL)** key to return to the main menu.

Recalling readings

To recall the data stored in a reading buffer from the virtual front panel:

1. Press the **RECALL** key.
2. Select **DATA** or **STATISTICS**.
3. Select the buffer to display: CHANA-BUFF1 or CHANB-BUFF2. The data or statistics are displayed.
 - If data is being recalled, the reading display is on the top left, and the buffer location number is on the right. The source values are on the lower left side of the display (if enabled); the timestamp (if used) is on the lower right side.
 - If statistics are being recalled, the information includes values for MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, and PK-PK.
 - The source display field identifies the buffer: SrcA1 (buffer 1), SrcA2 (buffer 2), SrcB1 (buffer 1), SrcB2 (buffer 2).
4. To view data or statistics, turn navigation wheel or use the **CURSOR** keys.
5. To exit from the reading buffer recall screens, press the **EXIT (LOCAL)** key.

The buffer location number indicates the memory location of the source-measure reading. For example, location #000001 indicates that the displayed source-measure reading is stored at the first memory location.

If the timestamp is enabled, the first source-measure reading stored in the buffer (#0000001) is timestamped at 0.000 seconds. Subsequent readings are timestamped relative to when the first measurement was made. The interval between readings depends on the reading rate.

Remote reading buffer programming

You can get readings by making overlapped or sequential measurements. Overlapped commands do not finish executing before the next command starts. Sequential commands complete execution before the next command starts executing.

The measured value is not the only component of a reading. The measurement status (for example, “In Compliance” or “Overranged”) is also an element of data associated with a particular reading.

All routines that return measurements can store the measurements in the reading buffers. Overlapped measurements always return readings in a reading buffer. Nonoverlapped measurement functions can return single-point measurement values or store multiple values in a reading buffer.

A reading buffer is based on a Lua table. The measurements are retrieved with ordinary array accesses. If `rb` is a reading buffer, the first measurement is accessed as `rb[1]` and the ninth measurement as `rb[9]`. The additional information in the table is accessed as additional members of the table.

The load, save, and write operations for reading buffers function differently in the remote state. From a remote command interface, you can extract data from reading buffers as the instrument acquires the data.

Dedicated reading buffer designations

Each source-measure unit (SMU) contains dedicated reading buffers:

- `smua.nvbuffer1` (buffer 1 for channel A)
- `smua.nvbuffer2` (buffer 2 for channel A)
- `smub.nvbuffer1` (buffer 1 for channel B)
- `smub.nvbuffer2` (buffer 2 for channel B)

To access a reading buffer, include the name of the SMU in the attribute. For example, the following command stores readings from channel A into buffer 1:

```
smua.measure.overlappedi (smua.nvbuffer1)
```

Reading buffer commands

The following tables summarize commands associated with the reading buffers. See [TSP command reference](#) (on page 15-1) for detailed reading buffer command information.

Reading buffer commands*

Command	Description
Commands to save and clear readings	
<code>smuX.savebuffer (smuX.nvbufferY)</code>	Saves the reading buffer to the nonvolatile memory on the 2606B.
<code>smuX.nvbuffer1.clear()</code>	Clears buffer 1.
<code>smuX.nvbuffer2.clear()</code>	Clears buffer 2.
<code>mybuffer = smuX.makebuffer (n)</code>	Creates a dynamically allocated buffer for <i>n</i> readings.
<code>mybuffer = nil</code>	Deletes the dynamically allocated buffer.
<code>savebuffer (smuX.nvbuffer1, "csv", "/usb1/mybuffer.csv")</code>	Saves the reading buffer to a USB flash drive.

Command	Description
Commands to store readings	
<code>smuX.measure.count = count</code>	The number of measurements to acquire.
<code>smuX.measure.overlappedi(rbuffer)</code>	Makes current measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.overlappediv(ibuffer, vbuffer)</code>	Makes both current and voltage measurements; stores current readings in <i>ibuffer</i> and stores voltage readings in <i>vbuffer</i> .
<code>smuX.measure.overlappedp(rbuffer)</code>	Makes power measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.overlappedr(rbuffer)</code>	Makes resistance measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.overlappedv(rbuffer)</code>	Makes overlapped voltage measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.v(rbuffer)</code>	Makes voltage measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.i(rbuffer)</code>	Makes current measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.iv(ibuffer, vbuffer)</code>	Makes both current and voltage measurements; stores current readings in <i>ibuffer</i> and stores voltage readings in <i>vbuffer</i> .
<code>smuX.measure.r(rbuffer)</code>	Makes resistance measurements; stores readings in <i>rbuffer</i> .
<code>smuX.measure.p(rbuffer)</code>	Makes power measurements; stores readings in <i>rbuffer</i> .
<code>smuX.trigger.measure.v(rbuffer)</code>	Configures voltage measurements to be made during a sweep, including where readings are stored (<i>rbuffer</i>).
<code>smuX.trigger.measure.i(rbuffer)</code>	Configures current measurements to be made during a sweep, including where readings are stored (<i>rbuffer</i>).
<code>smuX.trigger.measure.r(rbuffer)</code>	Configures resistance measurements to be made during a sweep, including where readings are stored (<i>rbuffer</i>).
<code>smuX.trigger.measure.p(rbuffer)</code>	Configures power measurements to be made during a sweep, including where readings are stored (<i>rbuffer</i>).
<code>smuX.trigger.measure.iv(ibuffer, vbuffer)</code>	Configures both current and voltage measurements to be made during a sweep, including where readings are stored; current readings are stored in <i>ibuffer</i> and voltage readings are stored in <i>vbuffer</i> .

Command	Description
Commands to access readings	
<code>printbuffer(start_index, end_index, st_1, st_2, ... st_n)</code>	Prints data from buffer subtables: ▪ <i>start_index</i> (starting index of values to print) ▪ <i>end_index</i> (ending index of values to print) ▪ <i>st_1, st_2, ... st_n</i> (subtables from which to print, each separated by a comma)

* `smuX` can be `smua` for channel A or `smub` for channel B

Buffer storage control attributes

The following table contains buffer storage control attributes.

NOTE

Before changing the `collectsourcevalues`, `collecttimestamps`, or `timestampresolution` attributes, you must clear the buffer using the `smuX.nvbuffer1.clear()` or `smuX.nvbuffer2.clear()` command.

Buffer storage attribute	Description
<code>appendmode</code>	The append mode is either off or on. When the append mode is off, a new measurement to this buffer overwrites the previous contents. When the append mode is on, the first new measurement is stored at the end of the existing data. This attribute is off when the buffer is created.
<code>cachemode</code>	When this attribute is on, the reading buffer cache improves access speed to reading buffer data. When running successive operations that overwrite reading buffer data without running any commands that automatically invalidate the cache, the reading buffer may return stale cache data. This attribute is initialized to on when the buffer is created.
<code>collectsourcevalues</code>	When this attribute is on, source values are stored with readings in the buffer. This value, off or on, can be changed only when the buffer is empty. When the buffer is created, this attribute is initialized to off.
<code>collecttimestamps</code>	When this attribute is on, timestamps are stored with readings in the buffer. This value, off or on, can be changed only when the buffer is empty. When the buffer is created, this attribute is initialized to off.
<code>fillcount</code>	The reading buffer fill count sets the number of readings to store before restarting at index 1. If the value is 0, then the capacity of the buffer is used. This attribute is only used when the <code>fillmode</code> attribute is set to <code>FILL_WINDOW</code> .
<code>fillmode</code>	The reading buffer fill mode controls how new data is added to the reading buffer. When this attribute is set to <code>FILL_ONCE</code> , the reading buffer does not overwrite readings. If the buffer fills up, new readings are discarded. When this attribute is set to <code>FILL_WINDOW</code> , new readings are added after existing data until the buffer holds <code>fillcount</code> elements. Once there are <code>fillcount</code> elements, new data starts overwriting data starting at index 1.
<code>timestampresolution</code>	The timestamp resolution, in seconds. When the buffer is created, its initial resolution is 0.000001 seconds. At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. This value can be increased for long tests.

Buffer read-only attributes

The following table contains buffer read-only attributes that access the buffer parameters.

Attribute	Description
basetimestamp	The timestamp of when the reading at <code>rb[1]</code> was stored, in seconds from midnight January 1, 1970 GMT. See Time and date values (on page 15-2) for additional details.
capacity	The total number of readings that can be stored in the reading buffer.
n	The number of readings in the reading buffer.
next	This attribute indicates where the next element that is added to the reading buffer is stored.

Buffer storage control programming examples

The programming examples below illustrate the use of buffer storage control attributes.

Command	Description
<code>smua.nvbuffer1.collectsourcevalues = 1</code>	Enable source value storage.
<code>smua.nvbuffer1.appendmode = 1</code>	Enable buffer append mode.
<code>smua.nvbuffer1.collecttimestamps = 0</code>	Disable timestamp storage.
<code>smua.nvbuffer1.timestampresolution = 0.001</code>	Set timestamp resolution to 0.001024 s.
<code>smua.nvbuffer1.fillcount = 50</code>	Set 50 as the number of readings the buffer stores before restarting at index 1.
<code>smua.nvbuffer1.fillmode = 0</code>	Set the reading buffer to fill once (do not overwrite old data).

Buffer read-only attribute programming examples

The following programming examples illustrate use of buffer read-only attributes.

Command	Description
<code>number = smua.nvbuffer1.n</code>	Request the number of readings in the buffer.
<code>buffer_size = smua.nvbuffer1.capacity</code>	Request buffer size.

Statistic attributes

Use the `smuX.buffer.getstats()` function to access the reading buffer data statistics. The table below lists the attributes that you can use to access the reading buffer statistics.

Attributes for accessing reading buffer data

Attribute	When returned	Description
<code>n</code>	Always	The number of data points on which the statistics are based
<code>mean</code>	When <code>n > 0</code>	The average of all readings added to the buffer
<code>stddev</code>	When <code>n > 1</code>	The standard deviation of all readings (samples) added to the buffer
<code>min</code>	When <code>n > 0</code>	A table containing data about the minimum reading value added to the buffer
<code>max</code>	When <code>n > 0</code>	A table containing data about the maximum reading value added to the buffer

If `n` equals zero (0), all other attributes are `nil` because there is no data to base any statistics on. If `n` equals 1, the `stddev` attribute is `nil` because the standard deviation of a sample size of 1 is undefined.

The `min` and `max` entries have the attributes described in the following table (`bufferVar` is the name of the buffer). See [smuX.buffer.getstats\(\)](#) (on page 15-193) for additional information.

Min and max entry attributes

Attribute	Description
<code>measurefunction</code>	String indicating the function that was measured for the reading (current, voltage, ohms, or watts)
<code>measurerrange</code>	The full-scale range value for the measurement range used when the measurement was made
<code>reading</code>	The reading value
<code>sourcefunction</code>	String indicating the source function at the time of the measurement (current or voltage)
<code>sourceoutputstate</code>	String indicating the state of the source (off or on)
<code>sourcerange</code>	Full-scale range value for the source range used when the measurement was made
<code>sourcevalue</code>	If <code>bufferVar.collectsourcevalues</code> is enabled, the sourced value in effect at the time of the reading
<code>status</code>	Status value for the reading; the status value is a floating-point number that encodes the status value into a floating-point value
<code>timestamp</code>	If <code>bufferVar.collecttimestamps</code> is enabled, the timestamp, in seconds, between when the reading was acquired and when the first reading in the buffer was acquired; adding this value to the base timestamp produces the actual time the measurement was acquired

Example:

The following programming example illustrates how to output mean and standard deviation statistics from buffer 1:

```
statistics = smua.buffer.getstats(smua.nvbuffer1)
print(statistics.mean, statistics.stddev)
```

Reading buffer attributes

Use the reading buffer attributes to access the reading buffer data. The table below displays the attributes that you can use to access the reading buffer data.

Recall attribute*	Description
measurefunctions	An array (a Lua table) of strings indicating the function measured for the reading (current, voltage, ohms, or watts).
measureranges	An array (a Lua table) of full-scale range values for the measure range used when the measurement was made.
readings	An array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly; that is, <code>rb[2]</code> and <code>rb.readings[2]</code> access the same value.
sourcefunctions	An array (a Lua table) of strings indicating the source function at the time of the measurement (current or voltage).
sourceoutputstates	An array (a Lua table) of strings indicating the state of the source (off or on).
sourceranges	An array (a Lua table) of full-scale range values for the source range used when the measurement was made.
sourcevalues	If enabled, an array (a Lua table) of the sourced values in effect at the time of the reading.
statuses	An array (a Lua table) of status values for all the readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value. See Buffer status (on page 6-13).
timestamps	If enabled, an array (a Lua table) of timestamps, in seconds, of when each reading occurred. These are relative to the <code>basetimestamp</code> for the buffer. See Reading buffer commands (on page 6-7).

* The default attribute is `readings`, which can be omitted.

Examples:

The following programming example illustrates how to output 100 channel A readings from buffer 1:

```
printbuffer(1, 100, smua.nvbuffer1.readings)
```

Similarly, the following outputs 100 channel A corresponding source values from buffer 1:

```
printbuffer(1, 100, smua.nvbuffer1.sourcevalues)
```

The default reading attribute is `readings`, which can be omitted. If `readings` is omitted, the following also outputs 100 channel A readings from buffer 1:

```
printbuffer(1, 100, smua.nvbuffer1)
```

Buffer status

The buffer reading status attribute includes the status information as a numeric value; see the following table for values. For example, to access status information for the second element of SMU channel A buffer 1, use the following command:

```
stat_info = smua.nvbuffer1.statuses[2]
```

Buffer status bits

Bit	Name	Hex value	Description
B0	Reserved	0x01	Reserved for future use
B1	Overtemp	0x02	Overtemperature condition
B2	AutoRangeMeas	0x04	Measure range was autoranged
B3	AutoRangeSrc	0x08	Source range was autoranged
B4	4Wire	0x10	4-wire (remote) sense mode was enabled
B5	Rel	0x20	Relative offset was applied to a reading
B6	Compliance	0x40	Source function was in compliance
B7	Filtered	0x80	Reading was filtered

Dynamic reading buffers

Reading buffers can also be allocated dynamically. You create and allocate the dynamic reading buffers with the `smuX.makebuffer(n)` command, where *n* is the number of readings the buffer can store. For example, the following command allocates a reading buffer named `mybuffer` that can store 100 readings:

```
mybuffer = smua.makebuffer(100)
```

You can delete allocated reading buffers by sending the following command:

```
mybuffer = nil
```

You can use dynamically allocated reading buffers interchangeably with the `smuX.nvbufferY` buffers that are described in [Dedicated reading buffer designations](#) (on page 6-7).

Buffer examples

Dedicated reading buffer example

The following programming example illustrates how to store data using dedicated reading buffer 1. In the example, the 2606B loops for voltages from 0.01 V to 1 V with 0.01 V steps (performing a staircase sweep), stores 100 current readings and source values in buffer 1, and then recalls all 100 readings and source values.

```
-- Restore 2606B defaults.
smua.reset()
-- Select channel A display.
display.screen = display.SMUA
-- Display current.
display.smua.measure.func = display.MEASURE_DCAMPS
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Clear buffer 1.
smua.nvbuffer1.clear()
-- Enable append buffer mode.
smua.nvbuffer1.appendmode = 1
-- Enable source value storage.
smua.nvbuffer1.collectsourcevalues = 1
-- Set the count to 1.
smua.measure.count = 1
-- Select the source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set the bias voltage to 0 V.
smua.source.levelv = 0.0
-- Turn on the output.
smua.source.output = smua.OUTPUT_ON
-- Loop for voltages from 0.01 V to 1 V.
for v = 1, 100 do
    -- Set the source voltage.
    smua.source.levelv = v * 0.01
    -- Measure the current and store in nvbuffer1.
    smua.measure.i(smua.nvbuffer1)
end
-- Turn off the output.
smua.source.output = smua.OUTPUT_OFF
-- Output readings 1 to 100.
printbuffer(1, smua.nvbuffer1.n, smua.nvbuffer1.readings)
-- Output source values 1 to 100.
printbuffer(1, smua.nvbuffer1.n, smua.nvbuffer1.sourcevalues)
```

Dual buffer example

The programming example below shows a script that stores current and voltage readings using buffer 1 for current and buffer 2 for voltage readings. The 2606B stores 100 current and voltage readings and then recalls all 100 sets of readings.

```
-- Restore 2606B defaults.
smua.reset()
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select measure V autorange.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Clear buffer 1.
smua.nvbuffer1.clear()
-- Clear buffer 2.
smua.nvbuffer2.clear()
-- Set buffer count to 100.
smua.measure.count = 100
-- Set measure interval to 0.1 s.
smua.measure.interval = 0.1
-- Select source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Output 1 V.
smua.source.levelv = 1
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Store current readings in buffer 1, voltage readings in buffer 2.
smua.measure.overlappediv(smua.nvbuffer1, smua.nvbuffer2)
-- Wait for buffer to fill.
waitcomplete()
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
-- Output buffer 1 readings 1 to 100.
printbuffer(1, 100, smua.nvbuffer1)
-- Output buffer 2 readings 1 to 100.
printbuffer(1, 100, smua.nvbuffer2)
```


Dynamically allocated buffer example

The programming example below illustrates how to store data to an allocated buffer called `mybuffer`. The 2606B stores 100 current readings in `mybuffer` and then recalls all the readings.

```
-- Restore 2606B defaults.
smua.reset()
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select measure V autorange.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Set the buffer count to 100.
smua.measure.count = 100
-- Set the measure interval to 0.1 s.
smua.measure.interval = 0.1
-- Select the source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set the source voltage to output 1 V.
smua.source.levelv = 1
-- Turn on the output.
smua.source.output = smua.OUTPUT_ON
-- Create a temporary reading buffer.
mybuffer = smua.makebuffer(smua.measure.count)
-- Store current readings in mybuffer.
smua.measure.overlappedi(mybuffer)
-- Wait for the buffer to fill.
waitcomplete()
-- Turn off the output.
smua.source.output = smua.OUTPUT_OFF
-- Output readings 1 to 100 from mybuffer.
printbuffer(1, 100, mybuffer)
-- Delete mybuffer.
mybuffer = nil
```

Triggering

In this section:

Triggering	7-1
Remote triggering overview.....	7-1
Using the remote trigger model	7-3
SMU event detectors.....	7-7
Using trigger events to start actions on trigger objects.....	7-10
Digital I/O port and TSP-Link synchronization lines	7-11
Timers	7-13
Event blenders	7-21
LAN triggering overview	7-22
Command interface triggering	7-24
Trigger generator	7-24
Manual triggering	7-25
Interactive triggering.....	7-25
Hardware trigger modes.....	7-29
Understanding synchronous triggering modes	7-33
Digital I/O	7-37
TSP-Link trigger lines.....	7-41

Triggering

Triggering allows you to source signals and capture measurements when an input signal or combination of input signals meets a set of conditions that you set. Triggering controls the timing of when source and measure operations happen during a sweep. See [Sweep operation](#) (on page 5-34) for details on sweeping.

Remote triggering overview

There are two programming methods for triggering:

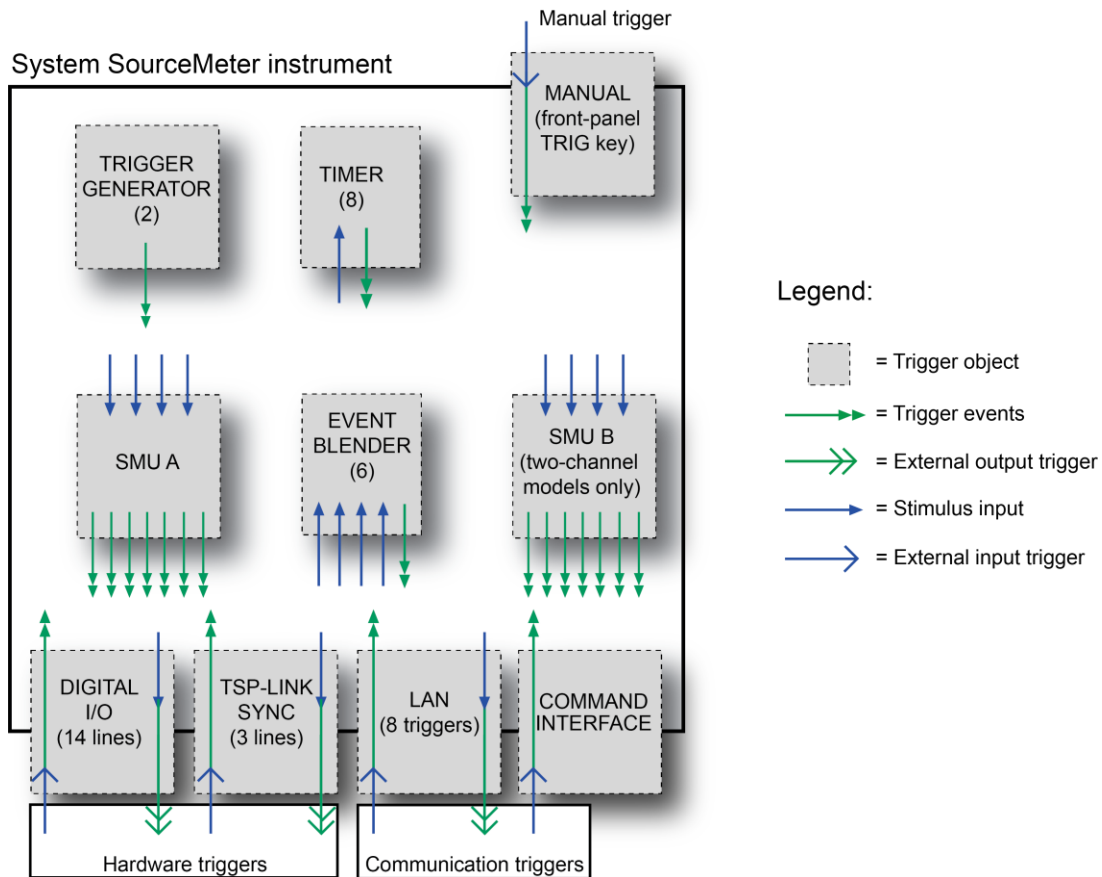
- Using the trigger model
- Interactive triggering

You can obtain very precise timing and synchronization between channels of multiple instruments using the trigger model to control the actions of the source-measure unit (SMU). To achieve such precise timing, use a static trigger configuration. When a static trigger configuration is not possible, you can use the interactive triggering method to control the timing and actions of the SMU.

Both programming methods use trigger objects. Trigger objects generate and monitor trigger events. External triggers are possible using digital I/O, TSP-Link® synchronization lines, LAN, command interface, and the manual trigger (the TRIG key).

The following figure graphically represents all the trigger objects of the 2606B instrument.

Figure 59: Triggering overview



Trigger events are identified by means of an event ID. The following table describes the trigger event IDs.

Trigger event IDs*	
Event ID**	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measure action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed

* Use the name of the trigger event ID to set a stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).

** `smuX` can be `smua` for channel A or `smub` for channel B

Using the remote trigger model

The source-measure unit (SMU) in the 2606B has a remote trigger model that supports a wide range of triggering features for source sweeps, triggered measurements, and pulse actions.

Measurements using the trigger model can be made synchronously with sourcing actions or they can be made asynchronously. The following figures graphically illustrate both modes of the remote trigger model.

Figure 60: Remote trigger model: Normal (synchronous) mode

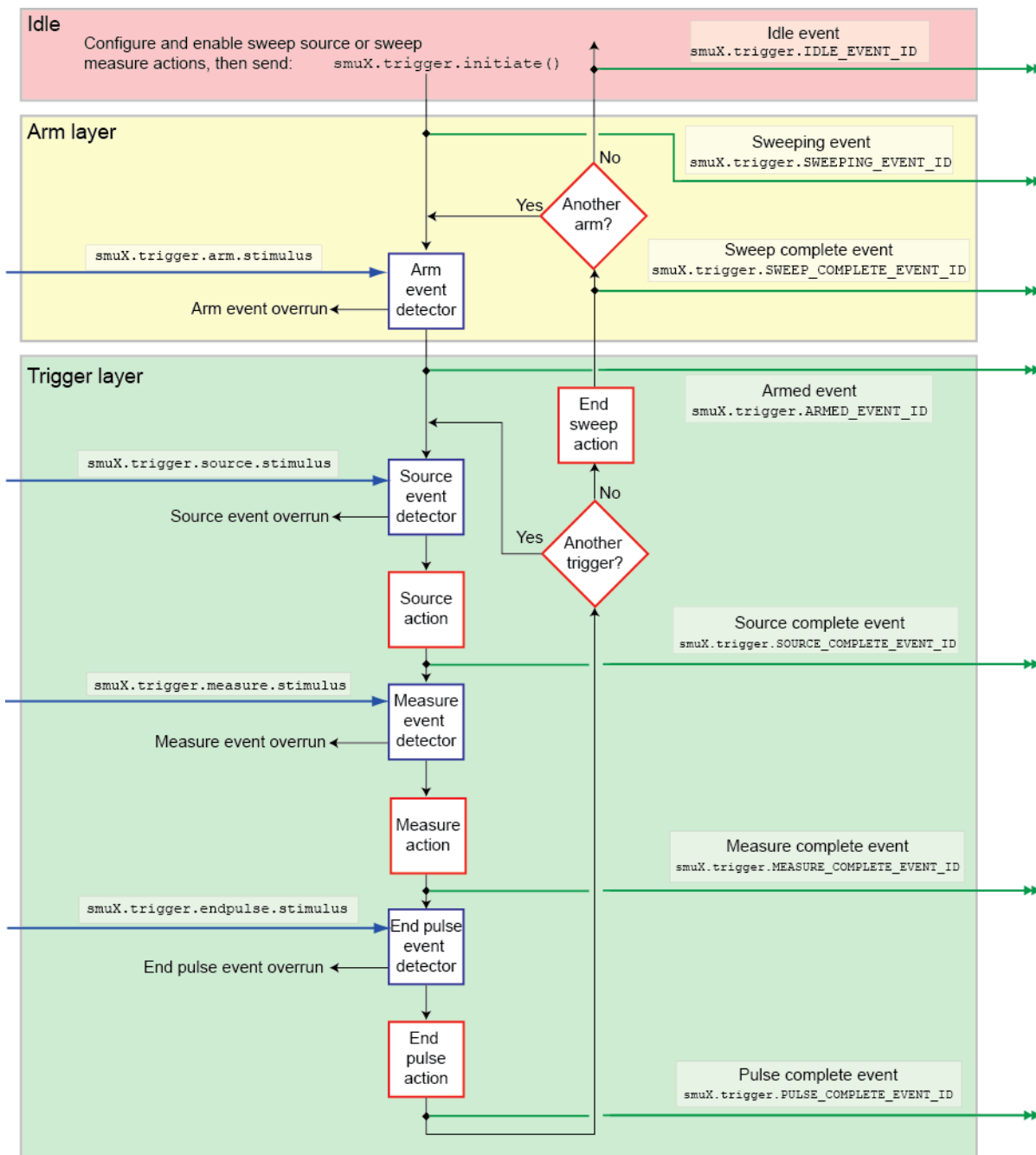
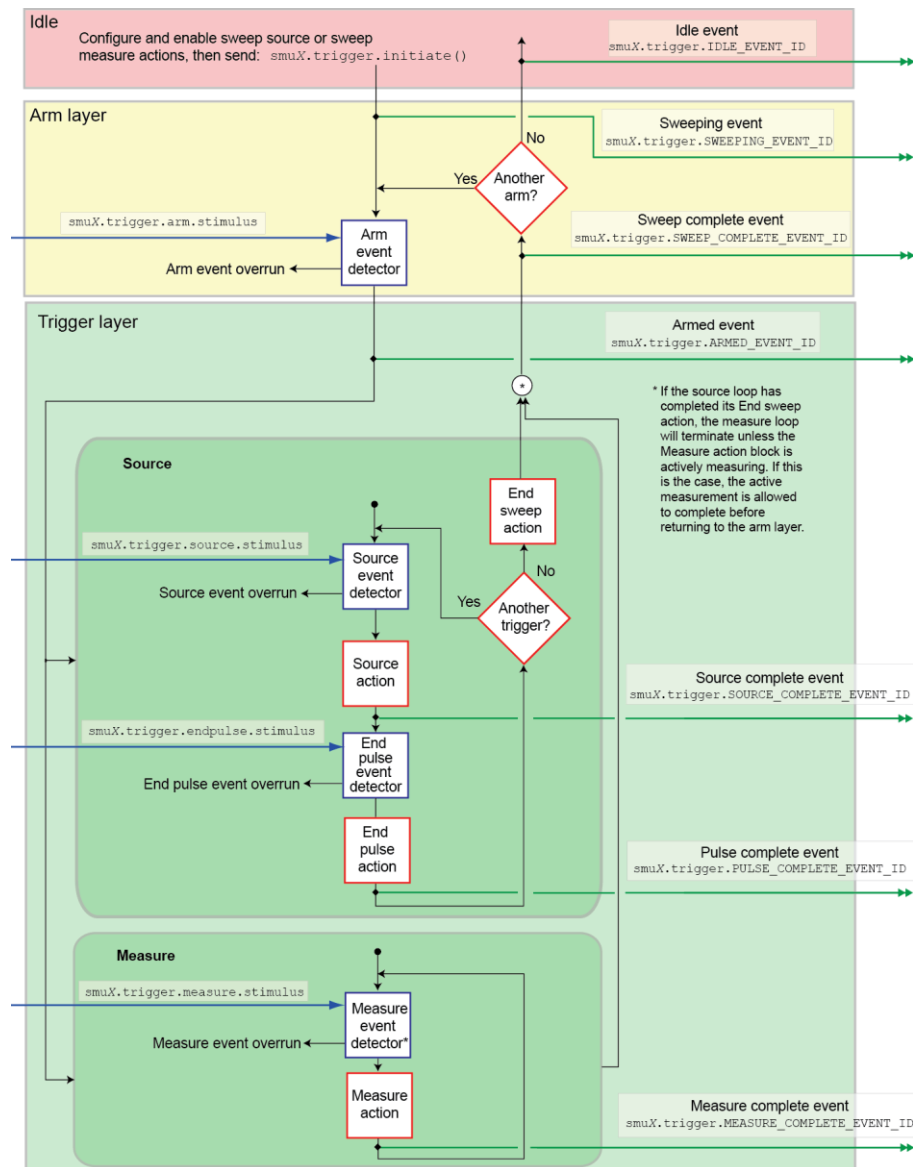


Figure 61: Remote trigger model: Asynchronous mode



When the `smuX.trigger.measure.action` attribute is set to `smuX.DISABLE` or `smuX.ENABLE`, the trigger model operates in synchronous measurement mode. When it is set to `smuX.ASYNC`, it operates in asynchronous mode.

Each section of the trigger model performs a function:

Idle state	If a sweep is not in process, the SMU is in the idle state. Use the <code>smuX.trigger.initiate()</code> function to move the SMU from the idle state to the arm layer.
Arm layer	Begins a sweep. Each sweep starts and ends in the arm layer.
Trigger layer	All source, measurement, and pulse actions occur in the trigger layer. ▪ Source: Outputs the programmed voltage or current source value. ▪ Measurement: Where the current, voltage, resistance, and power measurements occur. ▪ End pulse: The end pulse action sources the idle (or bias) level if the pulse mode is enabled.

The remote trigger model dictates the sequence of operation for the SMU when it is configured to perform a sweep. When the SMU comes to an event detector, it suspends operation and waits for the event you have assigned to the stimulus input. If no event is assigned, the SMU continues uninterrupted past the event detector and through the trigger model. When the SMU comes to an action block, it performs the appropriate action, if enabled. The SMU loops through the arm and trigger layers until the programmed arm and trigger counts are satisfied.

Configuring source and measure actions

You can configure the source action using any of the following functions:

```
smuX.trigger.source.linearY()
smuX.trigger.source.logY()
smuX.trigger.source.listY()
```

Where:

Y = Source function (v = voltage, i = current)

Source functions cannot be changed within a sweep.

To enable the source action, set the `smuX.trigger.source.action` attribute to `smuX.ENABLE`.

The source-measure unit (SMU) can be configured to perform any or all available measurements during a sweep using the `smuX.trigger.measure.Y()` function. To enable the measure action for a simple synchronous sweep, set the `smuX.trigger.measure.action` attribute to `smuX.ENABLE`. To enable the measure action for an asynchronous sweep, set the `smuX.trigger.measure.action` attribute to `smuX.ASYNC`.

NOTE

In asynchronous mode, trigger your measurements before the source completes the sweep (before the end sweep action occurs). If the source loop has completed its end sweep action, the measure loop terminates unless the measure action block is actively measuring. If this is the case, the active measurement is allowed to complete before returning to the arm layer.

NOTE

For more information about the sweep functions, refer to [Sweep operations](#) (on page 5-34).

Configured source and measure delays are imposed when the SMU executes the source and measure action blocks. Additionally, if the measure count setting is greater than one, then the measure count is satisfied each time the measure action is performed.

The arm and trigger counts must be set to control how many times the SMU executes the source and measure actions. The arm count indicates the number of times to execute the complete sweep. The trigger count sets the number of loops in the trigger layer. Typically, you set the trigger count to be equal to the number of points in the configured sweep. If the trigger count is not equal to the number of points configured in the sweep, then one of the following occurs:

- If the trigger count is greater than the number of points in a sweep, the SMU satisfies the trigger count by restarting the sweep values from the beginning. The points are configured by the `smuX.trigger.source.linearY()`, `smuX.trigger.source.logY()`, or `smuX.trigger.source.listY()` command.
- If the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

For example, configure a three-point linear voltage sweep from 1 V to 3 V, with the trigger count set to 2. The SMU outputs 1 V, 2 V. If the trigger count is set to 6, the SMU outputs the values 1 V, 2 V, 3 V, 1 V, 2 V, 3 V, repeating the source values twice in a single sweep.

Enabling pulse mode sweeps using the end pulse action

Enable pulse mode sweeps using the end pulse action. The example command below illustrates how to configure pulse mode sweeps by setting the end pulse action:

```
smua.trigger.endpulse.action = smua.SOURCE_IDLE
```

You can use timers to configure the pulse width and period (see [Timers](#) (on page 7-13) for more information). To disable pulse mode sweeps, set the `smuX.trigger.endpulse.action` attribute to `smuX.SOURCE_HOLD`.

SMU event detectors

As shown in [Using the remote trigger model](#) (on page 7-3), the source-measure unit (SMU) has multiple event detectors to control the timing of various actions, as shown in the table below. Each event detector monitors for the trigger event assigned to the associated stimulus input. Operation through the trigger model is delayed at the event detector until the programmed trigger event occurs.

If the stimulus input is set to zero (0), the SMU continues uninterrupted through the remote trigger model.

Event detectors

Event detector	Function
Arm	Controls entry into the trigger layer of the trigger model.
Source	Controls execution of the source action.
Measure	Controls execution of the measurement action.
End pulse	Controls execution of the end pulse action.

For the SMU, action overruns occur when a new trigger is detected before the previous trigger is acted upon. When the trigger model is configured for asynchronous measurements, a measurement trigger generates an overrun if the SMU is not ready to start a new measurement.

Clearing SMU event detectors

When an event detector is cleared, the event detector discards previously detected trigger events. This prevents the source-measure unit (SMU) from using trigger events that were detected during the last sweep or while it is in the arm layer, and allows it to start monitoring for new trigger events.

SMU event detectors are automatically cleared when:

- A sweep is initiated using the `smuX.trigger.initiate()` function*.
- The SMU moves from the arm layer into the trigger layer and the `smuX.trigger.autoclear` attribute is enabled.

* `smuX` can be `smua` for channel A or `smub` for channel B

Using the TRIG key to trigger a sweep

You can configure the source-measure unit (SMU) to perform a sweep in which each source step is triggered by the virtual front-panel TRIG key. The source action is preceded by the source event detector. The SMU pauses operation at an event detector until a programmed event occurs. The SMU can be programmed to wait at the source event detector (that is, not start the source action) until the TRIG key is pressed.

To configure the virtual front-panel TRIG key to trigger the source action, assign the trigger event created by the TRIG key (`display.trigger.EVENT_ID`) to the source stimulus input (`smuX.trigger.source.stimulus`).

The programming example below illustrates how to configure a 10-point linear voltage sweep on SMU A, in which each step is triggered by the TRIG key:

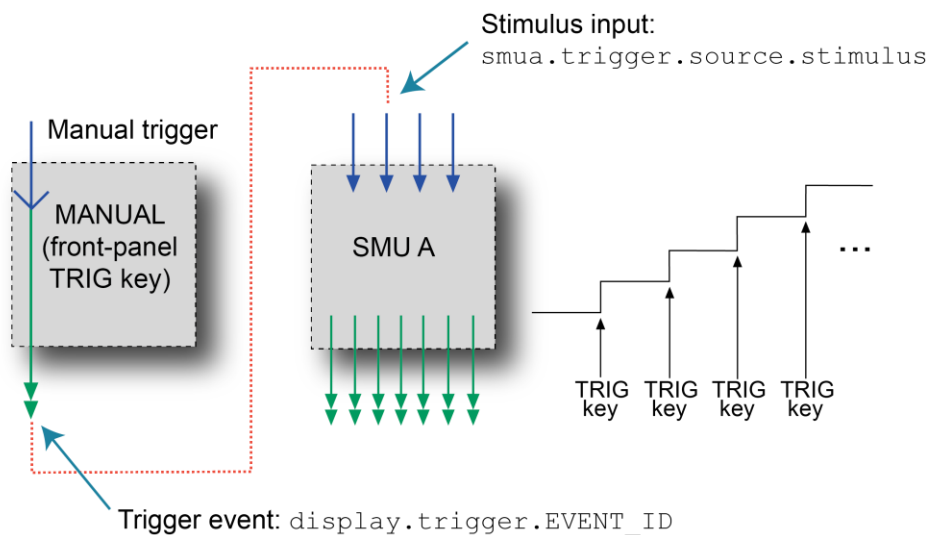
```

-- Configure a 10-point source voltage sweep.
smua.trigger.source.linearv(1, 10, 10)
smua.trigger.source.action = smua.ENABLE
-- Configure the TRIG key press as an input trigger for source action.
smua.trigger.source.stimulus = display.trigger.EVENT_ID
-- Configure the SMU to execute a single 10-point sweep.
smua.trigger.count = 10
smua.trigger.arm.count = 1
-- Turn on the output in preparation for the sweep.
smua.source.output = smua.OUTPUT_ON
-- Start the sweep and clear the event detectors.
smua.trigger.initiate()
-- The SMU waits for the virtual front-panel TRIG key press before executing
-- each source action.
-- Wait for the sweep to complete.
waitcomplete()

```

The following figure graphically illustrates this example. See [Sweep operation](#) (on page 5-34) for more information about sweep operation.

Figure 62: Front-panel TRIG key triggering



Legend:

-  = Trigger object
-  = Trigger events
-  = Stimulus input
-  = External input trigger

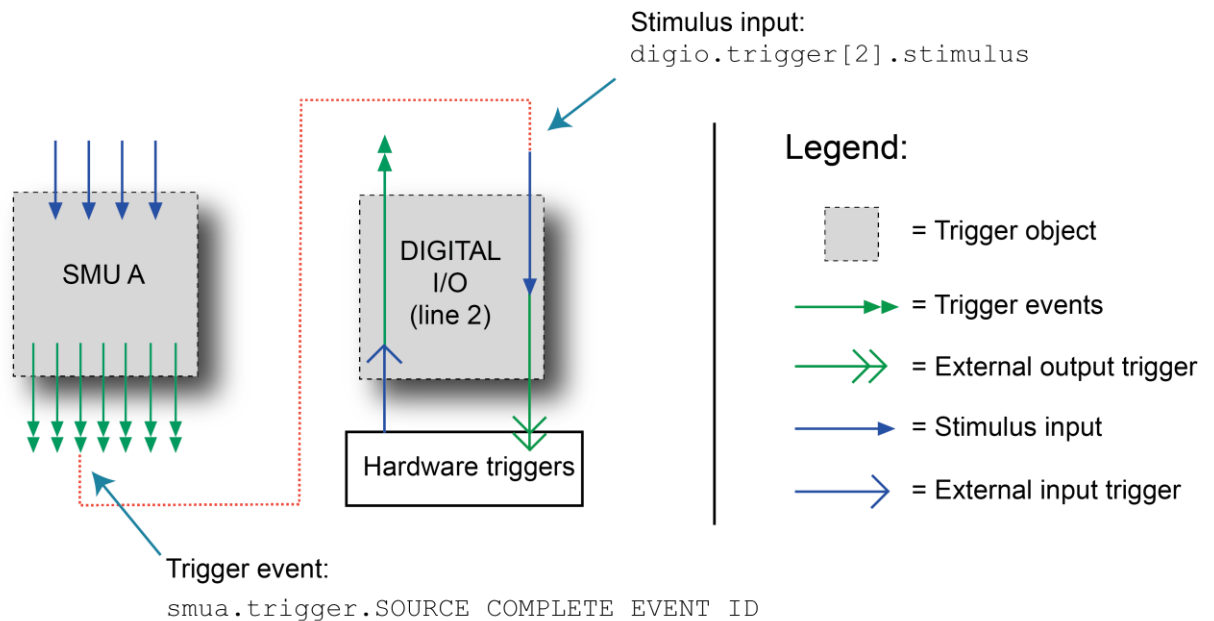
Using trigger events to start actions on trigger objects

You can configure trigger objects to respond to events generated by other trigger objects, such as using a digital I/O trigger to initiate a sweep. To configure a trigger object to monitor for an event, assign the event ID of the trigger event to the stimulus input. When the specified trigger event occurs, the trigger object performs an action. The programming example below illustrates how to generate a digital I/O line 2 output trigger pulse for each SMU A source complete event:

```
-- Configure digio line 2 to generate an output trigger pulse each
-- time SMU A generates a source complete event.
digio.trigger[2].stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID
```

The following figure illustrates this example.

Figure 63: Using trigger events to start actions



A stimulus input can be configured to monitor for only one trigger event ID at a time. To monitor more than one event, use an event blender. See [Event blenders](#) (on page 7-21) for more information.

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 the operation event registers of the status model. Refer to [Status model](#) (on page 10-1) and the sections on each trigger object for details on the conditions that generate an action overrun.

Digital I/O port and TSP-Link synchronization lines

Each module of the 2606B has two sets of hardware lines that can be used for triggering: 14 digital I/O lines and three TSP-Link® synchronization lines. These trigger objects can be configured and controlled in the same way.

See [Digital I/O](#) (on page 7-37) for more information about connections and direct control of the digital I/O and TSP-Link synchronization lines.

Mode

The mode indicates the type of edge the hardware lines detect as an external input trigger. Mode also indicates the type of signal generated as an external output trigger. The following table describes the hardware trigger modes for the hardware trigger lines. For additional detail, refer to [Hardware trigger modes](#) (on page 7-29).

NOTE

To disable triggering on the hardware trigger lines, set the mode to bypass. This allows direct control of the line.

Hardware trigger mode summary

Trigger mode	Output		Input
	Unasserted	Asserted	Detects
Bypass	N/A	N/A	N/A
Either edge	High	Low	Either
Falling edge	High	Low	Falling
Rising edge	The programmed state of the line determines if the behavior is similar to RisingA or RisingM: - High similar to RisingA - Low similar to RisingM		
RisingA	High	Low	Rising
RisingM	Low	High	Not available
Synchronous	High latching	Low	Falling
SynchronousA	High latching	High	Falling
SynchronousM	High	Low	Rising

Pulse width

Specifies the pulse width of the output trigger signal when the hardware line is asserted.

Trigger configuration on hardware lines

You can configure the 2606B to send digital signals to trigger external instruments. You can link the output triggers to the completion of certain source-measure actions to enable hardware handshaking. The programming example below illustrates this.

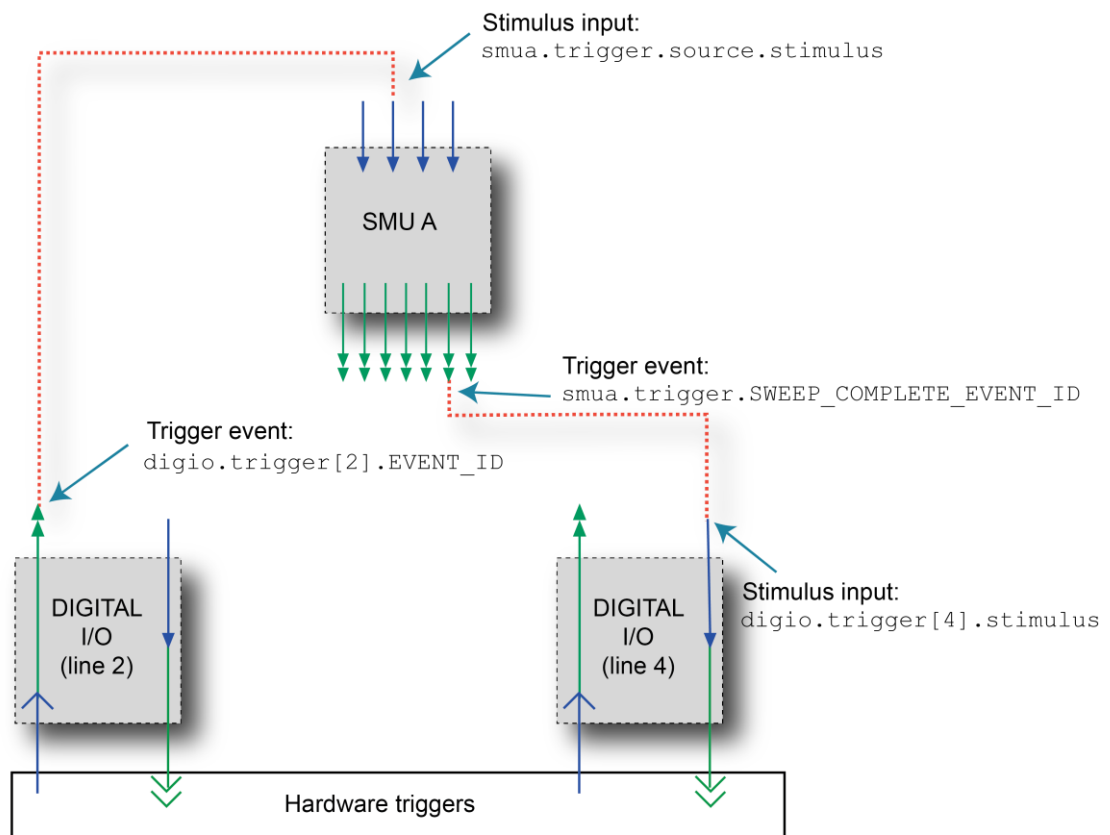
```

-- Configure the 2606B to detect a rising
-- edge on digital I/O line 2.
digio.trigger[2].mode = digio.TRIG_RISINGA
digio.trigger[2].clear()
-- Configure SMU A to start its source action when a
-- trigger event occurs on digital I/O line 2.
smua.trigger.source.stimulus = digio.trigger[2].EVENT_ID
-- Configure digital I/O line 4 to output a 1 ms
-- rising-edge trigger pulse at the completion of
-- the SMU sweep.
digio.trigger[4].mode = digio.TRIG_RISINGM
digio.trigger[4].pulsewidth = 0.001
digio.trigger[4].stimulus = smua.trigger.SWEEP_COMPLETE_EVENT_ID

```

The triggering setup for this example is shown in the following figure.

Figure 64: External instrument triggering



Legend:

- = Trigger object
- = Trigger events
- = External output trigger
- = Stimulus input
- = External input trigger

Action overruns on hardware lines

An action overrun occurs when a trigger event is received before the digital I/O or TSP-Link® line is ready to process it. The generation of an action overrun is dependent upon the trigger mode selected for that line. For more details on the causes of action overruns, see [Hardware trigger modes](#) (on page 7-29). Use the status model to monitor for the occurrence of action overruns. For details, see [Status model](#) (on page 10-1).

Timers

A timer is a trigger object that performs a delay when triggered. You can use timers to create delays, to start measurements, and step the source value at timed intervals. When a delay expires, the timer generates a trigger event. The 2606B has eight independent timers available for each module.

Timer attributes

Each timer has attributes that you can configure. These attributes are described in the following sections.

Count

The count sets the number of events to generate each time the timer generates a trigger event. Each event is separated by the delay set by the `trigger.timer[N].delay` command.

To configure the count, use the `trigger.timer[N].count` command.

Set the count number to 0 (zero) to cause the timer to generate trigger events indefinitely.

Timer delays

You can configure timers to perform the same delay each time or set up a delay list that allows the timer to sequence through an array of delay values. All delay values are specified in seconds.

A delay is the period after the timer is triggered and before the timer generates a trigger event. The programming example below illustrates how to configure timer 3 for a 10 s delay:

```
trigger.timer[3].delay = 10
```

You can configure a custom delay list to 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 value in the list. The timer repeats the delay list after all the elements in the delay list have been used. The programming example below illustrates how to configure timer 3 for delays of 2, 10, 15, and 7 seconds:

```
-- Configure timer 3 to complete delays of 2 s, 10 s,  
-- 15 s, and 7 s.  
trigger.timer[3].delaylist = {2, 10, 15, 7}
```

NOTE

Assigning a value to the delay attribute is the same as configuring it with a one-element delay list.

Pass-through mode

When enabled, the timer generates a trigger event immediately when it is triggered. The timer generates additional trigger events each time a delay expires. If the pass-through attribute is disabled, the timer does not generate a trigger event until after the first delay elapses. The programming example below illustrates how to configure timer 3 by enabling pass-through mode:

```
trigger.timer[3].passthrough = true
```

Triggering a timer

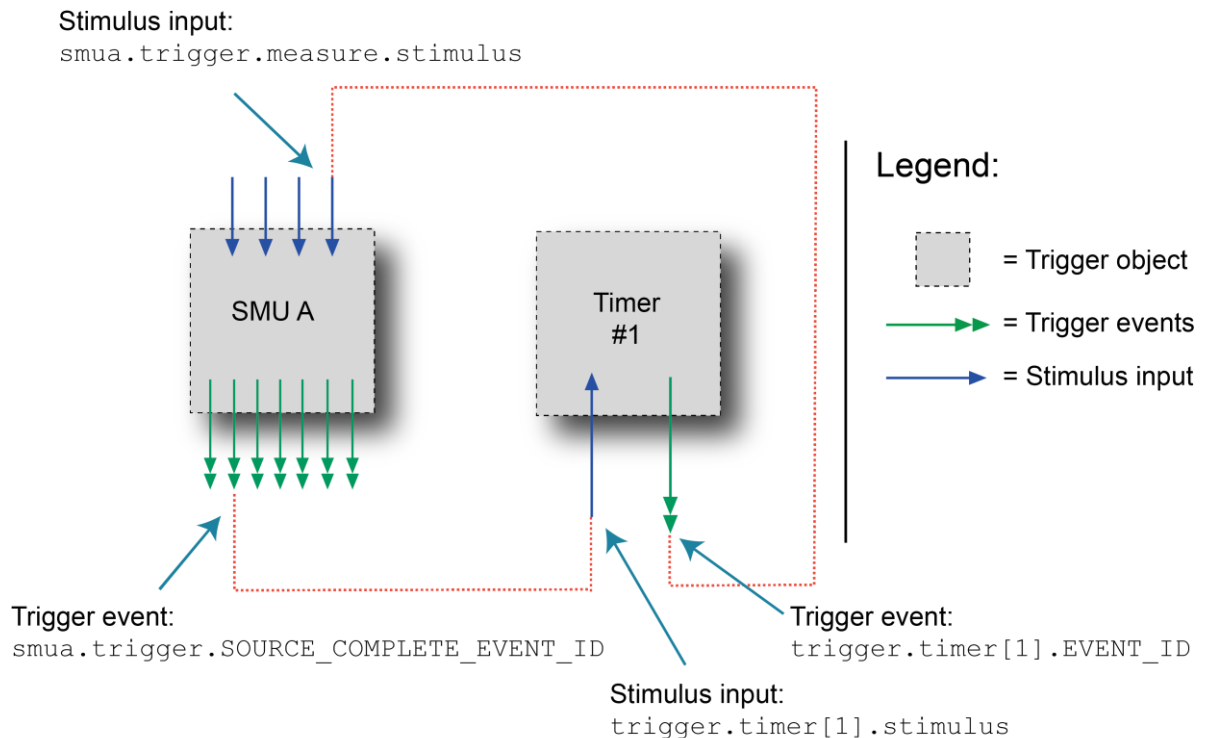
You can configure a timer to start a delay when a trigger object generates a trigger event. Timers cannot be started with a command. A trigger event from a trigger object must be used to initiate a delay.

Assigning the stimulus attribute

Assign an event ID to the `trigger.timer[N].stimulus` attribute to configure the timer to start a delay when a specific trigger event occurs. The programming example below illustrates how to configure a source-delay-measure (SDM) cycle.

```
-- Configure the timer to begin when source action completes.
trigger.timer[1].stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID
-- SMUA delay before a measurement begins.
smua.trigger.measure.stimulus = trigger.timer[1].EVENT_ID
```

Figure 65: Using a timer for an SDM cycle



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 [Status model](#) (on page 10-1).

Using timers to perform pulse mode sweeps

You can use timers to control the pulse width during a pulsed sweep. To create a pulse train, a second timer must be used to configure the pulse period. The following topics provide examples that show a single pulse output and a pulse train output.

NOTE

To create a pulse, the SMU end pulse action `smuX.trigger.endpulse.action` must be set to `smuX.SOURCE_IDLE`.

Single pulse example

The SMU programming example below illustrates how to use a single timer to control the pulse width of a single-shot pulse measurement. The programming example configures the timer and SMU as follows:

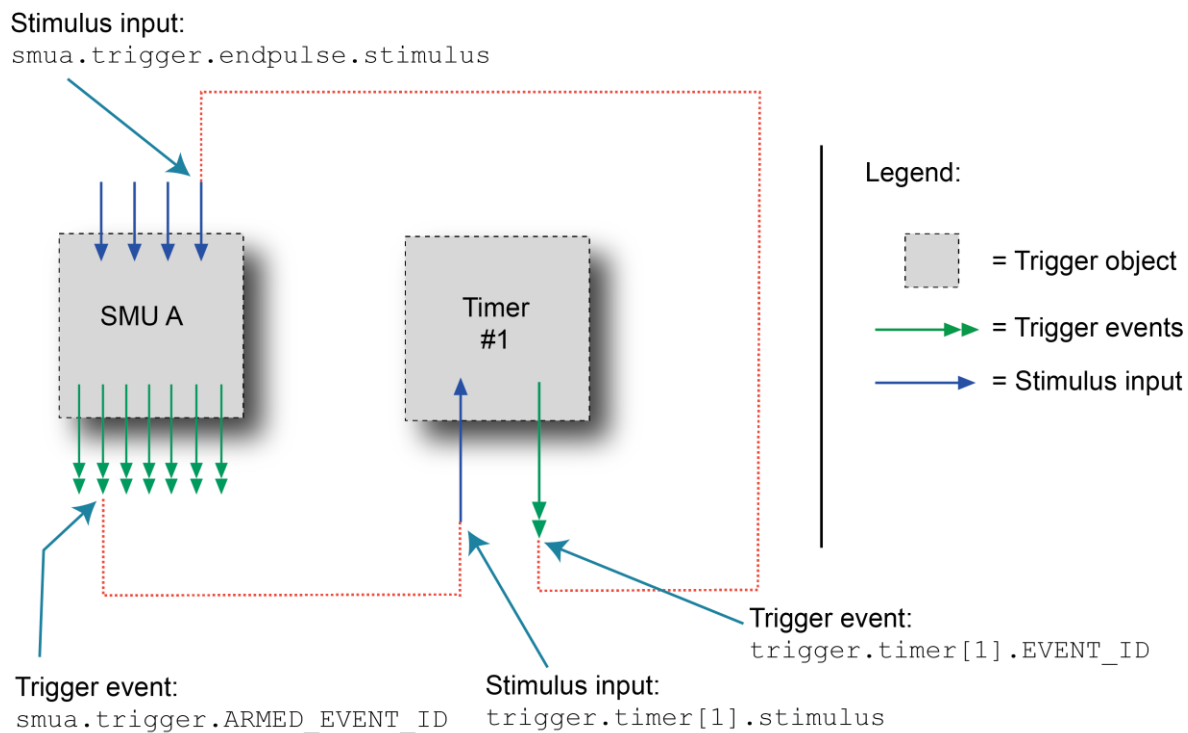
Timer 1: Pulse-width timer

- Set the delay attribute of a timer equal to the appropriate pulse width.
- Configure the timer to trigger when the SMU moves out of the arm layer of the trigger model.
- Assign the trigger event generated by the timer to the stimulus input of the SMU end pulse event detector.

SMU A

- Configure the source action to start immediately by setting the stimulus input of the source event detector to 0.
- Set the end pulse action to `SOURCE_IDLE`.

The following figure shows the trigger setup for this example.

Figure 66: Single-pulse triggering

Single pulse example code

NOTE

Even though no measurements are made in this example, a measure range is set. When sourcing voltage, it is good practice to set the current measure range equal to the triggered source limit range. This is especially important when the triggered limit is greater than 100 mA. If the measure range is not set, it may affect the shape of the pulse. This step is not necessary when sourcing current.

```
-- Reset SourceMeter instrument to default conditions.
reset()

-- Generate a single pulse with the following characteristics:
--      * Bias (idle) level = 0 V
--      * Pulse level = 5 V
--      * Pulse width = 500 us

-- Configure the source function.
smua.source.func = smua.OUTPUT_DCVOLTS

-- Set the voltage source range and the idle or bias source level and limit.
smua.source.rangev = 5
smua.source.levelv = 0
smua.source.limiti = 0.1

-- Configure the trigger-timer parameters to output a single 500 us pulse.
trigger.timer[1].delay = 0.0005
trigger.timer[1].count = 1
trigger.timer[1].passthrough = false
-- Start the timer when the SMU moves from the ARM layer to the TRIGGER layer.
trigger.timer[1].stimulus = smua.trigger.ARMED_EVENT_ID

-- Configure the trigger model to execute a single-point voltage pulse list sweep.
-- No measurements are made.
smua.trigger.source.listv({5})
smua.trigger.source.action = smua.ENABLE
smua.trigger.measure.action = smua.DISABLE
-- Set the trigger source limit to the same value as the bias limit.
smua.trigger.source.limiti = smua.LIMIT_AUTO
smua.measure.rangei = 0.1
-- Configure the source action to start immediately.
smua.trigger.source.stimulus = 0
-- Configure the endpulse action to achieve a pulse.
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
-- Set the appropriate counts for the trigger model.
smua.trigger.arm.count = 1
smua.trigger.count = 1

-- Turn on the SMU output and initiate the trigger model to output a single pulse.
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
-- Wait for the sweep to complete.
waitcomplete()

-- Turn off SMU output.
smua.source.output = smua.OUTPUT_OFF
```

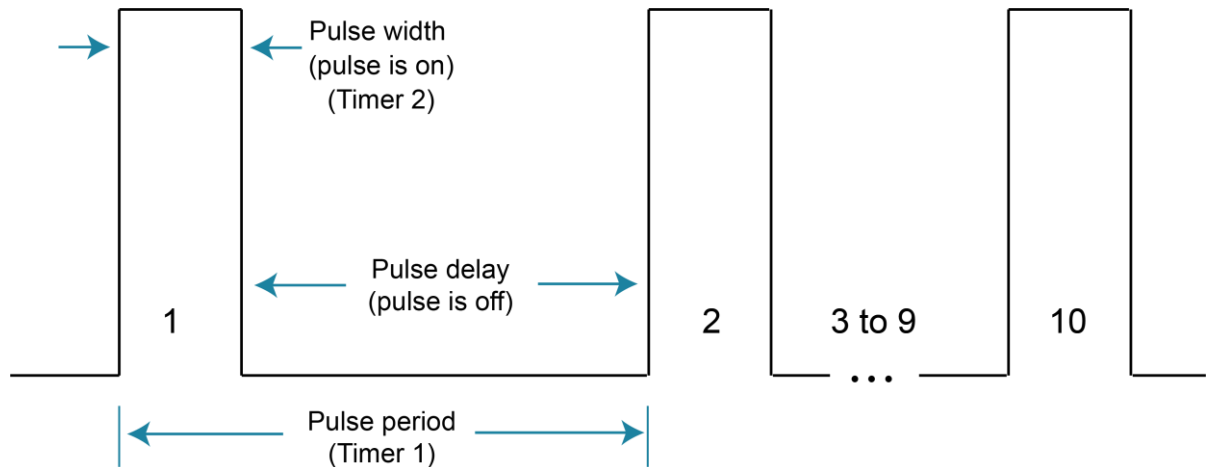
Pulse train example

The SMU programming example below illustrates how to use two timers: One to control the pulse period, a second to control the pulse width. The example configures the timers and SMU as follows:

Timer 1: Pulse period timer

- Set the delay attribute to the appropriate pulse period (see the following figure).
- Configure the timer to start when the sweep is initiated.
- Enable the pass-through attribute so that the timer generates a trigger event at the start of the first delay.
- Set the count equal to one less than the total number of pulses to output.

Figure 67: Pulse train



Timer 2: Pulse width timer

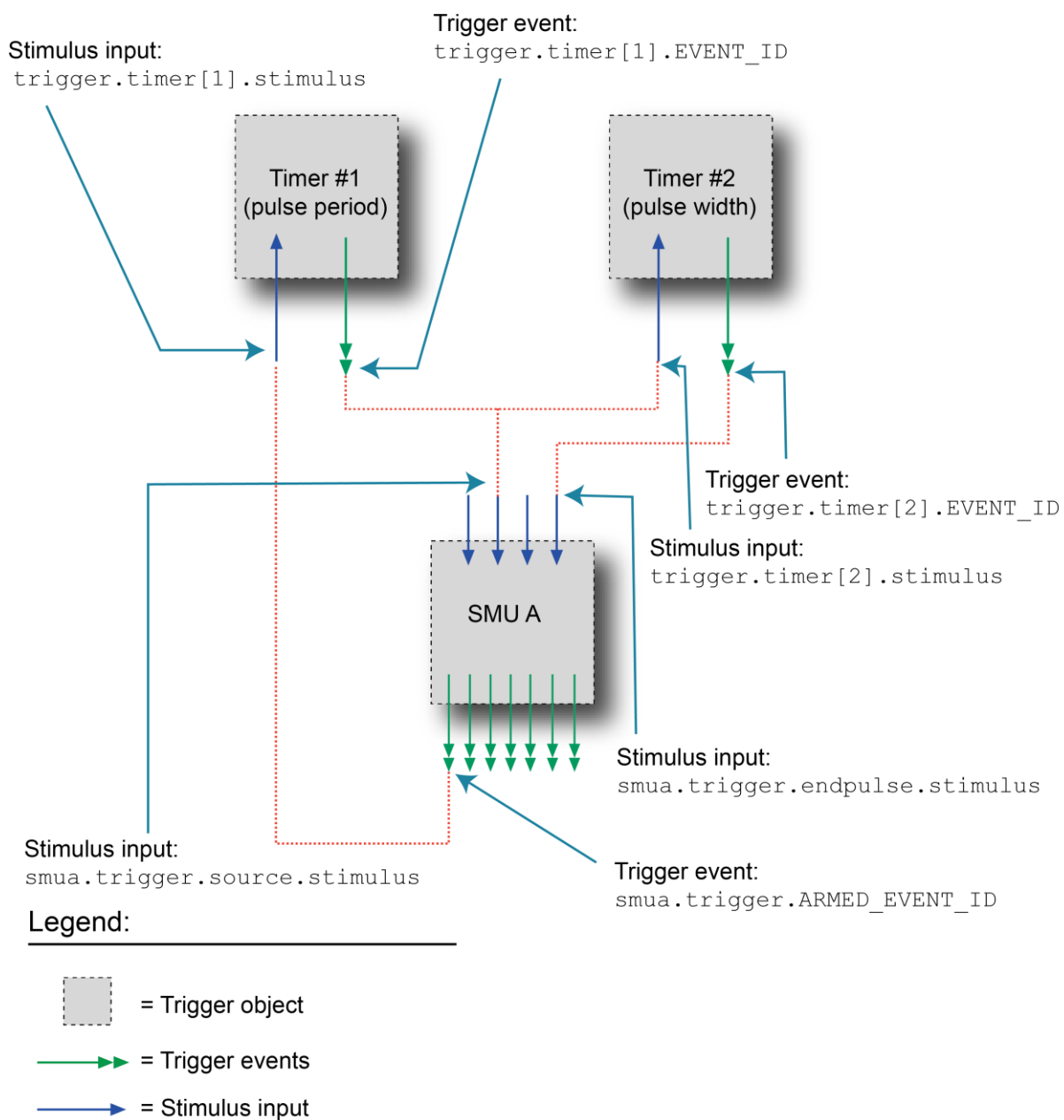
- Set the delay attribute to an appropriate pulse width (see the following figure).
- Set the stimulus input to the event ID of timer 1 (the start of each pulse is the start of the pulse period).
- Set the count equal to 1 so that only one pulse is issued per period.

SMU A

- Set the source stimulus input to the event ID of timer 1 so that the source action starts when the period starts.
- Set the end pulse action to `smua.SOURCE_IDLE` so that the output is returned to the idle level after the pulse completes.
- Set the end pulse stimulus input to the event ID of timer 2 so that the end pulse action executes when the pulse width timer expires.
- Set the trigger count equal to the total number of pulses to output.
- Set the arm count to 1.

The following figure shows the trigger setup for this example.

Figure 68: Pulse train triggering



Pulse train example code

NOTE

Even though no measurements are made in this example, a measure range is set. When sourcing voltage, it is good practice to set the current measure range equal to the triggered source limit range. This is especially important when the triggered limit is greater than 100 mA. If the measure range is not set, it may impact the shape of the first pulse in the train. This step is not necessary when sourcing current.

```
-- Reset the SourceMeter instrument to default conditions.
reset()

-- Generate a 10-point pulse train with the following characteristics:
--      * Bias (Idle) Level = 0 V
--      *      Pulse Level = 5 V
--      *      Pulse Width = 600 us
--      *      Pulse Period = 5 ms

-- Configure the source function.
smua.source.func = smua.OUTPUT_DCVOLTS

-- Set the voltage source range and the bias source level and limit.
smua.source.rangev = 5
smua.source.levelv = 0
smua.source.limiti = 0.1

-- Use trigger timer 1 to control the period and trigger timer 2 to control the
-- pulse width. Alias the timers for convenience and clarity.
period_timer = trigger.timer[1]
pulsewidth_timer = trigger.timer[2]

-- Configure the period timer to output 10 total trigger events.
period_timer.delay = 0.005
-- The effective count is 10 because the passthrough setting is true.
period_timer.count = 9
-- Configure the timer to immediately output a trigger event when it is started.
period_timer.passthrough = true
-- Start the timer when the SMU moves from the ARM layer to the TRIGGER layer.
period_timer.stimulus = smua.trigger.ARMED_EVENT_ID

-- Configure the pulse width timer to output one trigger event for each period.
pulsewidth_timer.delay = 0.0006
pulsewidth_timer.count = 1
-- Do not immediately output a trigger event when pulse width timer is started.
pulsewidth_timer.passthrough = false
-- Start the pulse width timer with the period timer output trigger event.
pulsewidth_timer.stimulus = period_timer.EVENT_ID

-- Configure the trigger model to execute a 10-point fixed-level voltage pulse
-- train. No measurements are made.
smua.trigger.source.listv({5})
smua.trigger.source.action = smua.ENABLE
smua.trigger.measure.action = smua.DISABLE
```

```
-- Set the trigger source limit, which can be different than the bias limit.
-- This is an important setting for pulsing in the extended operating area.
smua.trigger.source.limiti = 1
smua.measure.rangei = 1
-- Trigger SMU source action with the period timer event.
smua.trigger.source.stimulus = period_timer.EVENT_ID
-- Configure the endpulse action to achieve a pulse.
smua.trigger.endpulse.action = smua.SOURCE_IDLE
-- Trigger the SMU end pulse action with a pulse width timer event.
smua.trigger.endpulse.stimulus = pulsewidth_timer.EVENT_ID
-- Set the trigger model count to generate one 10-point pulse train.
smua.trigger.arm.count = 1
smua.trigger.count = 10

-- Turn on the SMU output and initiate the trigger model to output the pulse train.
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
-- Wait for the sweep to complete.
waitcomplete()

-- Turn off SMU output.
smua.source.output = smua.OUTPUT_OFF
```

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 virtual front panel.

You can program up to six event blenders for each module of the 2606B.

Event blender modes

Event blenders perform logical AND and logical OR functions on trigger events. For example, trigger events can be triggered when either a manual trigger or external input trigger is detected.

- **Or:** Generates an event when an event is detected on *any* one of the four stimulus inputs
- **And:** Generates an event when an event is detected on *all* of the assigned stimulus inputs

Set the `trigger.blender[N].orenable` attribute to configure the event blender mode. Setting the attribute to `true` enables OR mode; setting the attribute to `false` enables AND mode.

Assigning input trigger events

Each event blender has four stimulus inputs. You can assign a different trigger event ID to each stimulus input. The programming example below illustrates how to assign the source complete event IDs of SMU A and SMU B to stimulus inputs 1 and 2 of event blender 1:

```
trigger.blender[1].stimulus[1] = smua.SOURCE_COMPLETE_EVENT_ID
trigger.blender[1].stimulus[2] = smub.SOURCE_COMPLETE_EVENT_ID
```

Action overruns

Action overruns are generated by event blenders depending on the mode, as shown in the following table. Use the status model to monitor for the occurrence of action overruns. For details, see [Status model](#) (on page 10-1).

Action overruns

Mode	Action overrun
And	Generates an overrun when a second event on any of its inputs is detected before generating an output event.
Or	Generates an overrun when two events are detected simultaneously.

LAN triggering overview

Triggers can be sent and received over the LAN interface. The 2606B supports LAN extensions for instrumentation (LXI) and has eight LAN triggers that generate and respond to LXI trigger packets.

Understanding hardware value and pseudo line state

LAN triggering is similar to hardware synchronization except that LXI trigger packets are used instead of hardware signals. A bit in the LXI trigger packet called the hardware value simulates the state of a hardware trigger line. The 2606B stores the hardware value of the last LXI trigger packet that was sent or received as the pseudo line state.

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 2606B ignores the hardware value of the packet and generates a trigger event. The 2606B 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.

Changes in the hardware value of consecutive LXI trigger packets are interpreted as edge transitions. Edge transitions generate trigger events. If the hardware value does not change between successive LXI trigger packets, the 2606B assumes an edge transition was missed and generates a trigger event. The following table illustrates 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

Set the LAN trigger mode to configure the edge detection method in incoming LXI trigger packets. The mode that is selected also determines the hardware value in outgoing LXI trigger packets. The following table lists the LAN trigger modes.

LAN trigger modes

Trigger mode	Input detected	Output generated	Notes
Either edge	Either	Negative	
Falling edge	Falling	Negative	
Rising edge	Rising	Positive	
RisingA	Rising	Positive	Same as Rising edge
RisingM	Rising	Positive	Same as Rising edge
Synchronous	Falling	Positive	Same as SynchronousA
SynchronousA	Falling	Positive	
SynchronousM	Rising	Negative	

The programming example below illustrates how to configure the LAN trigger mode.

```
-- Set LAN trigger 2 to falling edge.
lan.trigger[2].mode = lan.TRIG_FALLING
```

Understanding LXI trigger event designations

LAN trigger objects generate LXI trigger events, which are LAN0 to LAN7 (zero based). In the command table, the LXI trigger events can be accessed using `lan.trigger[1]` through `lan.trigger[8]`.

`lan.trigger[1]` corresponds to LXI trigger event LAN0 and `lan.trigger[8]` corresponds to LXI trigger event LAN7.

Generating LXI trigger packets

You can configure the 2606B to output an LXI trigger packet to other LXI instruments.

To generate LXI trigger packets:

1. Call the `lan.trigger[N].connect()` function.
2. Select the event that triggers the outgoing LXI trigger packet by assigning the specific event ID to the LAN stimulus input.

Make sure to use the same LXI domain on both the 2606B instrument and the other instrument. If the 2606B has a different LXI domain than the instrument at the other end of the trigger connection, the LXI trigger packets are ignored by both instruments.

Command interface triggering

A command interface trigger occurs when:

- A VXI-11 `device_trigger` method is invoked (VXI-11 only)
- A `*TRG` message is received
- A USBTMC TRIGGER message is received (USB only)

Use `trigger.EVENT_ID` to monitor for command interface triggers. To ensure that commands and triggers issued over the command interface are processed in the correct order, a trigger event is not generated until:

- The trigger command is executed
- `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 2606B command queue. The 2606B 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

The command queue can fill up with trigger entries if too many `*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` attribute (see [TSP command reference](#) (on page 15-1) for more information), 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 like `abort` are not processed.

Trigger generator

The 2606B has two trigger generators available from each module that you can use to generate trigger events. Use the `trigger.generator[N].assert()` function to directly trigger events from the command interface or a script (for example, you can trigger a sweep while the instrument is under script control).

The `trigger.generator[N].EVENT_ID` constant is an identification number that identifies events generated by this generator. To have another trigger object respond to trigger events generated by this generator, set the `stimulus` attribute of the other object to the value of this constant.

Manual triggering

The TRIG key is used for manual triggering. Each time the TRIG key is pressed, a trigger event is generated. You can monitor for a manual trigger event using the event ID `display.trigger.EVENT_ID`. See [Using the TRIG key to trigger a sweep](#) (on page 7-8) for an example of how to use a manual trigger.

There are no action overruns for manual triggering.

Interactive triggering

The complexity of some test system configurations may not allow a static trigger setup. These configurations require more dynamic control of triggering than the static trigger setup provides. For such cases, a setup providing interactive trigger programming allows the generation and detection of trigger events that can be controlled on demand under remote control. For example, you can use interactive triggering when you need to make multiple source function changes or implement conditional branching to other test setups based on recent measurements.

Detecting trigger events using the wait() function

Most of the 2606B trigger objects, except for SMU trigger objects, have built-in event detectors that monitor for trigger events. The event detector only monitors events generated by that object and cannot be configured to monitor events generated by any other trigger object. Using the `wait()` function of the trigger object causes the 2606B instrument to suspend command execution until a trigger event occurs or until the specified timeout period elapses.

For example, use `trigger.blender[N].wait(Y)` to suspend command execution until an event blender generates an event, where *N* is the specific event blender and *Y* 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 10 seconds for a front-panel TRIG key press.
display.trigger.wait(10)
-- 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 10.
digio.trigger[10].wait(30)
```

Using the assert function to generate output triggers

You can use certain trigger objects to generate output triggers on demand. These trigger objects are the digital I/O lines, TSP-Link synchronization lines, and the LAN.

The programming example below illustrates how to generate an output trigger using the assert function of the trigger object.

NOTE

Connection parameters and commands that establish a connection are not shown in this example.

```
-- Generate a falling-edge trigger on digital I/O line 3.
digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[3].assert()
-- Generate a rising edge trigger on TSP-Link sync line 1.
tsplink.trigger[1].mode = tsplink.TRIG_RISINGM
tsplink.trigger[1].assert()
-- Generate a LAN trigger on LAN pseudo line 6.
lan.trigger[6].mode = lan.TRIG_EITHER
lan.trigger[6].assert()
```

Using the release function of the hardware lines

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.trigger[1].mode = digio.TRIG_FALLING
digio.trigger[1].pulsewidth = 0
digio.trigger[1].assert()
-- Release digio line 1.
digio.trigger[1].release()
-- Output another external trigger.
digio.trigger[1].assert()
```

For information about hardware lines, see [Digital I/O port and TSP-Link synchronization lines](#) (on page 7-11).

Using the set function to bypass SMU event detectors

The set functions are useful whenever you want the source-measure unit (SMU) to continue operation without waiting for a programmed trigger event.

There is a set function for each SMU event detector. When called, the function immediately satisfies the event detector, allowing the SMU to continue through the trigger model.

For example, you can use a set function when you want the SMU to immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur. You can use a set function to start actions on the SMU if there is a missed trigger event.

The programming example below illustrates how to have the SMU immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur.

```
-- Immediately sets the arm event detector of SMU A
-- to the detected state.
smua.trigger.arm.set()
-- Sets the measure event detector of SMU A.
smua.trigger.measure.set()
```

Event detector overruns

If a second trigger event is generated before an event detector clears, the trigger object generates a detector overrun. You can check for detector overruns by reading the `overrun` attribute of the trigger object. The attribute is set to `true` when an overrun occurs. You can use the `clear()` function to immediately clear the event detector, discarding any history of previous trigger events. The `clear()` function also clears any detector overruns.

NOTE

Detector overruns are not the same as the action overruns that are reported in the status model.

The programming example below illustrates how to check for and respond to detector overruns.

```
testOver = digio.trigger[4].overrun
if testOver == true then
    print("Digital I/O overrun occurred.")
end
```

Examples using interactive triggering

The following examples demonstrate how to use interactive triggering.

Command interface interactive trigger example

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 2606B performs a voltage reading.

```
-- Clear any previously detected command interface triggers.
trigger.clear()
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Wait 30 seconds for a command interface trigger.
triggered = trigger.wait(30)
-- Get a voltage reading.
reading = smua.measure.v()
-- Send a command interface trigger to trigger the measurement.
*TRG
```

NOTE

*TRG cannot be used in a script.

Manual triggering example

The programming example below illustrates how to pause a script and prompt the operator to press the TRIG key when the operator is ready to continue. If the TRIG key is not pressed, the test continues after waiting 10 minutes (600 seconds).

```
display.clear()
display.trigger.clear()
display.setCursor(1, 1)
display.setText("Take a Break")
display.setCursor(2, 1)
display.setText("Press TRIG to continue")
display.trigger.wait(600)
display.clear()
```

Digital I/O triggering interactive example

The programming example below illustrates how to configure digital I/O line 2 as an input trigger and digital I/O line 14 as an output trigger. The 2606B waits for an external input trigger on digital I/O line 2. If a trigger event occurs, the 2606B outputs an external trigger on digital I/O line 14. If no trigger event is received on digital I/O line 2, the test is aborted.

```
-- Configure digital I/O lines 2 and 14 for input trigger detection
-- and output trigger generation, respectively.
digio.trigger[2].mode = digio.TRIG_RISINGA
digio.trigger[2].clear()
digio.trigger[14].mode = digio.TRIG_FALLING
digio.trigger[14].pulsewidth = 0.0001
-- Wait 15 seconds for a trigger event to occur on digital I/O line 2.
trigInput = digio.trigger[2].wait(15)
-- If a trigger event occurs on digital I/O line 2, assert an output
-- trigger on digital I/O line 14. If a trigger event does
-- not occur, turn off the output of smua and issue a message
-- on the virtual front-panel display.
if trigInput == true then
    digio.trigger[14].assert()
else
    smua.source.output = smua.OUTPUT_OFF
    display.screen = display.USER
    display.clear()
    display.setcursor(1, 1)
    display.settext("No trigger received. Test aborted.")
    exit()
end
```

Hardware trigger modes

You can use different hardware trigger modes for digital I/O and TSP-Link® synchronization. Use hardware triggers to integrate Keithley instruments and non-Keithley instruments in a test system. The 2606B supports 14 digital I/O lines and three TSP-Link synchronization lines that can be used for input or output triggering. For additional information about the hardware trigger modes, see [TSP command reference](#) (on page 15-1).

NOTE

For direct control of the line state, use the bypass trigger mode.

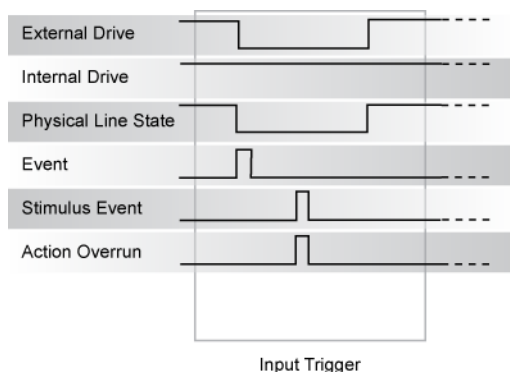
Falling edge trigger mode

The falling edge trigger mode generates low pulses and detects all falling edges. The figure titled "Falling edge input trigger" shows the characteristics of the falling edge input trigger; the figure titled "Falling edge output trigger" shows the falling edge output trigger.

Input characteristics:

- Detects all falling edges as input triggers.

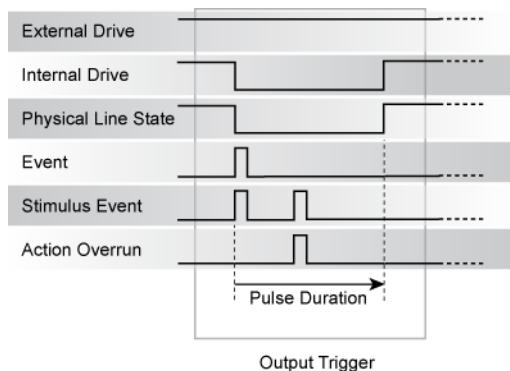
Figure 69: Falling edge input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse for the programmed pulse duration.
- An action overrun occurs if the physical line state is low and a source event occurs.

Figure 70: Falling edge output trigger



Rising edge master trigger mode

Use the rising edge master (RisingM) trigger mode (see the figure titled "RisingM output trigger") to synchronize with non-Keithley instruments that require a high pulse. Input trigger detection is not available in this trigger mode. You can use the RisingM trigger mode to generate rising edge pulses.

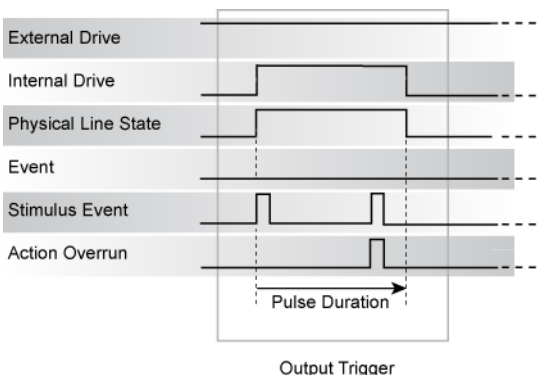
NOTE

The RisingM trigger mode does not function properly if the line is driven low by an external drive.

Output characteristics:

- Configured trigger events, as well as the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands, cause the physical line state to float high during the trigger pulse duration.
- An action overrun occurs if the physical line state is high when a stimulus event occurs.

Figure 71: RisingM output trigger



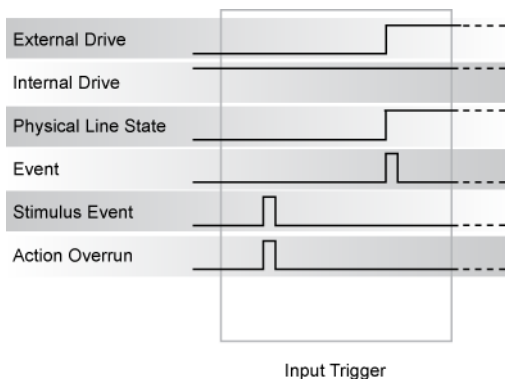
Rising edge acceptor trigger mode

The rising edge acceptor trigger mode (RisingA) generates a low pulse and detects rising edge pulses. Refer to the following figures.

Input characteristics:

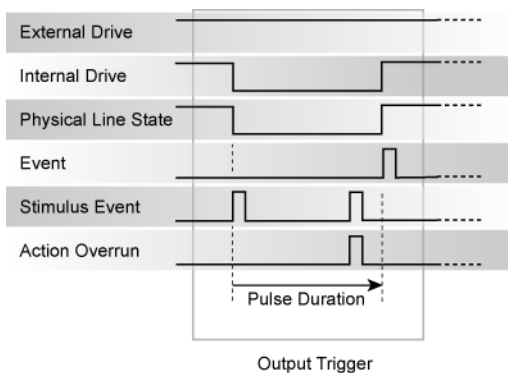
- All rising edges generate an input event.

Figure 72: RisingA input trigger



Output characteristics:

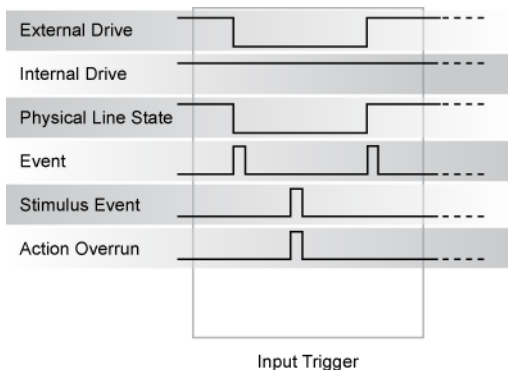
- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse that is similar to the falling edge trigger mode.

Figure 73: RisingA output trigger**Either edge trigger mode**

The either edge trigger mode generates a low pulse and detects both rising and falling edges.

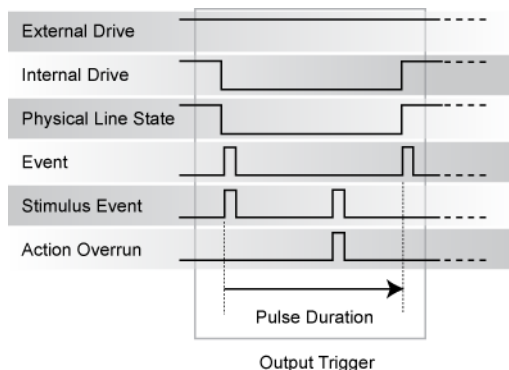
Input characteristics:

- All rising or falling edges generate an input trigger event.

Figure 74: Either edge input trigger

Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse that is similar to the falling edge trigger mode.
- An action overrun occurs if the physical line state is low while a stimulus event occurs.

Figure 75: Either edge output trigger

Understanding synchronous triggering modes

Use the synchronous triggering modes to implement bidirectional triggering, to wait for one node, or to wait for a collection of nodes to complete all triggered actions.

All non-Keithley instrumentation must have a trigger mode that functions similar to the SynchronousA or SynchronousM trigger modes.

To use synchronous triggering, configure the triggering master to SynchronousM trigger mode or the non-Keithley equivalent. Configure all other nodes in the test system to SynchronousA trigger mode or a non-Keithley equivalent.

Synchronous master trigger mode (SynchronousM)

Use the synchronous master trigger mode to generate falling edge output triggers, to detect the rising edge input triggers, and to initiate an action on one or more external nodes with the same trigger line.

In this mode, the output trigger consists of a low pulse. All non-Keithley instruments attached to the synchronization line in a trigger mode equivalent to SynchronousA must latch the line low during the pulse duration.

To use the SynchronousM trigger mode, configure the triggering master as SynchronousM and then configure all other nodes in the test system as Synchronous, SynchronousA, or to the non-Keithley Instruments equivalent.

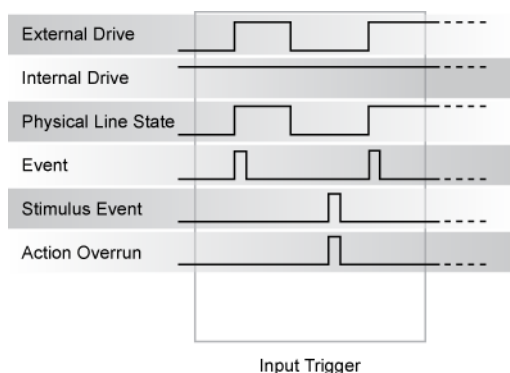
NOTE

Use the SynchronousM trigger mode to receive notification when the triggered action on all nodes is complete.

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.

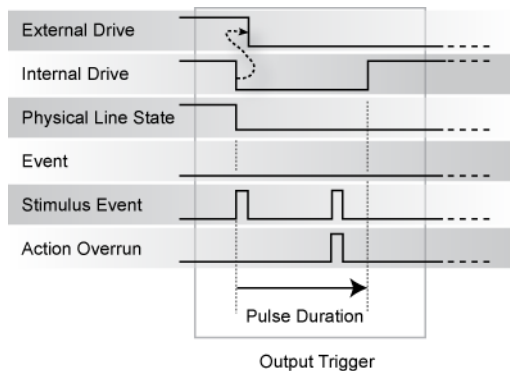
Figure 76: Synchronous master input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions 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 77: Synchronous master output trigger



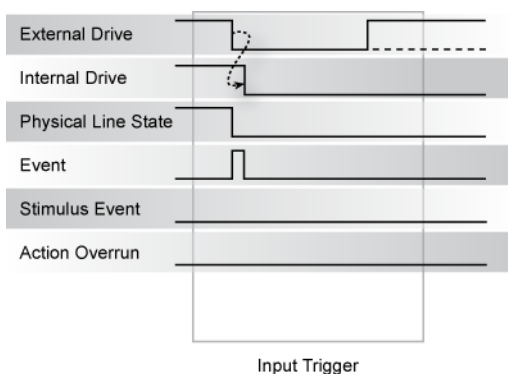
Synchronous acceptor trigger mode (SynchronousA)

Use the synchronous acceptor trigger mode (SynchronousA) on a trigger subordinate that operates with a trigger master configured for the SynchronousM trigger mode. The roles of the internal and external drives are reversed in the SynchronousA trigger mode.

Input characteristics:

- The falling edge is detected as the external drive pulses the line low, and the internal drive latches the line low.

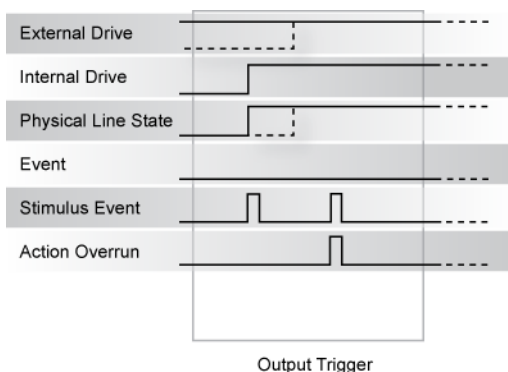
Figure 78: Synchronous acceptor input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions release the line if the line is latched low. The pulse width is not used.
- 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 79: Synchronous acceptor output trigger



Synchronous trigger mode

The synchronous trigger mode is a combination of SynchronousA and SynchronousM trigger modes. Use the Synchronous trigger mode for compatibility with older Keithley Instruments products.

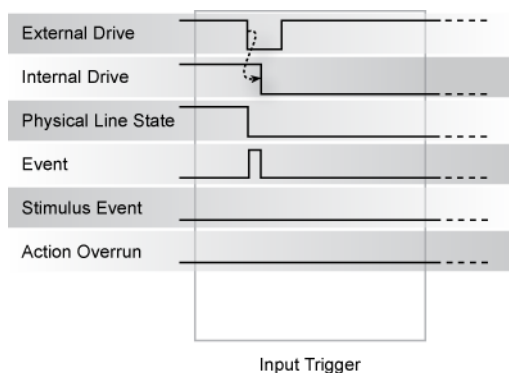
NOTE

Keithley Instruments recommends using SynchronousA and SynchronousM modes only.

Input characteristics:

- The falling edge generates an input event and latches the internal drive low.

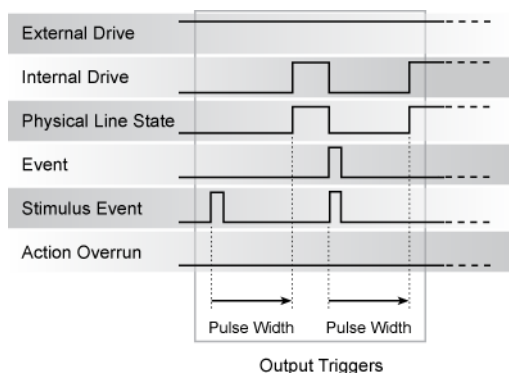
Figure 80: Synchronous input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions generate a low pulse for the programmed pulse duration if the line is latched low; a falling edge does not occur.
- A normal falling edge pulse generates when the internal drive is not latched low and the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions are issued.
- To mirror the SynchronousA trigger mode, set the pulse width to 1 μ s or any small nonzero value.
- Action overruns are disabled.

Figure 81: Synchronous output trigger



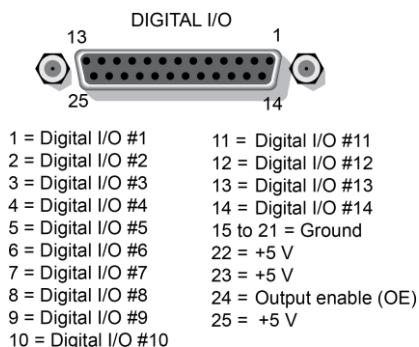
Digital I/O

The 2606B has a digital input/output port on each module that can be used to control external digital circuitry. For example, you can use a handler that performs binning operations with a digital I/O port.

Port configuration

The digital I/O port, a standard DB-25 socket connector (shown below), is on the front panel.

Figure 82: Digital I/O port



NOTE

For a schematic diagram of the digital I/O hardware, refer to the *Model 2606B Specifications* on the [Keithley Instruments support website](http://tek.com/support) (tek.com/support).

The port provides 14 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.

The +5 V dc output can be used to drive external logic circuitry. Maximum combined current output for all lines is 250 mA. These lines are protected by a self-resetting fuse with an hour recovery time.

The output enable (OE) line of the digital I/O can be used with a switch in the test fixture or component handler. With proper use, power is removed from the DUT when the lid of the fixture is opened. See [Using output enable](#) (on page 7-40) for more details.

⚠ WARNING

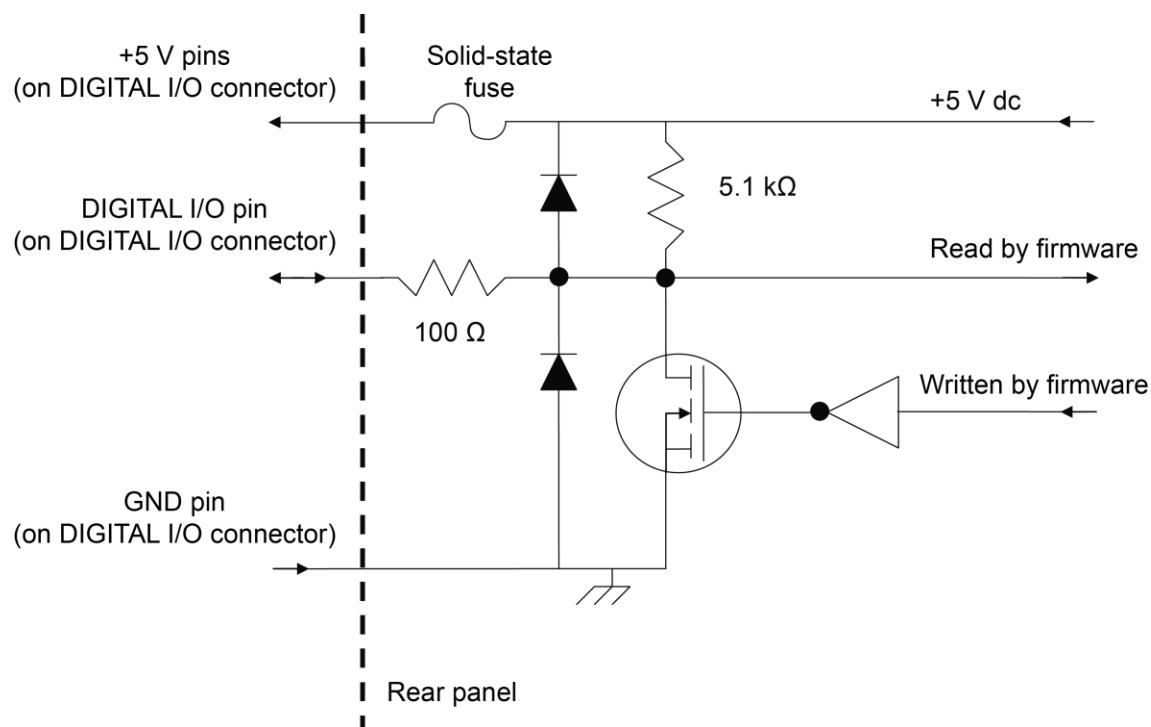
The digital I/O port of the 2606B is not suitable for control of safety circuits and should not be used to control a safety interlock. When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages.

If you are using other Keithley Instrument models that are equipment with a Trigger Link (TLINK), use a cable equipped with a male DB-25 connector (L-com part number CSMN25MF-5) to connect the digital I/O port to the other equipment.

Digital I/O configuration

The following figure shows the basic configuration of the digital I/O port. Writing a 1 to a line sets that line high ($\sim +5$ V). Writing a 0 to a line sets that line low (~ 0 V). Note that an external device pulls an I/O line low by shorting it to ground, so that a device must be able to sink at least $960\text{ }\mu\text{A}$ per I/O line.

Figure 83: Digital I/O interface schematic



Controlling digital I/O lines

Although the digital I/O lines are primarily intended for use with a device handler for limit testing, they can also be used for other purposes, such as controlling external logic circuits. You can control lines either from the virtual front panel or over a remote interface.

To set digital I/O values from the virtual front panel:

1. Press the **MENU** key, select **DIGOUT**, and then press the **ENTER** key or press the navigation wheel.
2. Select **DIG-IO-OUTPUT**, and then press the **ENTER** key or the navigation wheel.
Set the decimal value as required to set digital I/O lines in the range of 0 to 16,383 (see the table in [Digital I/O bit weighting](#) (on page 7-39)), and then press the **ENTER** key or the navigation wheel.
For example, to set digital I/O lines 3 and 8, set the value to 132.
3. Press the **EXIT (LOCAL)** key as needed to return to the main menu.

To write-protect specific digital I/O lines to prevent their values from being changed:

1. Press the **MENU** key, then select **DIGOUT** and then press the **ENTER** key or the navigation wheel.
2. Select **WRITE-PROTECT**, and then press the **ENTER** key or the navigation wheel.
3. Set the decimal value as required to write-protect digital I/O lines within the range of 0 to 16,383 (see [Digital I/O bit weighting](#) (on page 7-39)), and then press the **ENTER** key or the navigation wheel. For example, to write-protect digital I/O lines 4 and 10, set the value to 520.
4. Press the **EXIT (LOCAL)** key as needed to return to the main menu.

To remove write protection, reset the decimal value to include only the lines that you want to write protect. To remove write protection from all lines, set the value to 0.

Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in the following table.

Digital bit weight

Line #	Bit	Decimal weighting	Hexadecimal weighting
1	B1	1	0x0001
2	B2	2	0x0002
3	B3	4	0x0004
4	B4	8	0x0008
5	B5	16	0x0010
6	B6	32	0x0020
7	B7	64	0x0040
8	B8	128	0x0080
9	B9	256	0x0100
10	B10	512	0x0200
11	B11	1,024	0x0400
12	B12	2,048	0x0800
13	B13	4,096	0x1000
14	B14	8,192	0x2000

Remote digital I/O commands

Commands that control and access the digital I/O port are summarized in the following table. See the [TSP command reference](#) (on page 15-1) for complete details on these commands. See [Digital I/O bit weighting](#) (on page 7-39) for decimal and hexadecimal values used to control and access the digital I/O port and individual lines. Use these commands to trigger the 2606B using external trigger pulses applied to the digital I/O port, or to provide trigger pulses to external devices.

Use these commands to 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.

NOTE

You can use the digital I/O lines for both input and output. You must write a 1 to all digital I/O lines that are to be used as inputs.

Remote digital I/O commands

Command	Description
<code>digio.readbit(<i>bit</i>)</code>	Read one digital I/O input line
<code>digio.readport()</code>	Read digital I/O port
<code>digio.writebit(<i>bit</i>, <i>data</i>)</code>	Write data to one digital I/O output line
<code>digio.writeport(<i>data</i>)</code>	Write data to digital I/O port
<code>digio.writeprotect = <i>mask</i></code>	Write protect mask to digital I/O port

Digital I/O programming example

The programming commands below illustrate how to set bit B1 of the digital I/O port high, and then read the entire port value.

```
digio.trigger[1].mode = digio.TRIG_BYPASS
-- Set bit B1 high.
digio.writebit(1,1)
-- Read digital I/O port.
data = digio.readport()
```

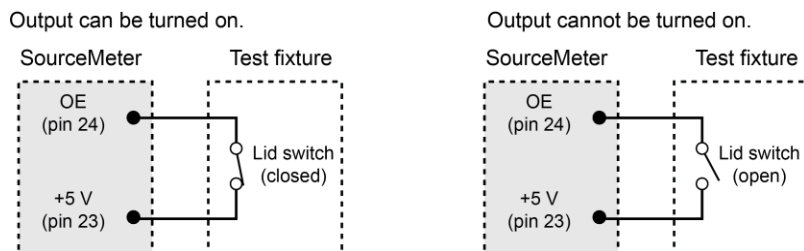
Using output enable

Each module of the 2606B includes a digital I/O port that provides an output enable line for use with a test fixture switch. When properly used, the output of the module turns OFF when the lid of the test fixture is opened. See [DUT Test Connections](#) (on page 4-3) for important safety information when using a test fixture.

WARNING

When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages. The digital I/O port of the 2606B is not suitable for control of safety circuits and should not be used to control a safety interlock.

When enabled, you can only turn on the output of the module when the output enable line is pulled high through a switch to +5 V (as shown). If the lid of the test fixture opens, the switch opens, and the output enable line goes low, turning the output of the 2606B off. The output will not automatically turn on when output enable is set high. The output cannot be turned back on until +5 V is applied to the output enable line.

Figure 84: Using the output enable line

Front-panel control of output enable

To activate the output enable line from the virtual front panel:

1. Press the **CONFIG** key followed by the **OUTPUT ON/OFF** control.
2. Choose **DIO-CONTROL**, then press the **ENTER** key or the navigation wheel.
3. To activate the output enable signal, select **OE_OUTPUT_OFF**. This causes the source-measure unit (SMU) output to be blocked if the output enable is not asserted (connected to +5 V).
To deactivate the output enable signal, select **NONE**. The state of the output enable signal has no effect on the SMU output.
4. Press the **EXIT (LOCAL)** key as needed to return to the normal display.

Remote control of output enable

Use one of these commands to control output enable action:

```
smua.source.outputenableaction = smua.OE_NONE
smub.source.outputenableaction = smub.OE_NONE
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

When set to `smuX.OE_NONE`, the 2606B does not take action when the output enable line is low.

When set to `smuX.OE_OUTPUT_OFF`, the instrument turns its output off as if the `smuX.source.output = smuX.OUTPUT_OFF` command was received. The instrument does not automatically turn its output on when the output enable line returns to the high state. For example, the following command activates the output enable for SMU A:

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

TSP-Link trigger lines

The 2606B has three trigger lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

The TSP-Link® trigger lines are built into the TSP-Link connection. Use the TSP-Link connectors on the front of the 2606B. If you are using a TSP-Link network, you do not have to modify any connections. See [TSP-Link system expansion interface](#) (on page 14-52) for detailed information about connecting to the TSP-Link system.

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 7-39) topic for digital bit weight values.

Remote TSP-Link trigger line commands

Commands that control and access the TSP-Link® trigger line port are summarized in the following table. See the [TSP command reference](#) (on page 15-1) for complete details on these commands. See the table in [Digital I/O bit weighting](#) (on page 7-39) for the decimal and hexadecimal values used to control and access the digital I/O port and individual lines.

Use the commands in following table to perform basic steady-state digital I/O operations; for example, you can program the 2606B to read and write to a specific TSP-Link trigger line or to the entire port.

NOTE

The TSP-Link trigger lines can be used for both input and output. You must write a 1 to all TSP-Link trigger lines that are used as inputs.

Remote trigger line commands

Command	Description
<code>tsplink.readbit(<i>bit</i>)</code>	Reads one digital I/O input line.
<code>tsplink.readport()</code>	Reads the digital I/O port.
<code>tsplink.writebit(<i>bit</i>, <i>data</i>)</code>	Writes data to one digital I/O line.
<code>tsplink.writeport(<i>data</i>)</code>	Writes data to the digital I/O port.
<code>tsplink.writeprotect = <i>mask</i></code>	Sets the write-protect mask of the digital I/O port.

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:

```
-- Set the TSP-Link trigger line to the trigger bypass mode.
tsplink.trigger[1].mode = tsplink.TRIG_BYPASS
-- Set bit B1 high.
tsplink.writebit(1, 1)
-- Read I/O port.
data = tsplink.readport()
```

Theory of operation

In this section:

Source-measure concepts	8-1
Analog-to-digital converter	8-14
Measurement settling time considerations	8-15
Effects of load on current source settling time.....	8-16
Creating pulses with the 2606B SMU.....	8-17

Source-measure concepts

This section provides detailed information about source-measure concepts, including:

- [Limit principles](#) (on page 8-1)
- [Overheating protection](#) (on page 8-2)
- [Operating boundaries](#) (on page 8-3)
- [Basic circuit configurations](#) (on page 5-5)
- [Guard](#) (on page 8-12)

Limit principles

A limit acts as a clamp. If the output reaches the limit value, the 2606B attempts to prevent the output from exceeding that value. This action switches the source from a voltage source to a current source (or from a current source to a voltage source) when a limit is reached.

As an example, assume the following:

- 2606B instrument: $V_{\text{SRC}} = 10 \text{ V}$; $I_{\text{LIMIT}} = 10 \text{ mA}$
- Device-under-test (DUT) resistance: 10Ω

With a source voltage of 10 V and a DUT resistance of 10Ω , the current through the DUT should be $10 \text{ V} / 10 \Omega = 1 \text{ A}$. However, because the limit is set to 10 mA , the current does not exceed that value, 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.

The 2606B SMU output does not exceed the compliance limit, except for the compliance limit conditions described in [Source-measure capabilities](#) (on page 5-2).

Overheating protection

Proper ventilation is required to keep the 2606B from overheating. Even with proper ventilation, the 2606B can overheat if the ambient temperature is too high. The 2606B has an overtemperature protection circuit that turns the output off if the instrument overheats. When the overtemperature protection circuit turns the output off, a message that indicates this condition is displayed. You cannot turn the output on until the instrument cools down.

Power equations to avoid overheating

To avoid overheating, do not operate any channel on the instrument in a manner that forces the instrument to exceed the allowable power dissipation, which is computed using the general power equation, shown below. Factors such as the ambient temperature, voltage and current of the instrument channel, and operating ranges affect the instrument temperature. Exceeding the allowable de-rated power dissipation may cause the temperature protection mechanism to engage.

You do not have to be concerned about overheating if the ambient temperature is $\leq 23\text{ }^{\circ}\text{C}$ and you are not using the 3 A range. However, if the temperature is above this level or the 3 A range is in use, the instrument may overheat if operated in a manner that exceeds the allowable de-rated power dissipation for the ambient temperature.

The left side of the general power equation describes how much power an instrument channel is generating. The right side describes the cooling capacity of the instrument (which is de-rated at temperatures above $23\text{ }^{\circ}\text{C}$). This equation incorporates all of the factors that can influence the power dissipated by the instrument. It becomes possible for the instrument to overheat when the power produced is not less than the power dissipated by the cooling system. The larger the left side is compared to the right side, the faster the instrument will overheat.

$$|(V_{OA} - V_P)(I_P)| \text{Duty Cycle} + |(V_{OA} - V_B)(I_B)| \cdot (1 - \text{Duty Cycle}) \leq \left(54 - P_{DER} - \frac{P_{OC}}{10}\right)$$

Where:

V_{OA}	The instrument output amplifier voltage. This constant is 18 for the 100 mV, 1 V, and 6 V ranges. It is 40 for the 20 V range.
V_P	The voltage level the instrument is attempting to force while at the pulse level. When operating in quadrants I or III (sourcing power), the sign of this voltage must be positive when used in the power equations.
I_P	The current flowing through the instrument channel while at the pulse level.
V_B	The voltage level the instrument is attempting to force while at the bias level. When operating in quadrants I or III (sourcing power), the sign of this voltage must be positive when used in the power equations.
I_B	The current flowing through the instrument channel while at the bias level.
T_{AMB}	The ambient temperature of the instrument operating environment.
P_{DER}	$= T_{AMB} - 23$ This factor represents the number of watts the instrument is derated when operating in environments above $23\text{ }^{\circ}\text{C}$. The maximum output power of each instrument channel is reduced by 1 W for each degree C above $23\text{ }^{\circ}\text{C}$. P_{DER} is 0 when the ambient temperature is below $23\text{ }^{\circ}\text{C}$.
P_{OC}	$=$ Power dissipated by other channels in the unit. This is the sum of the left side of the power equation for each of the other three channels in the unit.

- Duty cycle The value from 0 to 1 that represents the amount of time spent at pulse level versus bias level:
- $$\frac{\text{Pulse Time}}{(\text{Pulse Time} + \text{Bias Time})}$$
- For dc operation where the off state is 0 V or 0 A, Duty Cycle = 1.
- I_P and V_P The voltage and current levels when the instrument is on.

Operating boundaries

The 2606B is specified for operation in Quadrants I and III. When operating in Quadrant I or III, the instrument is operating as a source (voltage and current have the same polarity).

Depending on how the instrument is programmed and what is connected to the output (load or source), you can set the instrument to operate as a sink. When operating in the second (II) or fourth (IV) quadrant, the instrument is operating as a sink (voltage and current have opposite polarity). As a sink, it is dissipating power rather than sourcing it. An external source or an energy storage device, such as a capacitor, battery, or another SMU, can force operation in the sink region.

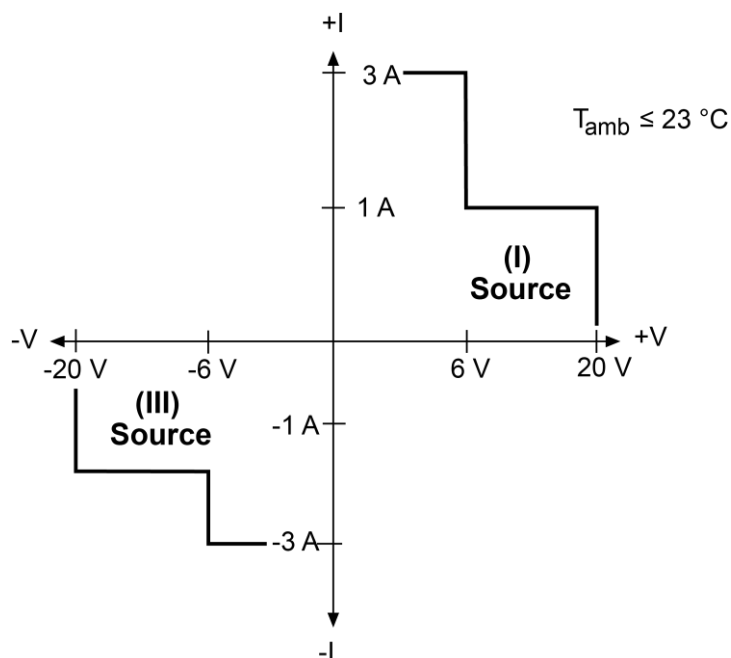
CAUTION

The 2606B is not designed to be operated as a sink instrument. You should avoid situations that cause the instrument to operate as a sink. Operation in Quadrant II or IV can result in overheating and possible equipment damage.

Refer to the specifications, available on tek.com/keithley, for more information on the quadrants of operation.

Continuous power operating boundaries

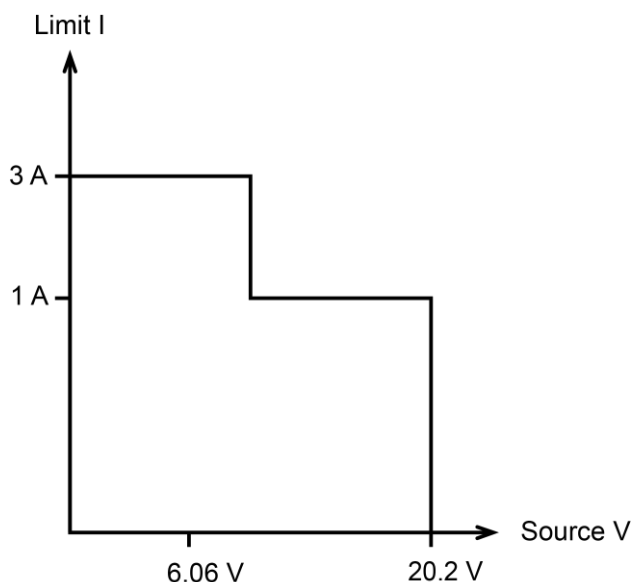
The general operating boundaries for the 2606B continuous power output are shown in the following figure (for derating factors, see [Power equations to avoid overheating](#) (on page 8-2)). In this drawing, the current (1 A and 3 A) and the voltage (6 V and 20 V) magnitudes are nominal values. Also note that the boundaries are not drawn to scale.

Figure 85: 2606B continuous power operating boundaries

V-source operating boundaries

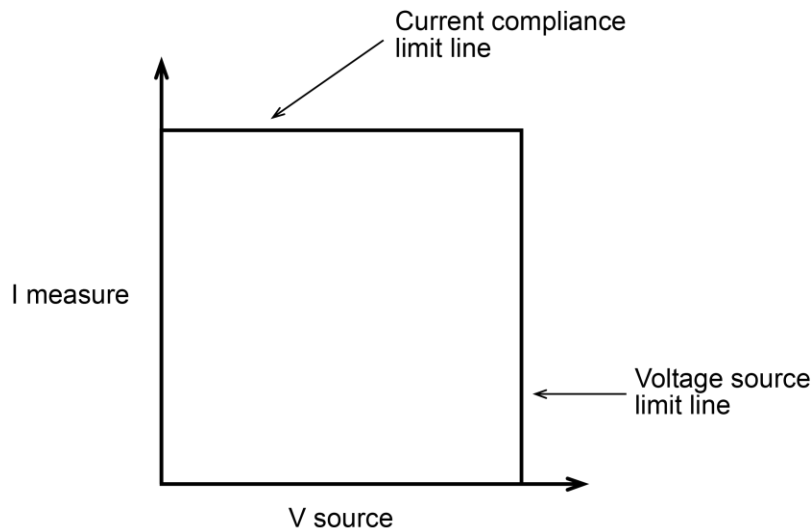
The following figures show the operating boundaries for the voltage source. Only the first quadrant of operation is shown. Operation in the other quadrant is similar.

The following figure shows the output characteristics for the voltage source. The 2606B can output up to 6.06 V at 3 A, or 20.2 V at 1 mA. Note that when sourcing more than 6.06 V, current is limited to 1 A.

Figure 86: 2606B output characteristics

The following figure shows the limit lines for the voltage source. The voltage source limit line represents the maximum source value possible for the presently selected voltage source range. For example, if you are using the 20 V source range, the voltage source limit line is at 20.2 V. The current compliance limit line represents the actual limit in effect. These limit lines are boundaries that represent the operating limits of the 2606B for this quadrant of operation. The operating point can be anywhere inside or on these limit lines. The limit line boundaries for the other quadrant is similar.

Figure 87: 2606B limit lines



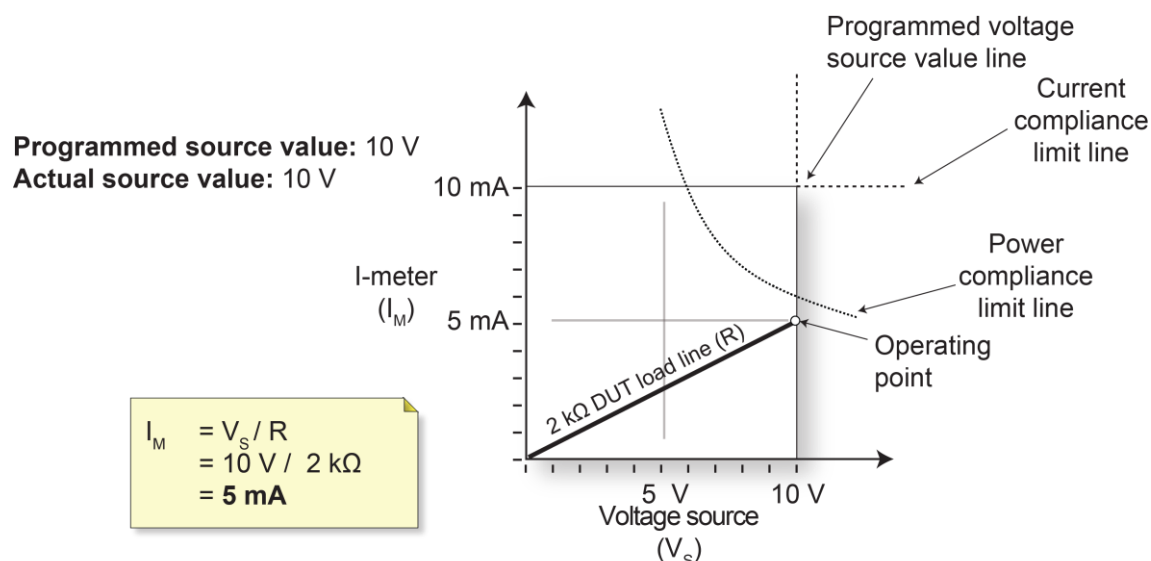
Load considerations for voltage source

The boundaries within which the 2606B operates depend on the load of the device-under-test (DUT) that is connected to the output. The following topics show operation examples for resistive loads that are 2 k Ω and 800 Ω , respectively. For these examples, the 2606B is programmed to source 10 V and limit current (10 mA). In addition, the 2606B is programmed to limit power (60 mW) for the [Normal voltage source operation](#) (on page 8-5) example and the [Voltage source operation in power compliance](#) (on page 8-7) example.

Normal voltage source operation

In the following figure, the 2606B is sourcing 10 V to the 2 k Ω load and subsequently measures 5 mA. The instrument is programmed to limit power (60 mW). As shown, the load line for 2 k Ω intersects the 10 V voltage source line at 5 mA. The current compliance limit and the power compliance limit are not reached, so the instrument is not limited through its compliance settings.

Figure 88: Normal voltage source operation

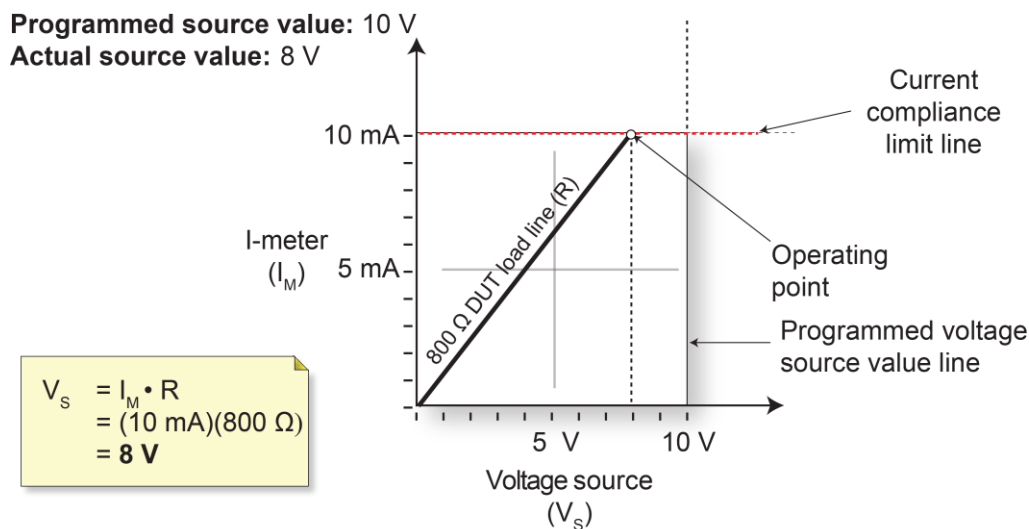


Voltage source operation in current compliance

In the following figure, the resistance of the load is decreased to 800 Ω . The DUT load line for 800 Ω intersects the current compliance limit line, placing the 2606B in compliance. When it is in compliance, the 2606B cannot source its programmed voltage (10 V). For the 800 Ω DUT, the 2606B only outputs 8 V (at the 10 mA limit).

Notice that as resistance decreases, the slope of the DUT load line increases. At zero resistance (shorted output), the 2606B sources virtually 0 V at 10 mA. Conversely, as resistance increases, the slope of the DUT load line decreases. As resistance approaches infinity (open output), the 2606B sources virtually 10 V at 0 mA.

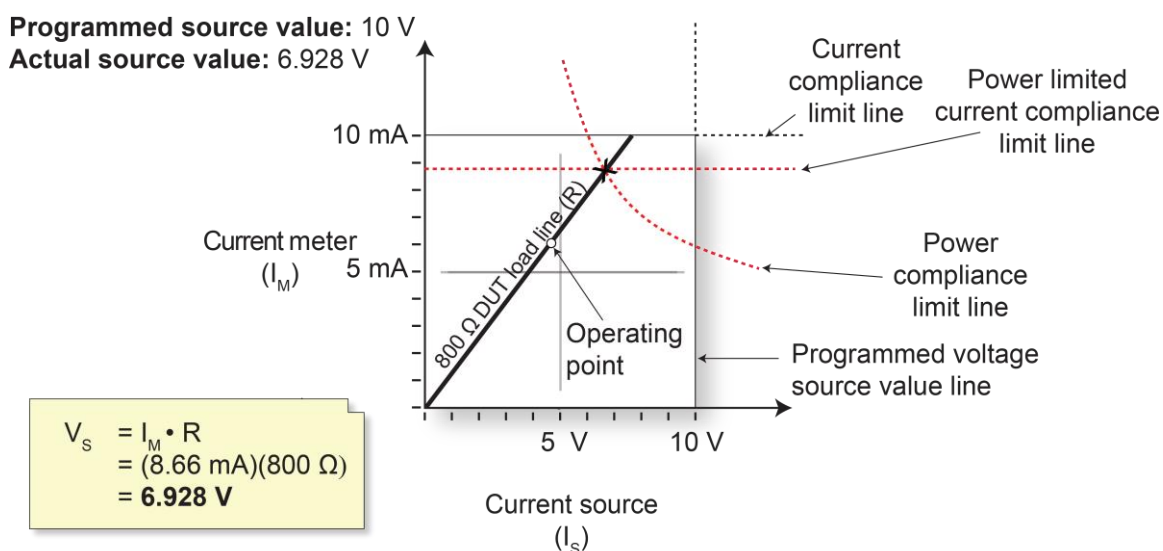
Figure 89: Voltage source operation in current compliance



Voltage source operation in power compliance

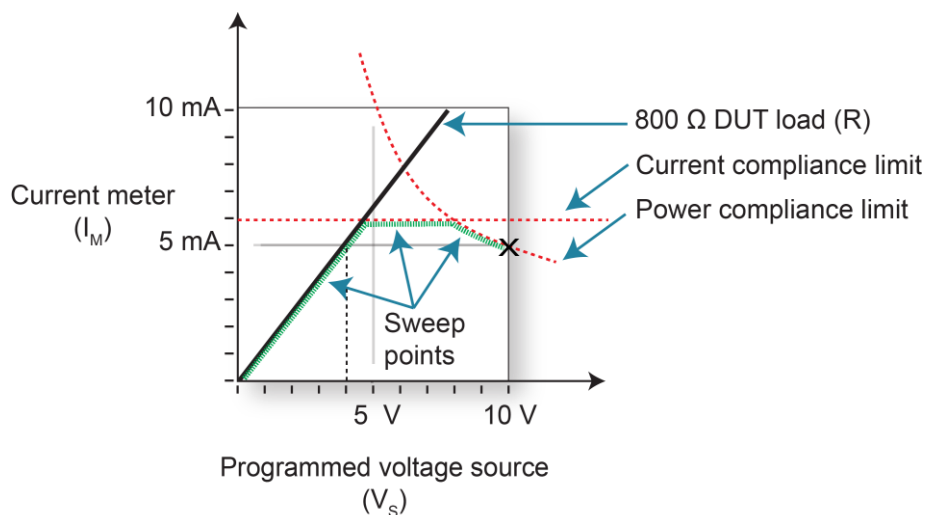
The figure below shows what happens if a power limit of 60 mW is applied. As the instrument attempts to output the programmed source value of 10 V, the power compliance limit line is reached, placing the 2606B in power compliance. The 2606B enforces the power compliance limit by setting the current compliance limit line to the new power limited current compliance limit line setting, which in this case is 8.66 mA. In compliance, the 2606B cannot source its programmed voltage (10 V). For the 800 Ω DUT, the 2606B only outputs 6.928 V (at the 60 mW limit). In this example, current never exceeds the programmed compliance of 10 mA, or the programmed power compliance of 60 mW, under any load.

Figure 90: Voltage source operation in power compliance



Voltage source sweep operation in current and power compliance

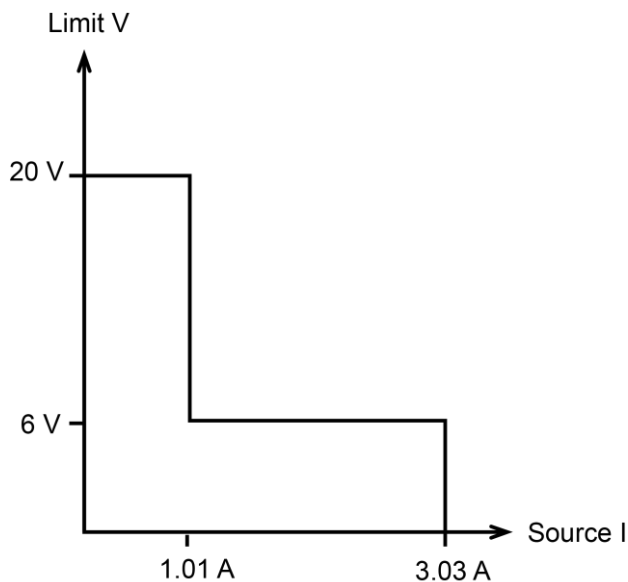
The following figure shows a voltage sweep on a resistive load of 800 Ω . For this example, the 2606B is programmed to sweep voltage to 10 V, limit current (6 mA) and limit power (50 mW). When sweeping, the actual source output varies according to the programmed source value until the current limit is reached. As the figure shows, the output sources the programmed value until placed in current compliance at the 6 mA limit. The sweep then continues. Programmed current source values increase along the green sweep points line, but the output remains at the value it was at when the instrument went into voltage compliance. This continues until the programmed source value sweeps to a high enough level that the power limit line is reached (50 mW). At this point, the current and voltage start to decrease, lowering the current and voltage values along the DUT load line. When the last point is swept (10 V), the actual output is 10 V (at 5 mA).

Figure 91: Programmed voltage source sweep operation in current and power compliance

I-source operating boundaries

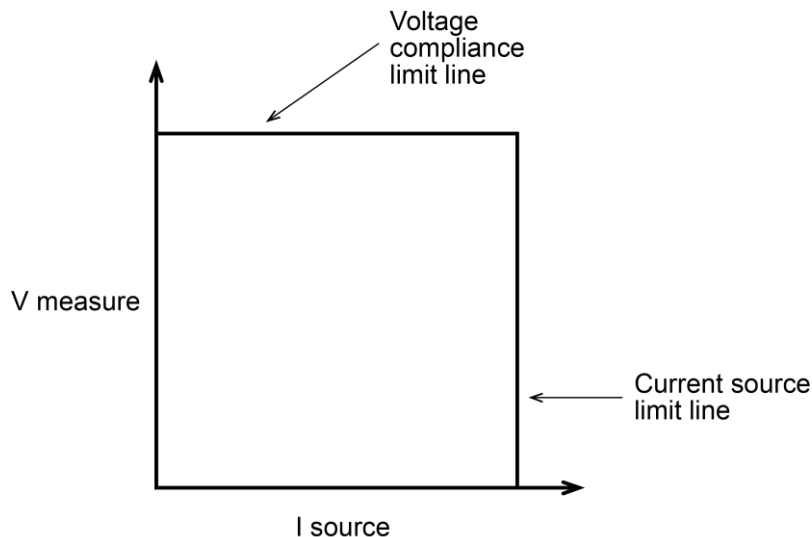
The following figures show the operating boundaries for the current source. Only the first quadrant of operation is shown; operation in the other quadrants is similar.

The first figure shows the output characteristics for the current source. As shown, the 2606B can output up to 1.01 A at 20 V or 3.03 A at 6 V. Note that when sourcing more than 1.01 A, voltage is limited to 6 V.

Figure 92: I-source output characteristics

The following figure shows the limit lines for the current source. The current source limit line represents the maximum source value possible for the presently selected current source range. The voltage compliance limit line represents the actual compliance that is in effect (see [Limits](#) (on page 4-1)). These limit lines are boundaries that represent the operating limits of the instrument for this quadrant of operation. The operating point can be anywhere inside (or on) these limit lines. The limit line boundaries for the other quadrants is similar.

Figure 93: I-source limit lines



Load considerations for current source

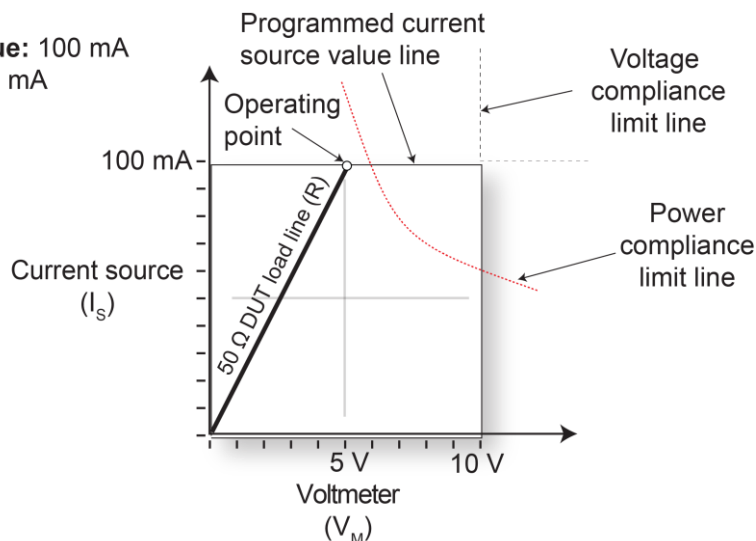
The boundaries within which the 2606B operates depend on the load (device-under-test (DUT)) that is connected to its output. The following topics show operation examples for resistive loads that are 50 Ω and 200 Ω , respectively. For these examples, the 2606B is programmed to source 100 mA and limit voltage (10 V). In addition, for normal current source operation and current source operation in power compliance, the 2606B is programmed to limit power (600 mW).

Normal current source operation

In normal current source operation, the instrument is sourcing 100 mA to the 50 Ω load and subsequently measures 5 V. As shown in the following figure, the load line for 50 Ω intersects the 100 mA current source line at 5 V. The voltage compliance limit and the power compliance limit are not reached (the instrument is not limited through its compliance settings).

Figure 94: Normal current source operation

Programmed source value: 100 mA
Actual source value: 100 mA



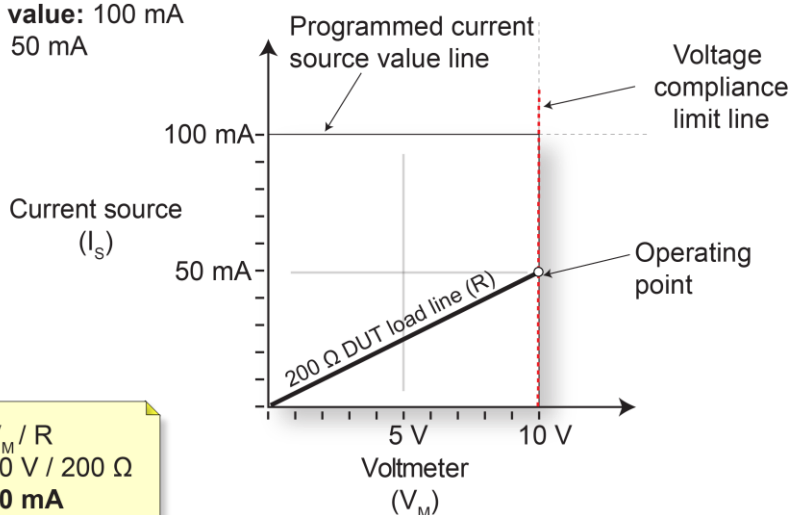
Current source operation in voltage compliance

The following figure shows what happens if the resistance of the load is increased to 200 Ω . The DUT load line for 200 Ω intersects the voltage compliance limit line, placing the instrument in voltage compliance. In voltage compliance, the instrument cannot source its programmed current (100 mA). For the 200 Ω DUT, the instrument only outputs 50 mA (at the 10 V limit).

As resistance increases, the slope of the DUT load line increases. As resistance increases and approaches infinity (open output), the 2606B sources virtually 0 mA at 10 V. Conversely, as resistance decreases, the slope of the DUT load line decreases. At zero resistance (shorted output), the 2606B sources 100 mA at virtually 0 V.

Figure 95: Current source operation in voltage compliance

Programmed source value: 100 mA
Actual source value: 50 mA

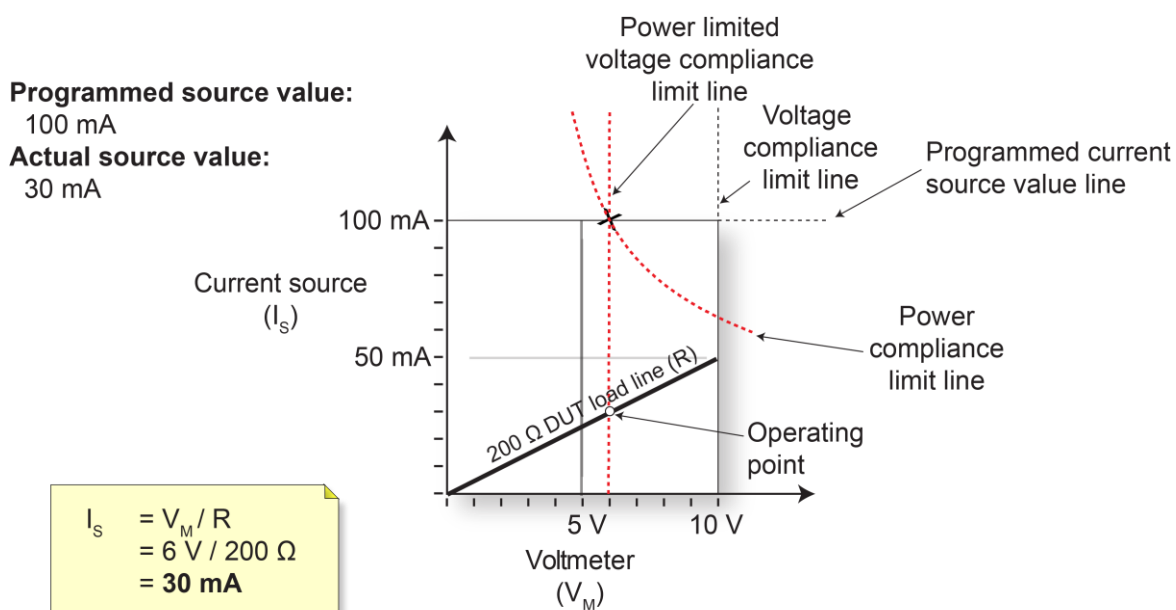


$$\begin{aligned} I_S &= V_M / R \\ &= 10 \text{ V} / 200 \Omega \\ &= 50 \text{ mA} \end{aligned}$$

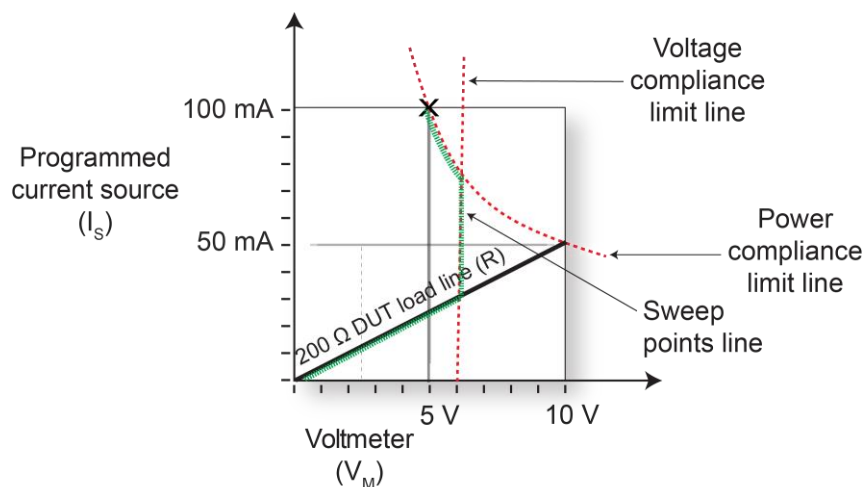
Current source operation in power compliance

The following figure shows the results when a power limit of 600 mW is applied. As the instrument attempts to output the programmed source value of 100 mA, the power limited voltage compliance limit line is reached, placing the instrument in power compliance. The instrument enforces the power compliance limit by setting the voltage compliance limit line to the new power limited voltage compliance limit line setting (in this case, 6 V). In compliance, the instrument cannot source its programmed current (100 mA). For a 200 Ω DUT, the instrument only outputs 30 mA (at the 6 V limit). In this example, voltage never exceeds the programmed compliance of 10 V or the programmed power compliance of 600 mW under any load.

Figure 96: Current source operation in power compliance



The following figure shows a current sweep on a resistive load of 200 Ω . For this example, the instrument is programmed to sweep current to 100 mA, limit voltage (6 V), and limit power (500 mW). When sweeping, the actual source output varies according to the programmed source value until the voltage limit is reached. As the figure shows, the output sources the programmed value until placed in voltage compliance at the 6 V limit. The sweep then continues (programmed current source values increase along the green sweep points line), but the output remains at the same value as when the instrument went into voltage compliance. This continues until the programmed source value sweeps to a high enough level that the power limit line is reached (500 mW). At this point, the voltage and the current start to decrease, lowering the current and voltage values along the DUT load line. When the last point is swept (100 mA), the actual output is 25 mA (at 5 V).

Figure 97: Current source sweep operation in voltage and power compliance

Guard

⚠ WARNING

GUARD is at the same potential as output HI. If hazardous voltages are present at output HI, they are also present at the GUARD terminal.

The rear-panel GUARD terminals are always enabled and provide a buffered voltage that is at the same level as the HI (or SENSE HI for remote sense) voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between HI and LO. In the absence of a driven guard, leakage in the external test circuit could be high enough to adversely affect the performance of the SMU.

Leakage current can occur through parasitic or nonparasitic leakage paths. An example of parasitic resistance is the leakage path across the insulator in a coaxial or triaxial cable. An example of nonparasitic resistance is the leakage path through a resistor that is connected in parallel to the device-under-test (DUT).

Guard connections

Guard is typically used to drive the guard shields of cables and test fixtures. Guard is extended to a test fixture from the cable guard shield. Inside the test fixture, the guard can be connected to a guard plate or shield that surrounds the device-under-test (DUT).

WARNING

A safety shield must be used whenever hazardous voltages ($>30\text{ V}_{\text{RMS}}$, $42\text{ V}_{\text{PEAK}}$) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the 2606B in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield. The figures in this topic show the metal case of a test fixture being used as a safety shield.

NOTE

See [Guarding and shielding](#) (on page 4-7) for details about guarded test connections.

Inside the test fixture, a triaxial cable can be used to extend guard to the DUT. The center conductor of the cable is used for HI, and the inner shield is used for guard.

The figures below show how cable guard can eliminate leakage current through the insulators in a test fixture. In this figure, leakage current (I_L) flows through the insulators (R_{L1} and R_{L2}) to LO, adversely affecting the low-current (or high-resistance) measurement of the DUT.

Also in the figures below, the driven guard is connected to the cable shield and extended to the metal guard plate for the insulators. Since the voltage on either end of R_{L1} is the same (0 V drop), no current can flow through the leakage resistance path, so the instrument only measures the current through the DUT.

Figure 98: Unguarded measurements

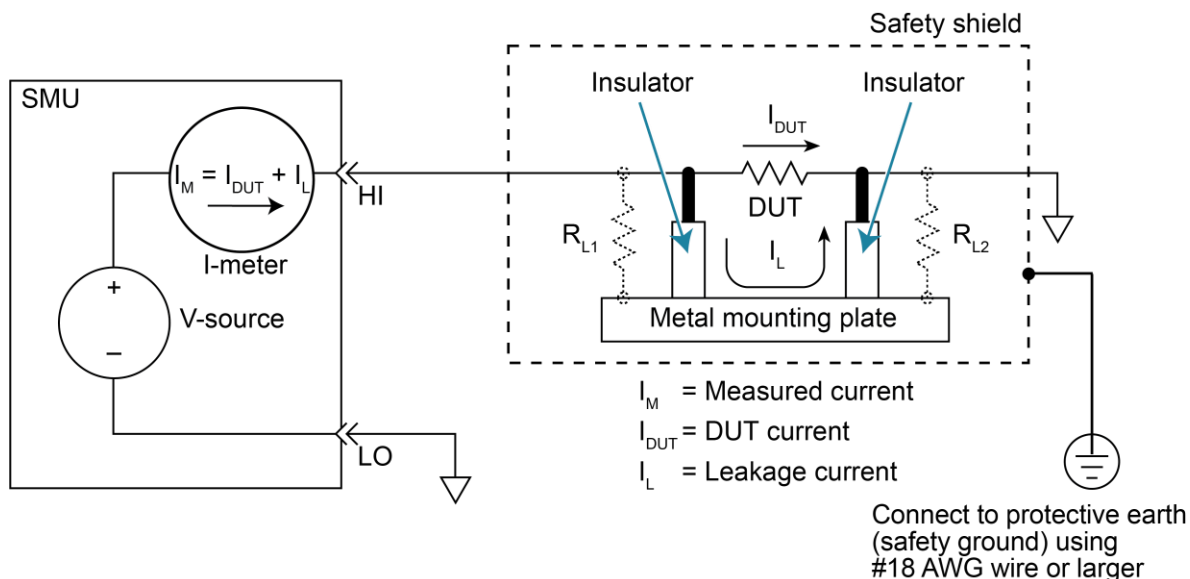
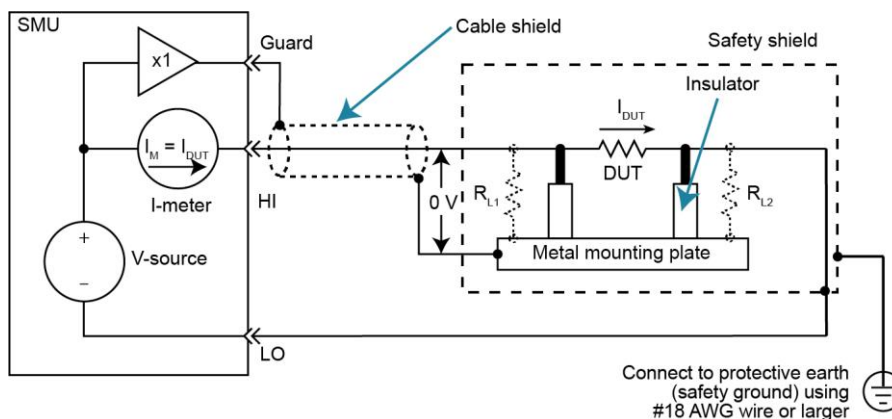


Figure 99: Guarded measurements



Analog-to-digital converter

The 2606B SMUs have integrating analog-to-digital converter (ADCs). The integrating ADCs use a ratiometric analog-to-digital conversion technique. Depending on the configuration of the integrating ADCs, periodic fresh reference measurements are required to minimize drift. The measurement aperture is used to determine the time interval between these measurement updates. For additional information, see [Autozero](#) (on page 5-10). To optimize operation of these ADCs, the instrument caches the reference and zero values for the ten most recent power-line cycles. For additional information, see [NPLC caching](#) (on page 5-11).

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. By default (when power is turned on or after a `smuX.reset()` command), delays are enforced to account for approximately 6τ or 6 time constants of a given range (to reach 0.1 percent of the final value, assuming 2.3τ per decade). The table below lists the current ranges and associated default delays.

Current measure settling time^{1, 2}	
Time required to reach 0.1% of final value after source level command is processed on a fixed range. Values below for $V_{out} = 2\text{ V}$ unless otherwise noted	
Current range	Settling time
1 A to 3 A	<120 μs (typical) ($R_{LOAD} > 6\ \Omega$)
100 mA to 10 mA	<80 μs (typical)
1 mA	<100 μs (typical)
100 μA	<150 μs (typical)
10 μA	<500 μs (typical)
1 μA	<2.5 ms (typical)
100 nA	<15 ms (typical)
¹ Delay factor set to 1. Compliance equal to 100 mA.	
² Time for measurement to settle after a V_{STEP} .	

NOTE

Delays are off by default for the 2606B but can be enabled.

You can manipulate the default delays to produce faster response times. Control the default delays by using the delay factor multiplier. The default value for delay factor multiplier is 1.0, but adjusting it to other values result in either a faster or slower response. For example, increasing the delay factor to 1.3 accounts for settling to 0.01 percent of the final value.

The following code provides measure delay examples for controlling settling time delay of SMU Channel A:

```
-- To turn off measure delay (default setting is smua.DELAY_AUTO).
smua.measure.delay = 0
-- set measure delay for all ranges to Y (in seconds).
smua.measure.delay = Y
-- To adjust the delay factor.
smua.measure.delayfactor = 1.0
```

The delay factor is used to multiply the default delays. Setting this value above 1.0 increases the delays; a value below 1.0 decreases the delay. Setting this value to 0.0 essentially turns off measurement delays. This attribute is only used when `smuX.measure.delay` is set to `smuX.DELAY_AUTO`.

Effects of load on current source settling time

The settling time of the source-measure unit (SMU) can be influenced by the impedance of the device-under-test (DUT) in several ways. One influence is caused by an interaction between the impedances of the SMU current source feedback element and the DUT. This interaction can cause a reduction in the bandwidth of the SMU. This reduction results in an increase in the settling time of the current source.

There is a maximum DUT impedance for each current source range for which the specified current settling times are maintained. The following table lists the DUT impedances for each of these current source ranges. For the latest specifications, go to tek.com/keithley. The settling time on a current source range can increase significantly when measuring DUTs that have an impedance that is higher than the maximum DUT impedance listed below.

Maximum DUT impedances for specified settling time performance

Range	SMU feedback impedance	Maximum DUT impedance
100 nA	40 M Ω	20 M Ω
1 μ A	1.2 M Ω	600 k Ω
10 μ A	400 k Ω	200 k Ω
100 μ A	12 k Ω	6 k Ω
1 mA	4 k Ω	2 k Ω
10 mA	120 Ω	60 Ω
100 mA	40 Ω	20 Ω
1 A	1 Ω	6 Ω
3 A	0.3 Ω	1.5 Ω

Creating pulses with the 2606B SMU

Although the 2606B is not a pulse generator, you can create pulses by programming the SMU to output a dc value and then return to an idle level. For information on how to create pulses, refer to [Sweep operation](#) (on page 5-34) and [Using the remote trigger model](#) (on page 7-3).

Pulse rise and fall times

NOTE

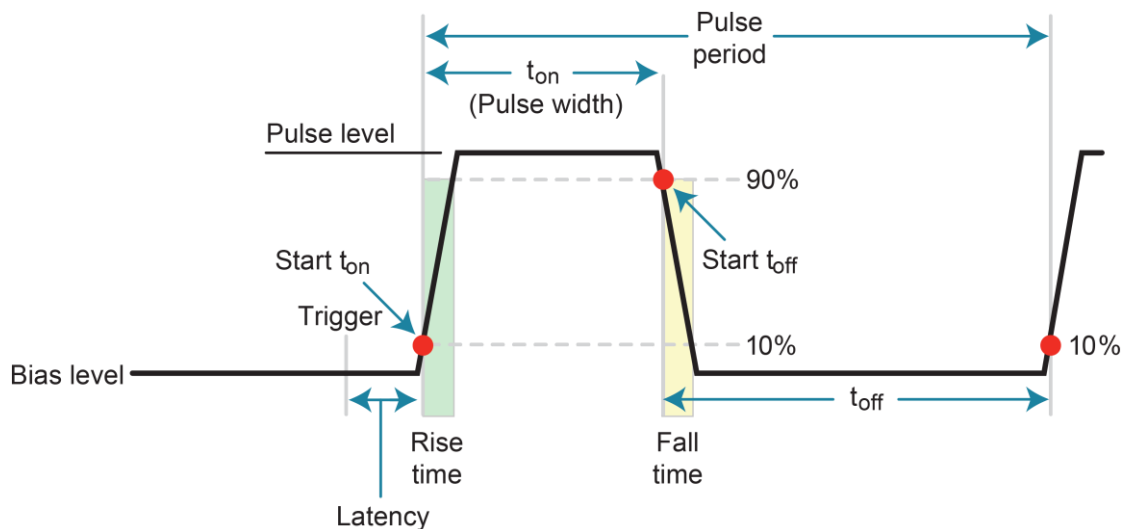
Although the 2606B can create pulses, it is not a pulse generator (pulse rise times are not programmable).

The pulse rise time is the time it takes a pulse to go from 10% to 90% of the maximum value of the pulse. Pulse fall time is similar but on the trailing edge of the pulse. For the 2606B, pulse rise and fall times can vary depending on the following factors:

- Cable specifications and connection configuration
- [Range and pulse settling](#) (on page 8-18)
- [Load and operating mode](#) (on page 8-18)
- Compliance limit settings (for details, see [Limit principles](#) (on page 8-1))

Refer to the 2606B specifications for details on source settling times. For the latest specifications, go to tek.com/keithley.

Figure 100: Pulse rise and fall times



Range and pulse settling

Each SMU range has different specifications for source settling times. This causes different rise and fall time characteristics depending on the set range.

In addition, pulse performance is dependent on the pulse setting as a percent of full scale. For example, a 100 mA pulse on the 1 A range (which is 10% of full scale) performs differently than a 1 A pulse on the 1 A range (which is full scale). Refer to the 2606B specifications for details. For the latest specifications, go to tek.com/keithley.

SMU load and operating mode

Settling times for the current source vary with the resistive load applied. In addition to the load, the times vary depending on whether the source-measure unit (SMU) is configured as a voltage source or a current source, and also if the voltage source range is selected.

Pulse width

The pulse width is the interval between 10% on the rising (leading) edge to 90% on the falling (trailing) edge. The jitter of the pulse width can change the pulse width. This is especially important for short pulse widths.

NOTE

With respect to pulse width, jitter is the short-term instability of the trailing edge relative to the leading edge.

The usable pulse width is largely affected by the source settling time and measurement speed, as well as by the measure count. Review the 2606B specifications for information on source settling time. For the latest specifications, go to tek.com/keithley.

LAN concepts and settings

In this section:

Overview	9-1
Install the crossover cable.....	9-1
Connecting to the LAN	9-2
Establishing a point-to-point connection.....	9-5
LAN speeds.....	9-7
Duplex mode	9-7
Viewing LAN status messages.....	9-8
Viewing the network settings.....	9-9
Selecting a LAN interface protocol	9-10
Logging LAN trigger events in the event log	9-12

Overview

This section describes how to connect to the LAN.

The 2606B is version 1.5 LXI Device Specification 2016 compliant. The 2606B is a scalable test system that can connect directly to a host computer or interact with a DHCP or DNS server and other LXI-compliant instruments on a local area network (LAN). The 2606B also supports Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are useful on a LAN with no central administration.

The 2606B is compliant with the IEEE Std 802.3 and supports full connectivity on a 10 or 100 megabits-per-second network. The LAN interface is an alternative to GPIB that can be used to build flexible test systems that include web access.

NOTE

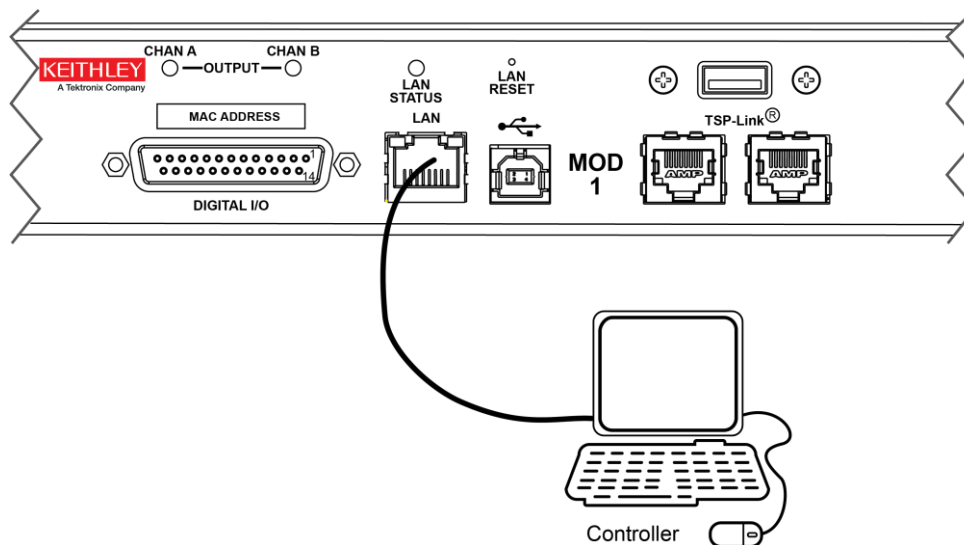
Please read this entire section before you connect the 2606B to the LAN.

Install the crossover cable

Use a LAN crossover cable (RJ-45, male to male) or a straight-through cable to connect your equipment. The 2606B includes a cable that you can use for the TSP-Link® network or LAN communications. The instrument automatically senses which cable you have connected.

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

Figure 101: LAN connection



NOTE

Connect the crossover cable into the same computer LAN port used during instrument configuration to ensure that the system is using the correct network card.

Connecting to the LAN

Each device on the LAN (corporate or private) requires a unique IP address. Contact your corporate information technology (IT) department for details about obtaining an IP address before you deploy the 2606B on a corporate or private network.

The instrument comes from the factory with both modules set to manual configuration method. The IP address for module 1 is set to 192.168.0.3 and the address for module 2 is set to 192.168.0.4. You can change these settings as needed, as described in the following topics.

NOTE

Contact your corporate IT department for permission before you connect the 2606B to a corporate network.

Set the IP address to be set automatically

When the TCP/IP configuration mode is set to Automatic, a DHCP server automatically sets the IP address, subnet mask, and the default gateway. To use this option, a DHCP server must be available on the network.

NOTE

The following steps reset the TCP/IP configuration, which also resets the instrument password. This allows remote access to LXI web page controls that may have been password-protected before the reset.

To use the LAN RESET switch to select Automatic configuration:

1. Turn on instrument power.
2. Wait for the instrument to complete power up.
3. Insert a straightened paper clip into the recessed LAN RESET switch on the 2606B front panel, pressing the switch briefly. The LAN status indicator illuminates when the connection is established.
4. To find the IP address that was automatically applied, use the LXI Discovery Tool, a utility that is available from the Resources tab of the [LXI Consortium website \(lxistandard.org\)](http://lxistandard.org).

Set the LAN configuration to Manual

When you use the manual configuration method with fixed LAN configuration settings, the instrument IP address, subnet mask, and gateway are set to the following fixed values:

- **IP address:** 192.168.0.2
- **Gateway:** 192.168.0.1
- **Subnet mask:** 255.255.255.0

You may need to set up a point-to-point connection to use these settings. Refer to [Establishing a point-to-point connection](#) (on page 9-5).

The LAN settings persist when the instrument power is turned off and then turned on again.

To use the LAN RESET switch to select manual mode using fixed settings:

1. If the instrument power is on, turn it off. You set the LAN RESET switch as the instrument powers on.
2. Insert a straightened paper clip into the recessed LAN RESET switch on the 2606B front panel, pressing and holding the switch while you turn the instrument on.
3. Continue pressing the LAN RESET switch until the CHAN A and CHAN B indicators on the front panel illuminate briefly.
4. Release the LAN RESET switch. The LAN status indicator flashes twice to confirm the fixed LAN configuration settings.
5. Connect remotely to the web interface of the instrument by typing `192.168.0.2` into the URL field of your web browser. When you are connected, the web interface is displayed.
6. In the navigation panel on the left side of the screen, select **IP Configuration**.
7. Select **Modify**. You may be prompted to log in using a password. The default password is "" (an empty string).
8. Enter your LAN configuration settings and select **Submit**.

Configuring the Domain Name System (DNS)

The Domain Name System (DNS) lets you type a domain name in the address bar to connect to the instrument. If you use DNS, you can use a name instead of an IP address.

Example:

Model2606B.XYZcompany.com

NOTE

Contact your corporate information technology (IT) department for information about DNS. If a DNS server is not part of the LAN infrastructure, do not use this setting.

To enable or disable DNS host name verification:

1. From the virtual front panel, press the **MENU** key, and then select **LAN > CONFIG > DNS > VERIFY**.
2. Turn the navigation wheel to select either **ENABLE** or **DISABLE**. When enabled, the instrument performs a DNS lookup to verify the DNS host name matches the value specified in the [lan.config.dns.hostname](#) (on page 15-128) attribute.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.

To enable or disable DNS registration:

1. From the virtual front panel, press the **MENU** key and select **LAN > CONFIG > DNS > DYNAMIC**.
2. Turn the navigation wheel to select either **ENABLE** or **DISABLE**. DNS registration works with the DHCP to register the host name specified in the `lan.config.dns.hostname` attribute with the DNS server.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To set the DNS server IP addresses:

1. From the virtual front panel, press the **MENU** key and select **LAN > CONFIG > DNS**.
2. Turn the navigation wheel to select either **DNS-ADDRESS1** or **DNS-ADDRESS2**.
3. Press the **ENTER** key.
4. Turn the navigation wheel to select and enter a valid IP address for the DNS server.
5. Press the **ENTER** key.
6. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
7. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Establishing a point-to-point connection

To enable access to the instrument web page and other web applications from a computer, use a one-to-one LAN connection and set up a static IP address between the host computer and the instrument.

The following instructions describe how to configure the IP address of the instrument. The IP address of the instrument is based on the present IP address of the host computer. Each device on the LAN (corporate or private) requires a unique IP address.

CAUTION

Do not change the IP address at any time without talking to your system administrator. Entering an incorrect IP address can prevent your workstation from connecting to your corporate network.

If you have problems, see [LAN troubleshooting suggestions](#) (on page 12-6).

NOTE

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 Enabled = NO (disabled).

Be sure to return all settings to their original configuration before reconnecting the host computer to a corporate network. Failure to do this could result in loss of data. Contact your system administrator for more information.

NOTE

These steps assume that you are making all the settings in the order shown here. If you only change one or a few settings, be aware that you need to apply the settings before they will be in effect. To apply the settings, from the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To configure the 2606B using the virtual front panel:

1. Get the IP address of the computer to which the instrument will be connected.
2. Press the **MENU** key to display the MAIN MENU.
3. Use the navigation wheel to select **LAN**. The LAN CONFIG menu is displayed.
4. Change the IP address assignment method:
 - a. Select **CONFIG > METHOD > MANUAL**, and then press the **ENTER** key.
 - b. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Enter the IP address using the LAN CONFIG menu:
 - a. Select **CONFIG > IP-ADDRESS**.
 - b. Enter the IP address. A portion of the computer's IP address is used as a base for the instrument's unique ID. Only the last three numbers (after the last decimal point) of the IP address will differ between the computer and the instrument. If the subnet mask is 255.255.255.0, the last three digits can be any value from 1 to 255.

For example, if the computer's IP address is 192.168.1.100, a unique IP address for the instrument might be 192.168.001.101.

NOTE

The instrument's IP address can have leading zeros, but the computer's IP address cannot.

- c. Use the navigation wheel to select and enter an appropriate IP address for the instrument. Be sure to record the instrument's IP address.
- d. Press **ENTER** key or navigation wheel to confirm the changes.
- e. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.

6. Change the subnet mask from the LAN CONFIG menu:
 - a. Select **CONFIG > SUBNETMASK**, and then press the **ENTER** key. The SUBNETMASK menu item is to the right of GATEWAY. Use the navigation wheel to scroll through the options.
 - b. Modify the SUBNETMASK value to match the computer settings recorded earlier (or 255.255.255.000 if DHCP Enabled = YES).
 - c. Press the **ENTER** key or the navigation wheel when you are finished changing all the characters.
 - d. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
7. From the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.
8. Open a web browser on the host computer.
9. Enter the IP address of the instrument in the web browser address box. For example, if the instrument IP address is 192.168.1.101, enter 192.168.1.101 in the browser address box.
10. Press **Enter** on the computer keyboard to open the instrument web page.

LAN speeds

Another characteristic of the LAN is speed. The 2606B negotiates with the host computer and other LXI-compliant devices on the LAN to transmit data at the highest speed possible. LAN speeds must be configured to match the speed of the other instruments on the network.

To set the LAN speed:

1. From the virtual front panel, press the **MENU** key and select **LAN > CONFIG > SPEED**.
2. Turn the navigation wheel to select either **10 Mbps** or **100 Mbps**.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the previous menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Duplex mode

The duplex mode is based on the LAN configuration. There are two settings:

- **Half-duplex:** Allows communications in both directions, but only one direction is active at a time (not simultaneously).
- **Full:** Permits communications in both directions simultaneously.

To set the duplex mode:

1. From the virtual front panel, press **MENU** key and select **LAN > CONFIG > DUPLEX**.
2. Turn the navigation wheel to select either **HALF** or **FULL**.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Viewing LAN status messages

To view the LAN status messages:

1. From the virtual front panel, press the **MENU** key and select **LAN > STATUS > CONFIG/FAULT**.
2. Press the **ENTER** key.

Figure 102: LAN CONFIG/FAULT



There are two types of LAN status messages:

- **LAN fault messages:** Communicate issues related to physical connectivity.
- **LAN configuration messages:** Communicate issues or events related to configuration.

The following table displays possible fault and configuration messages.

LAN CONFIG/FAULT messages

LAN message type	Possible messages
LAN fault	Could not acquire IP address
	Duplicate IP address detected
	DHCP lease lost
	Lan Cable Disconnected
LAN configuration	Starting DHCP Configuration
	DHCP Server Not Found
	DHCP configuration started on xxx.xxx.xxx.xxx
	Searching for DNS server(s)
	Starting DLLA Configuration
	DLLA Failed
	DLLA configuration started on xxx.xxx.xxx.xxx
	Starting Manual Configuration
	Manual configuration started on xxx.xxx.xxx.xxx
	Closed

Viewing the network settings

To view the active network settings:

1. From the virtual front panel, press the **MENU** key, and then select **LAN > STATUS**.
2. Use the navigation wheel to select one of the following network settings:
 - **IP-ADDRESS**
 - **GATEWAY**
 - **SUBNET-MASK**
 - **METHOD**
 - **DNS**
 - **MAC-ADDRESS**
3. Press the **ENTER** key to view the active setting.
4. Press the **EXIT (LOCAL)** key once to return to the STATUS menu.

Confirming the active speed and duplex negotiation

The 2606B automatically detects the speed and duplex negotiation active on the LAN. Once the speed and duplex negotiation is detected, the instrument automatically adjusts its own settings to match the LAN settings.

To confirm the active LAN speed and duplex mode:

1. From the virtual front panel, press the **MENU** key.
2. Select **LAN > STATUS**.
3. Use the navigation wheel to select one of the following:
 - **SPEED**
 - **DUPLEX**
4. Press the **ENTER** key to view the active setting.
5. Press the **EXIT (LOCAL)** key once to return to the STATUS menu.

Confirming port numbers

To view the port number assigned to each remote interface protocol:

1. From the virtual front panel, press the **MENU** key, and then select **LAN > STATUS > PORT**.
2. Use the navigation wheel to select one of the following:
 - **RAW-SOCKET**
 - **TELNET**
 - **VXI-11**
 - **DST**
3. Press the **ENTER** key to view the port number.
4. Press the **EXIT (LOCAL)** key once to return to the PORT menu.

The following table displays the remote interface protocols supported by the 2606B and their assigned port numbers.

Port number

Command interface	Port number
Raw socket	5025
Telnet	23
VXI-11	1024
DST (dead socket termination)	5030

Selecting a LAN interface protocol

You can use a remote interface protocol to connect to the 2606B. The 2606B provides Telnet, VXI-11, and raw socket LAN interfaces, with associated LAN protocols (each interface uses a different protocol). Select the interface based on the protocol needed.

You can also use a dead socket termination interface (DST) to troubleshoot connection 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.

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.

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.

Raw socket is a basic ethernet connection that communicates in a manner similar to RS-232 without explicit message boundaries. The instrument always terminates messages with a line feed, but because binary data may include bytes that resemble line-feed characters, it may be difficult to distinguish between data and line-feed characters.

Use raw socket as an alternative to VXI-11. Raw socket offers a faster connection than VXI-11. However, raw socket does not support explicit message boundaries, serial poll, and service requests.

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 are terminated and closed when the connection to the dead socket termination port is closed.

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 need a separate Telnet program to use this protocol.

The 2606B 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.

Logging LAN trigger events in the event log

You can use the event log to record all LXI triggers generated and received by the 2606B. You can view the event log using any command interface or the embedded web interface. The following figure shows the view of the LXI event log from the Log option in the embedded web interface.

Figure 103: LXI Event Log

Receive Time	EventID	From	PTP Timestamp		HWDetect	Sequence	Domain	Flags	Data
			Seconds	FractionalSeconds					
09:51:22.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341530	1	16	0x00
09:51:23.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341531	2	16	0x00
09:51:24.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341532	3	16	0x00
09:51:25.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341533	4	16	0x00
09:51:26.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341534	5	16	0x00
09:51:27.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341535	6	16	0x00
09:51:28.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341536	7	16	0x00
09:51:30.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341537	8	16	0x00
09:51:31.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341538	9	16	0x00
09:51:32.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341539	10	16	0x00
09:51:33.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341540	11	16	0x00
09:51:40.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341546	17	16	0x00
09:51:41.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341547	18	16	0x00

The timestamp, event identifier, IP address, and the domain name identify the incoming and outgoing LXI trigger packets. The following table provides detailed descriptions for the columns in the event log.

Event log descriptions

Column title	Description	Example
Receive Time	Displays the date and time that the LAN trigger occurred in UTC, 24-hour time	11:46:44.000 11 Mar 2020
Event ID	Identifies the <code>lan.trigger[N]</code> that generates an event	LAN0 = <code>lan.trigger[1]</code> LAN1 = <code>lan.trigger[2]</code> LAN2 = <code>lan.trigger[3]</code> LAN3 = <code>lan.trigger[4]</code> LAN4 = <code>lan.trigger[5]</code> LAN5 = <code>lan.trigger[6]</code> LAN6 = <code>lan.trigger[7]</code> LAN7 = <code>lan.trigger[8]</code>
From	Displays the IP address for the device that generates the LAN trigger	localhost 192.168.5.20
Timestamp	A timestamp that identifies the time the event occurred; the timestamp uses the following: - PTP timestamp - Seconds - Fractional seconds; the 2606B does not support the IEEE Std 1588 standard; the values in this field are always 0 (zero)	
HWDetect	Identifies a valid LXI trigger packet	LXI
Sequence	Each instrument maintains independent sequence counters: - One for each combination of UDP multicast network interface and UDP multicast destination port - One for each TCP connection	
Domain	Displays the LXI domain number; the default value is 0 (zero)	0
Flags	Contains data about the LXI trigger packet; values are: 1 - Error 2 - Retransmission 4 - Hardware 8 - Acknowledgments 16 - Stateless bit	16
Data	The values for this are always 0 (zero)	

Accessing the event log from the command interface

You can access the event log from any remote command interface. The event log must be enabled before LXI trigger events can be viewed. To enable the event log, send:

```
eventlog.enable = 1
```

To view the event log from a remote interface, send:

```
print(eventlog.all())
```

This command outputs one or more strings similar to the following:

```
14:14:02.000 17 Jun 2019, LAN0, 10.80.64.191, LXI, 0, 1560780842, not available, 0, 0x10,0x00
```

The string displays the same information as the web interface. Commas separate the fields. The fields output in the following order:

- Received time (UTC time)
- Event ID
- From (Sender)
- HWDetect / version
- Domain
- Sequence number
- Timestamp (PTP time)
- Epoch (from 1588)
- Flags
- Data

See the table in [Logging LAN trigger events in the event log](#) (on page 3-22) for detailed descriptions.

To generate log traffic, send the code:

```
local id = 1
lan.trigger[id].ipaddress = lan.status.ipaddress
lan.trigger[id].connect()
for domain = 1, 255 do
    print(domain)
    lan.lxidomain = domain
    lan.trigger[id].assert()
    delay(1)
end
```

Status model

In this section:

Overview	10-1
Clearing registers	10-13
Programming enable and transition registers.....	10-13
Reading registers	10-14
Status byte and service request (SRQ)	10-14
Status register sets	10-18
TSP-Link system status	10-24

Overview

Each Keithley Instruments 2606B provides status registers and queues that are collectively referred to as the status model. Through manipulation and monitoring of these registers and queues, you can view and control various instrument events. You can include commands in your test program that can determine if a service request (SRQ) event has occurred and the cause of the event.

The heart of the status model is the Status Byte Register. All status model registers and queues flow into the Status Byte Register.

The entire status model is illustrated in the [Status model diagrams](#) (on page 10-4).

Status register set contents

Typically, a status register set contains the following registers:

- **Condition** (`.condition`): A read-only register that is constantly updated to reflect the present operating conditions of the instrument.
- **Enable Register** (`.enable`): A read-write register that allows a summary bit to be set when an enabled event occurs.
- **Event Register** (`.event`): A read-only register that sets a bit to 1 when the applicable event occurs. If the enable register bit for that event is also set, the summary bit of the register is set to 1.
- **Negative Transition Register (NTR)** (`.ntr`): When a bit is set in this read-write register, it enables a 1 to 0 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.
- **Positive Transition Register (PTR)** (`.ptr`): When a bit is set in this read-write register, it enables a 0 to 1 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.

An event is represented by a condition register bit changing from a 1 to 0 or 0 to 1. When an event occurs and the appropriate NTR or PTR bit is set, the corresponding event register bit is set to 1. The event bit remains latched to 1 until the event register is read or the status model is reset. When an event register bit is set and its corresponding enable bit is set, the summary bit of the register is set to 1. This, in turn, sets a bit in a higher-level condition register, potentially cascading to the associated summary bit of the Status Byte Register.

Queues

The 2606B uses queues to store messages. The queues include:

- Command queue: Holds commands that are available for execution.
- Output queue: Holds response messages.
- Error queue: Holds error and status messages.

When a queue contains data, it sets the condition bit for that queue in one of the registers. The condition bits are:

- Command queue: CAV in the Operation Status Remote Summary Register
- Output queue: MAV in the Status Byte Register
- Error queue: EAV in the Status Byte Register

The CAV, MAV, and EAV bits in the registers are cleared when the queue is empty. Queues empty when:

- Commands are executed
- Errors are read from the error queue
- Response messages are read from the instrument

All 2606B queues are first-in, first-out (FIFO).

The [Status model diagrams](#) (on page 10-4) show how the queues are structured with the other registers.

Output queue

Response messages, such as those generated from print commands, are placed in the output queue. All remote command interfaces share the same output queue.

The output queue sets the message available (MAV) bit in the status model.

The data in the output queue is cleared by the *CLS command.

Error queue

The error queue holds error and status messages. As programming errors and status messages occur, a message that defines the error or status is placed in the error queue.

An error or status message is cleared from the error queue when it is read. You can also clear the error queue by sending the command `errorqueue.clear()`. An empty error queue clears the error available (EAV) bit in the Status Byte Register.

Messages in the error queue include a code number, message text, severity, and TSP-Link® node number. See [Error summary list](#) (on page 12-2) for a list of the messages.

When you read a single message from the error queue, the oldest message is read. If you attempt to read the error queue when it is empty, the error number 0 and “No Error” is returned.

The commands used to control the error queue are listed in the following table.

Error queue commands

Error queue command	Description
<code>errorqueue.clear()</code>	Clear error queue of all errors.
<code>errorqueue.count</code>	Number of messages in the error/event queue.
<code>errorCode, message, severity, errorNode = errorqueue.next()</code>	Request error code, text message, severity, and TSP-Link node number.

Status function summary

The following functions and attributes control and read the various registers. Additional information for the various register sets is included later in this section. Also, refer to the specific command as listed in [TSP commands](#) (on page 15-7).

Status function summary	
Type	Function or attribute
Status summary	<code>status.condition</code> <code>status.node_enable</code> <code>status.node_event</code> <code>status.request_enable</code> <code>status.request_event</code> <code>status.reset</code>
Measurement event	<code>status.measurement.*</code> <code>status.measurement.buffer_available.*</code> <code>status.measurement.current_limit.*</code> <code>status.measurement.instrument.*</code> <code>status.measurement.instrument.smua.*</code> <code>status.measurement.instrument.smub.*</code> <code>status.measurement.reading_overflow.*</code> <code>status.measurement.voltage_limit.*</code>

Status function summary	
Type	Function or attribute
Operation status	status.operation.* status.operation.calibrating.* status.operation.instrument.* status.operation.instrument.digio.* status.operation.instrument.digio.trigger_overrun.* status.operation.instrument.lan.* status.operation.instrument.lan.trigger_overrun.* status.operation.instrument.smua.* status.operation.instrument.smub.* status.operation.instrument.smua.trigger_overrun.* status.operation.instrument.smub.trigger_overrun.* status.operation.instrument.trigger_blender.* status.operation.instrument.trigger_blender.trigger_overrun.* status.operation.instrument.trigger_timer.* status.operation.instrument.trigger_timer.trigger_overrun.* status.operation.instrument.tsplink.* status.operation.instrument.tsplink.trigger_overrun.* status.operation.measuring.* status.operation.remote.* status.operation.sweeping.* status.operation.trigger_overrun.* status.operation.user.*
Questionable status	status.questionable.* status.questionable.calibration.* status.questionable.instrument.* status.questionable.instrument.smua.* status.questionable.instrument.smub.* status.questionable.over_temperature.* status.questionable.unstable_output.*
Standard event	status.standard.*
System summary	status.system.* status.system2.* status.system3.* status.system4.* status.system5.*

* = .condition, .event, .ntr, .ptr and .enable

Status model diagrams

The following figures graphically describe the status model:

- [Status byte and service request enable register](#) (on page 10-4)
- [System summary and standard event registers](#) (on page 10-5)
- [Measurement event registers](#) (on page 10-22)
- [Operation status registers](#) (on page 10-7)
- [Operation status trigger overrun registers](#) (on page 10-8)
- [Operation status trigger timer, trigger blender, and remote registers](#) (on page 10-9)
- [Operation status digital I/O and TSP-Link registers](#) (on page 10-10)
- [Questionable status registers](#) (on page 10-12)

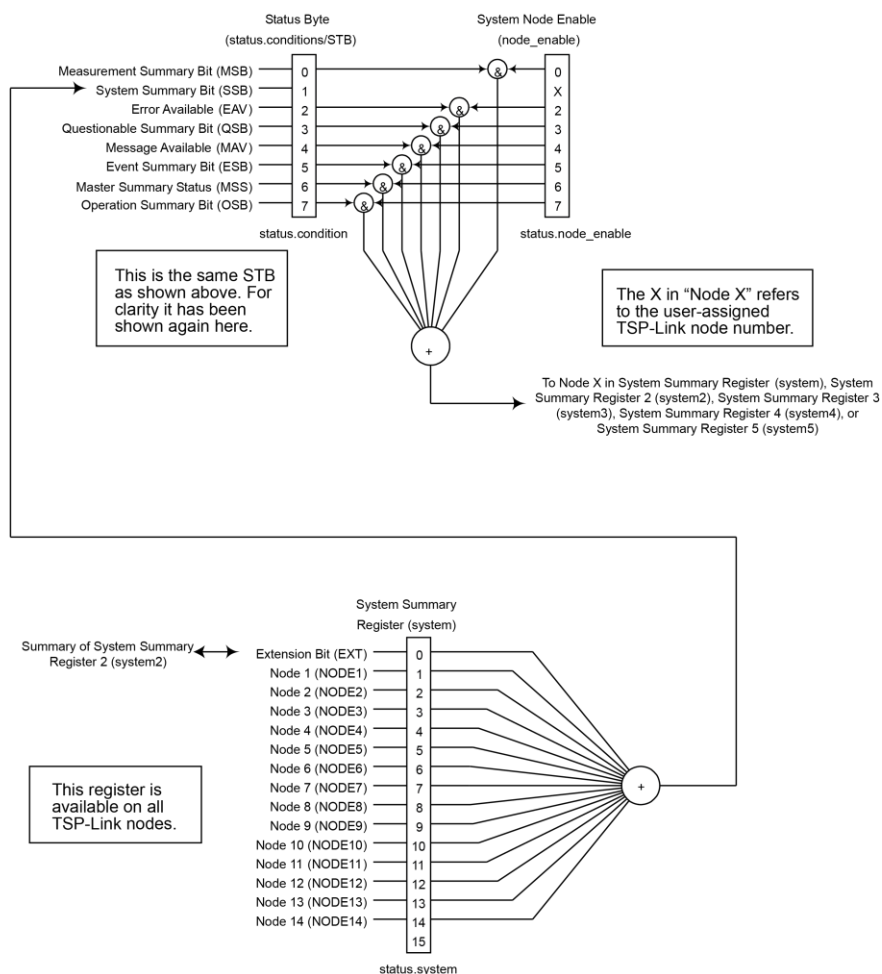
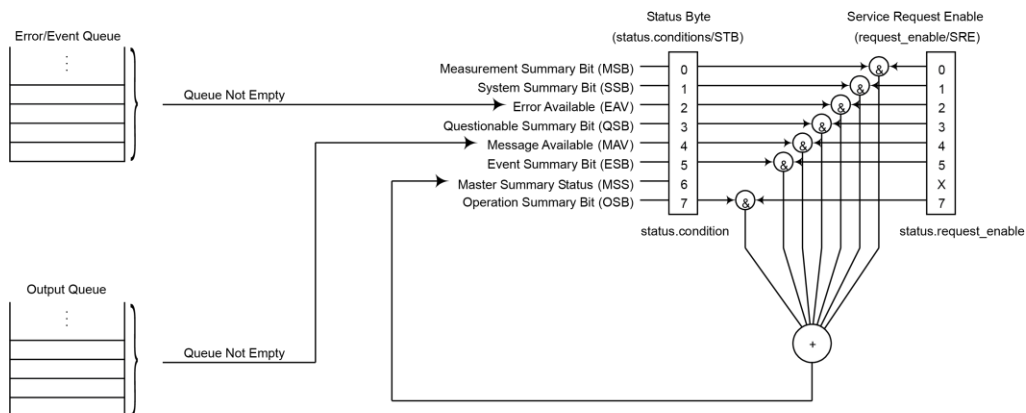
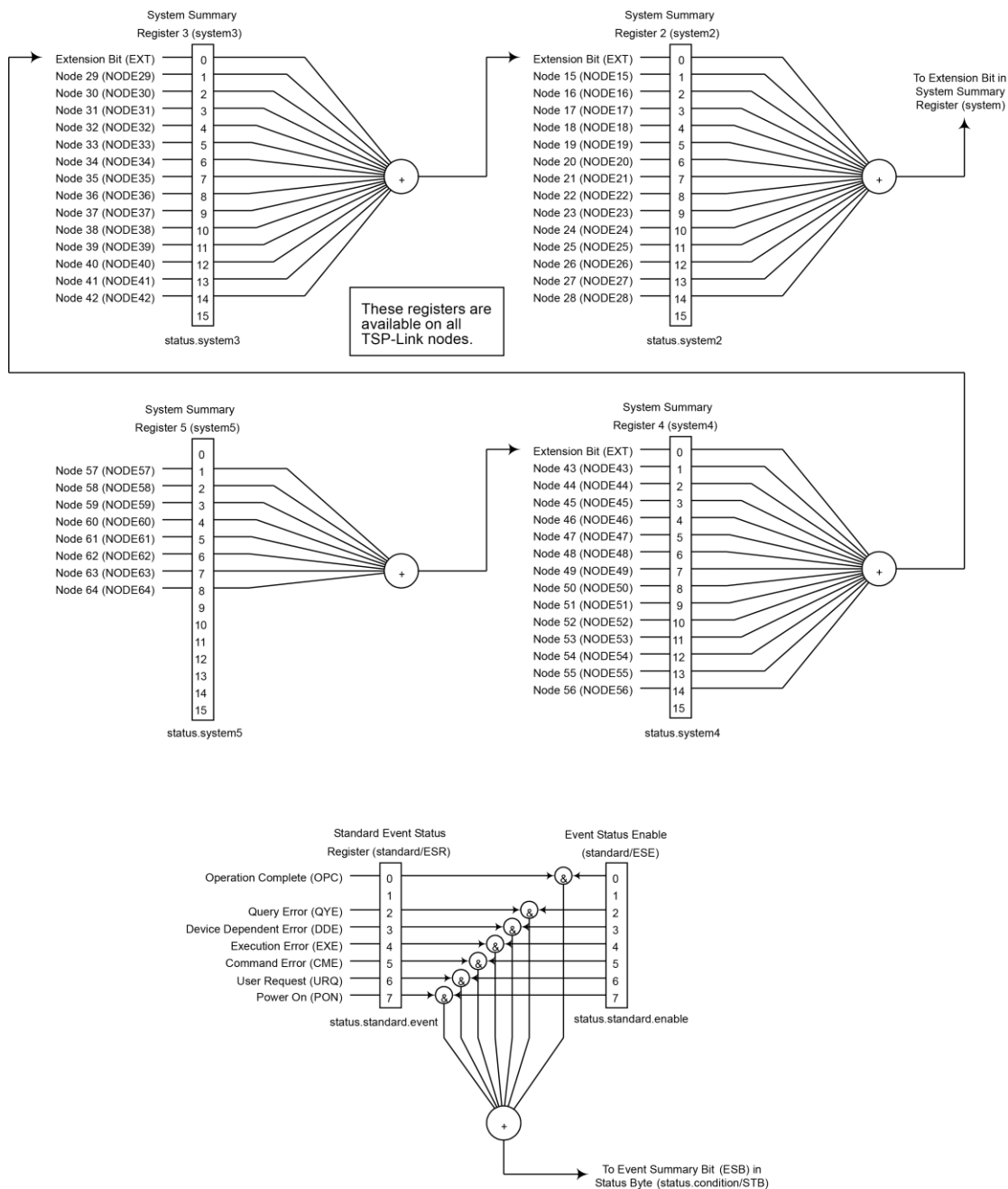
Figure 104: Status byte and service request enable register

Figure 105: System summary and standard event registers

Measurement event registers

Figure 106: Measurement event registers

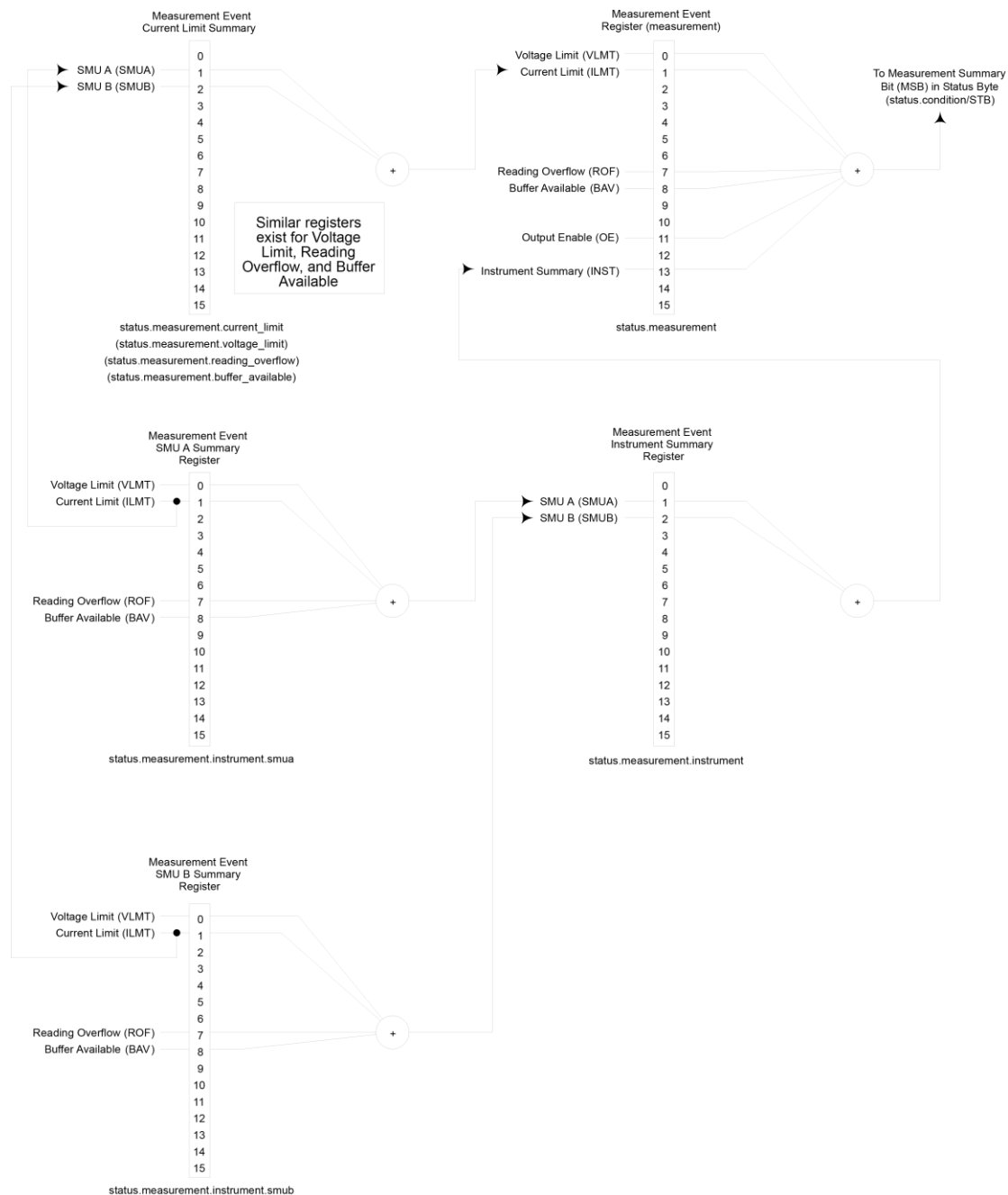


Figure 107: Operation status registers

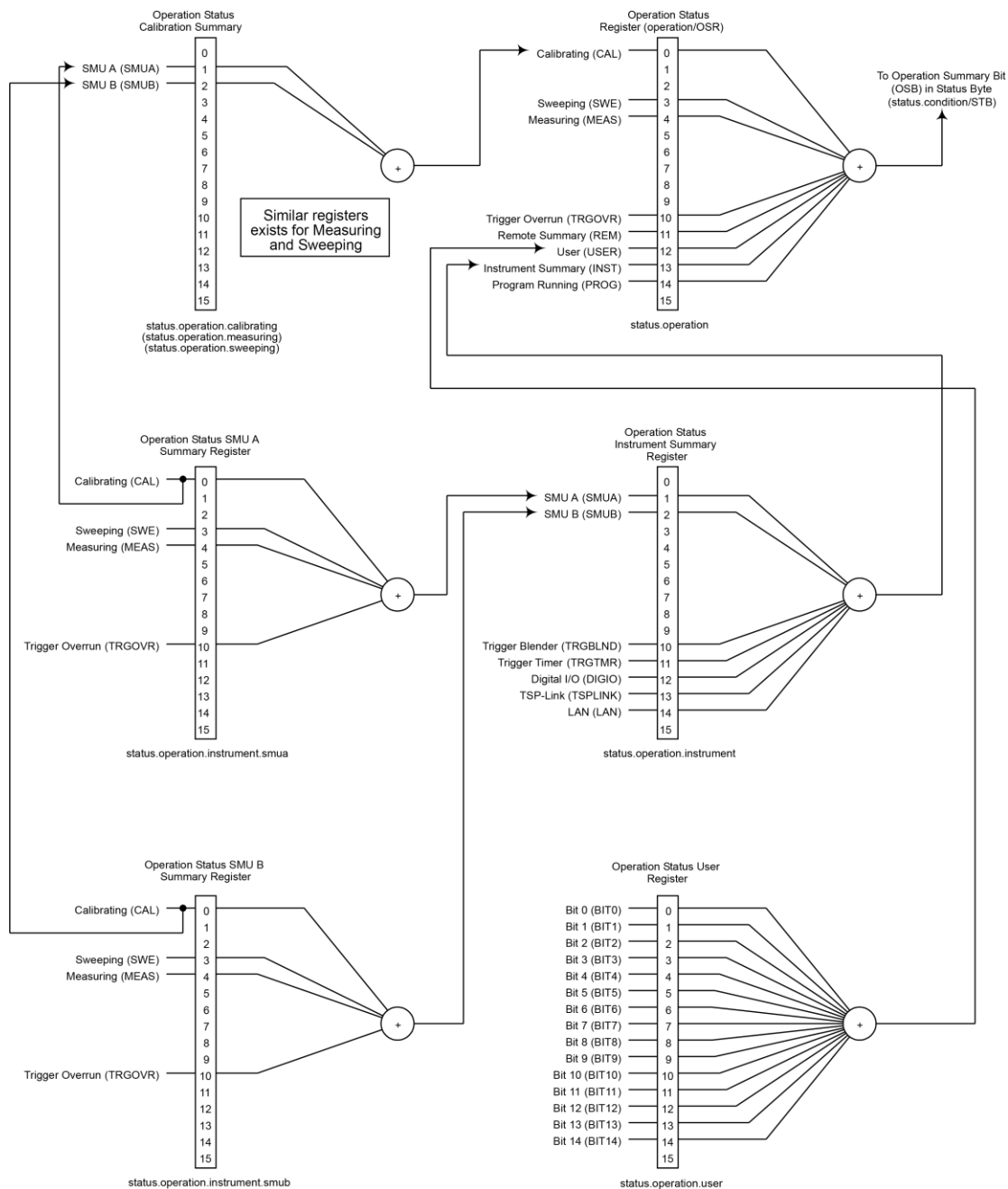


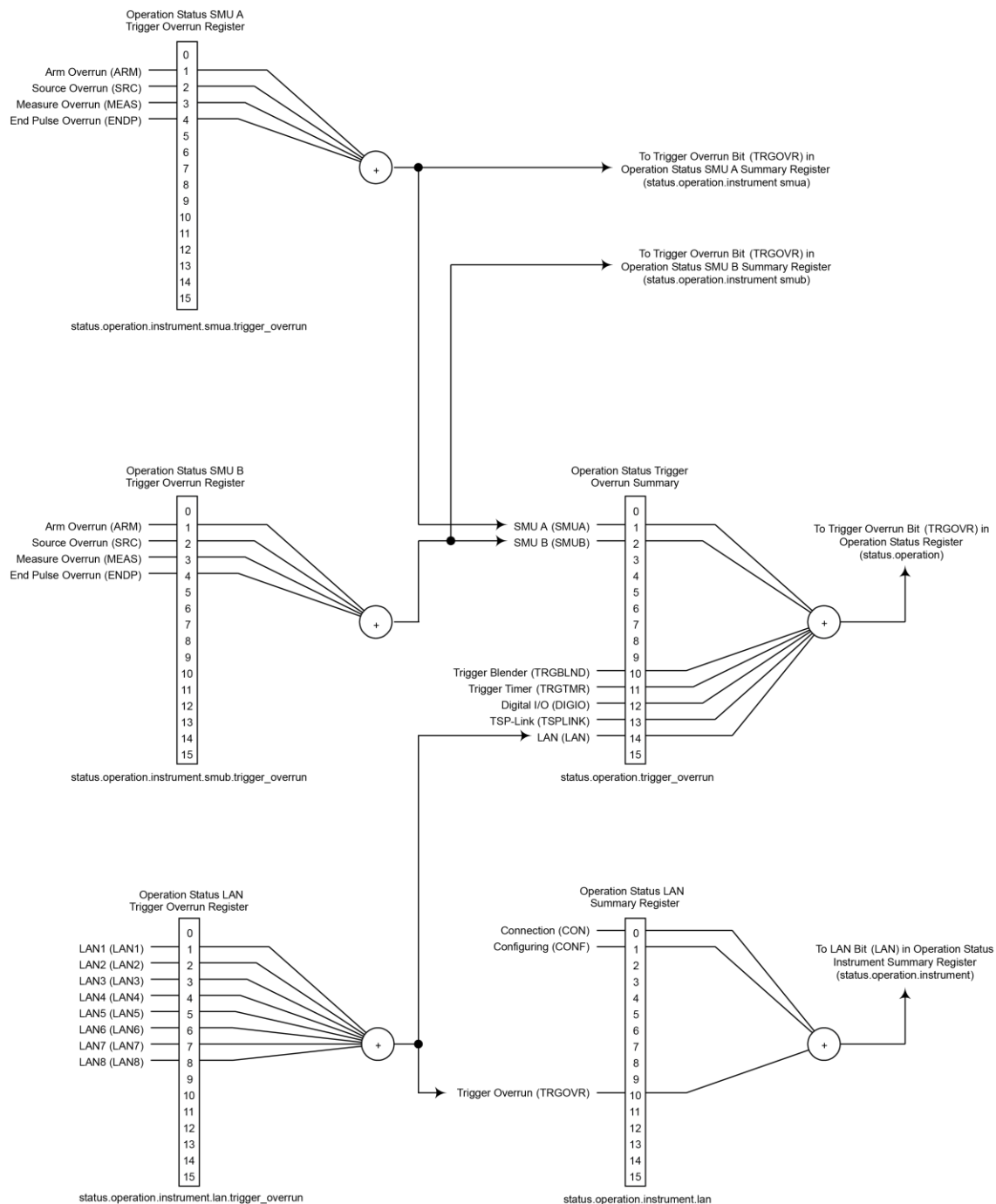
Figure 108: Operation status trigger overrun registers

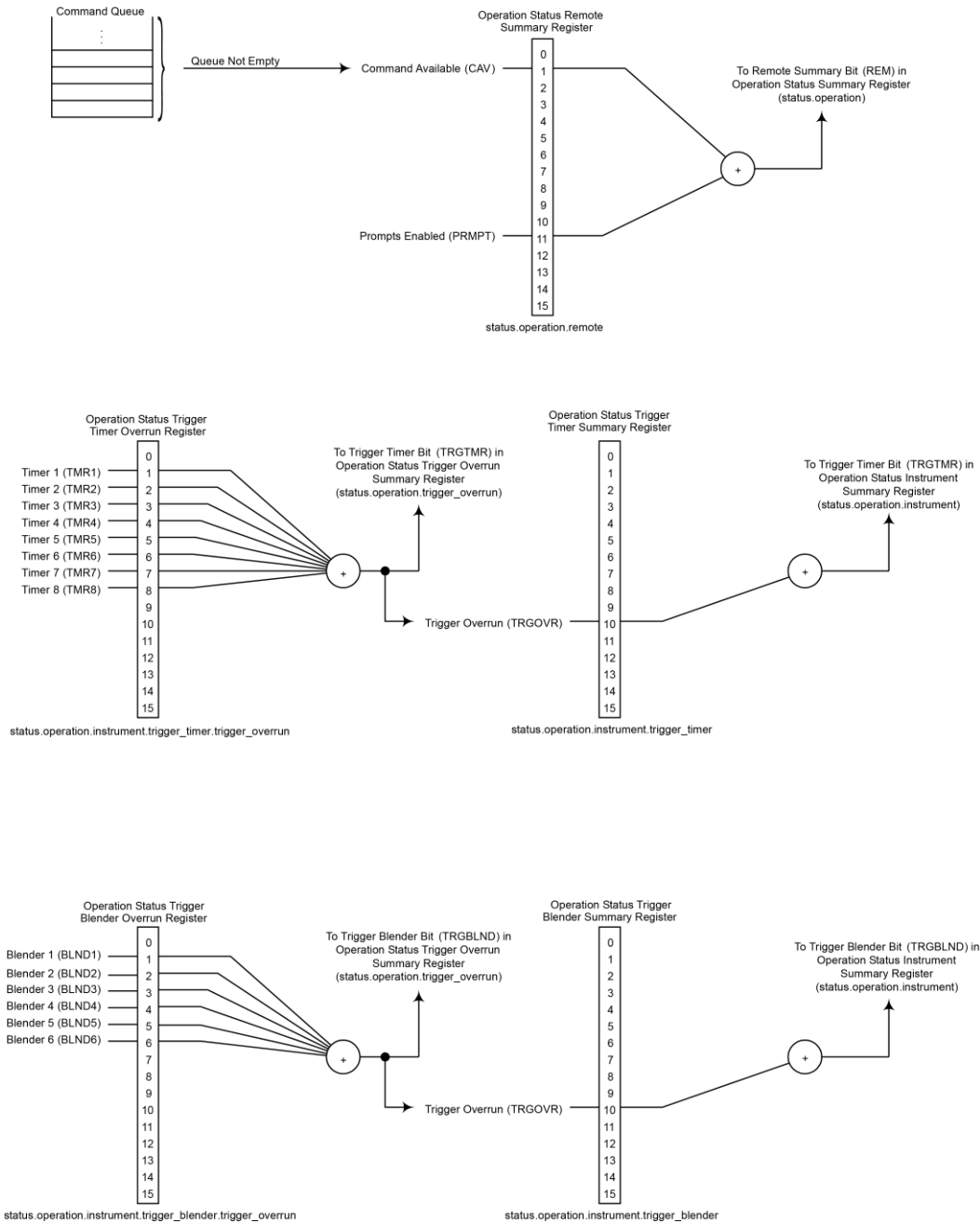
Figure 109: Operation status trigger timer, trigger blender, and remote registers

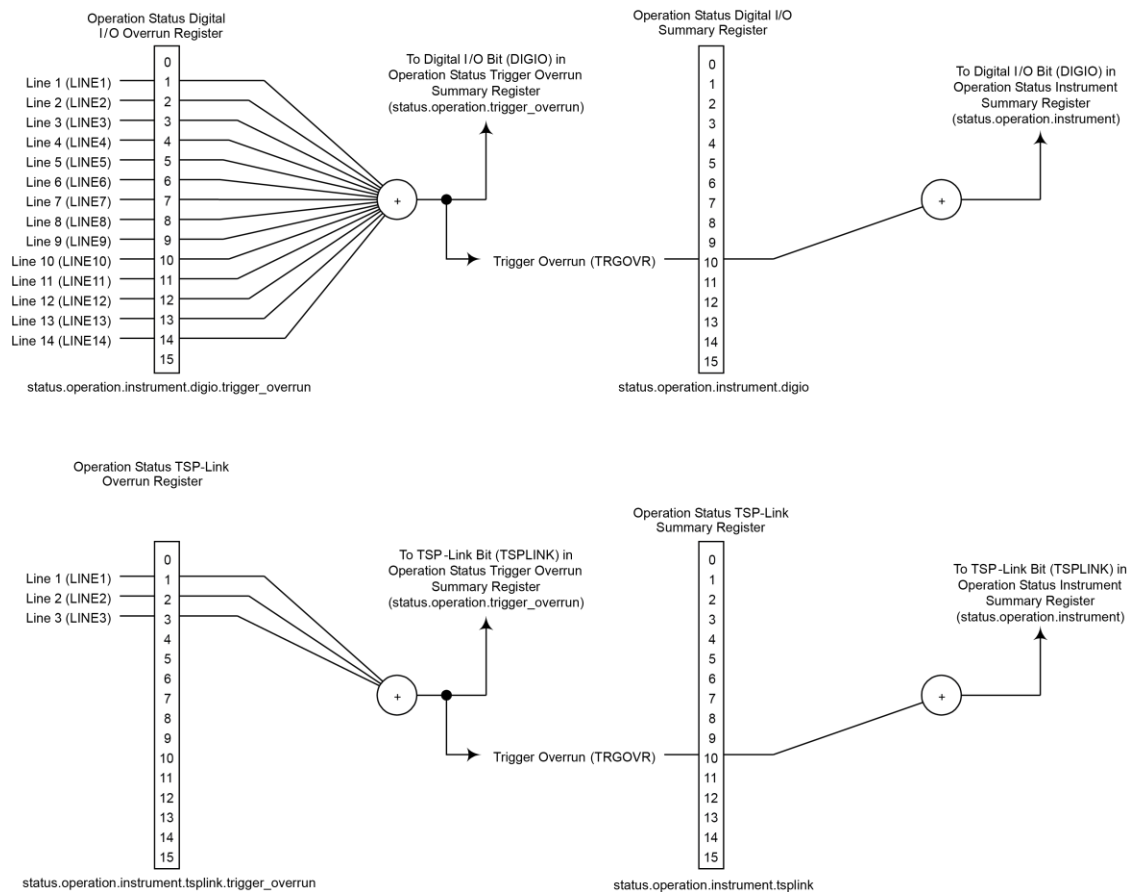
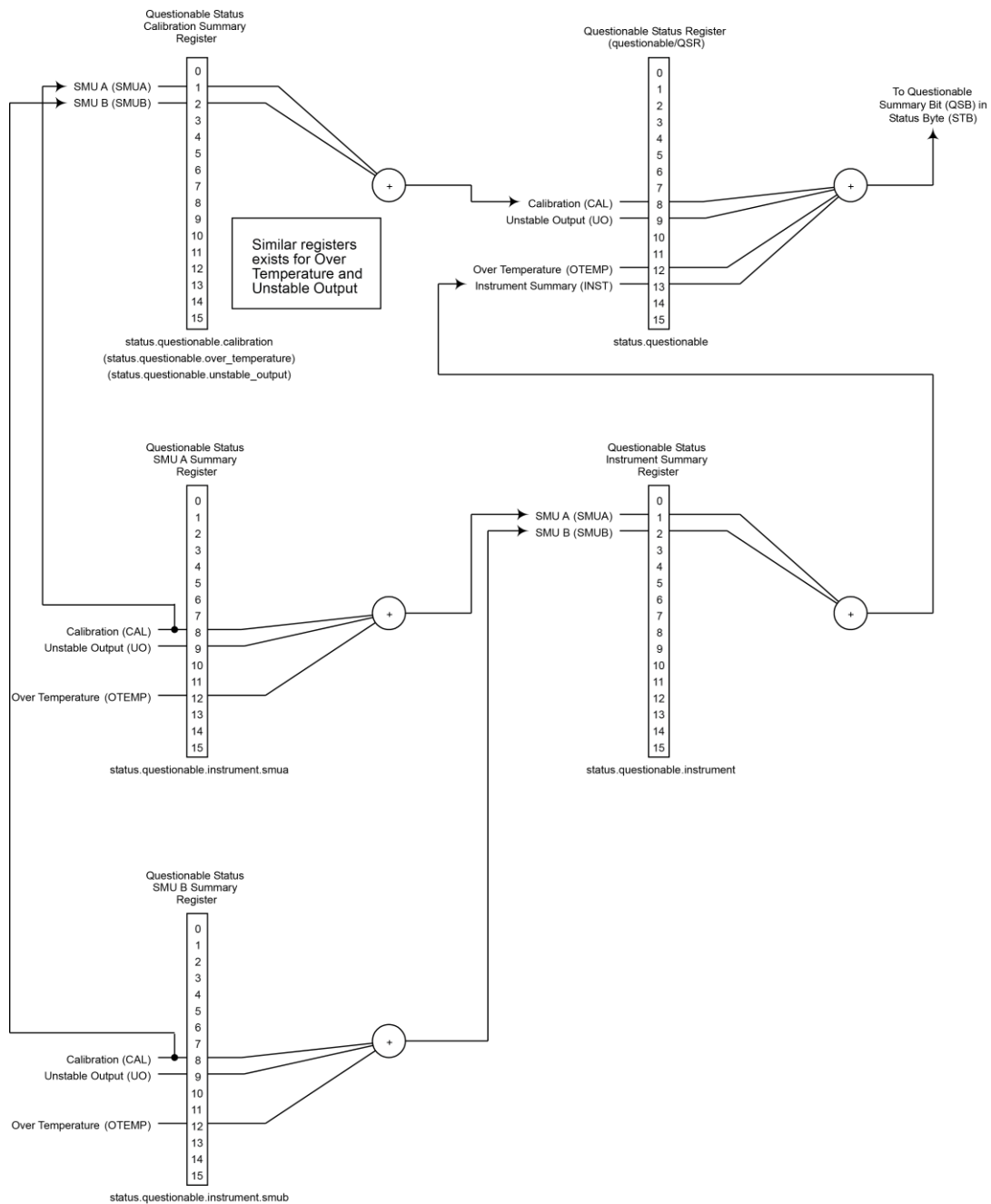
Figure 110: Operation status digital I/O and TSP-Link registers

Figure 111: Questionable status registers



Clearing registers

You can use commands to reset the status registers.

*CLS resets the bits of the event and NTR registers to 0 and sets all PTR register bits on. This command also clears the output queue.

`status.reset()` resets bits of the event and NTR registers to 0 and sets all PTR register bits on. Refer to [status.reset\(\)](#) (on page 15-339) for additional information.

In addition to these commands, you can reset the enable registers and the NTR to 0. To do this, send the individual command to program the register with a 0 as its parameter value. The PTR registers can be reset to their defaults by programming them with all bits on. The event registers are not programmable but you can clear them by reading them.

Programming enable and transition registers

The only registers that you can program are the enable and transition registers. All other registers in the status structure are read-only registers. The following explains how to determine the parameter values for the various commands used to program enable registers. The commands are summarized in [Common commands](#) (on page 16-1) and [Status function summary](#) (on page 10-3).

A command to program an event enable or transition register is sent with a parameter value that determines the state (0 or 1) of each bit in the appropriate register. The bit positions of the register (see the following figure) indicate the binary parameter value and decimal equivalent. To program one of the registers, send the decimal value for the bits to be set. The registers are discussed further in [Enable and transition registers](#) (on page 10-18).

Figure 112: 16-bit status register

Bit position	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

A. Bits 0 through 7

Bit position	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32768	16384	8192	4096	2048	1024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

B. Bits 8 through 15

When using a numeric parameter, registers are programmed by including the appropriate <mask> value. For example:

```
*ese 1169
status.standard.enable = 1169
```

To convert from decimal to binary, use the information shown in the above figure. For example, to set bits B0, B4, B7, and B10, a decimal value of 1169 would be used for the mask parameter (1169 = 1 + 16 + 128 + 1024).

Reading registers

Any register in the status structure can be read either by sending the common command query (where applicable), or by including the script command for that register in either the `print()` or `print(tostring())` command. The `print()` command outputs a numeric value; the `print(tostring())` command outputs the string equivalent. For example, any of the following commands requests the Service Request Enable Register value:

```
*SRE?  
print(tostring(status.request_enable))  
print(status.request_enable)
```

The response message is a decimal value that indicates which bits in the register are set. That value can be converted to its binary equivalent using the information in [Programming enable and transition registers](#) (on page 10-13). For example, for a decimal value of 37 (binary value of 100101), bits B5, B2, and B0 are set.

Status byte and service request (SRQ)

Service requests (SRQs) allow an instrument to indicate that it needs attention or that some event has occurred. When the controller receives an SRQ, it allows the controller to interrupt tasks to perform other tasks in order to address the request for service.

For example, you might program your instrument to send an SRQ when:

- All instrument operations are complete
- An instrument error occurs
- A specific operation has occurred

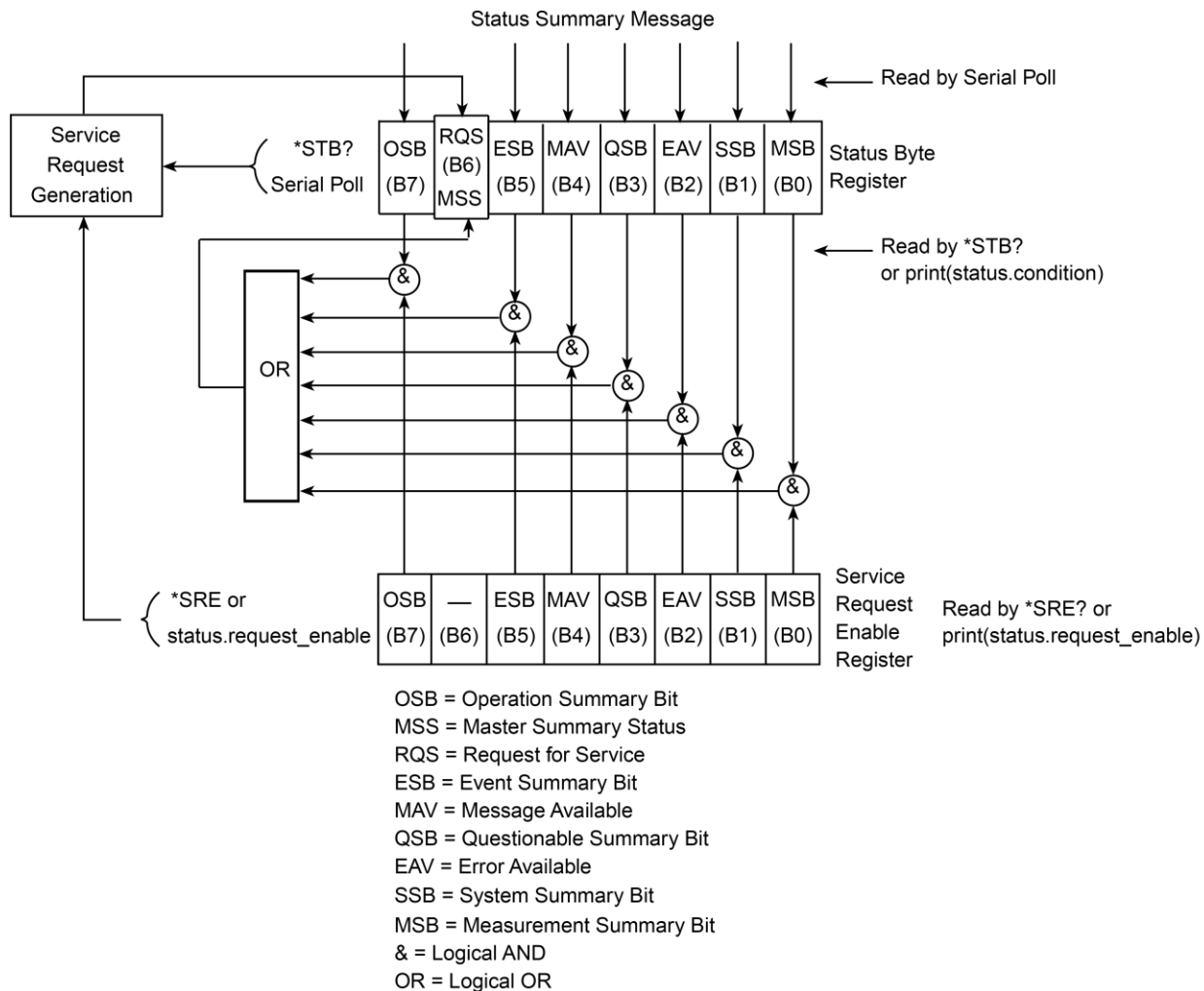
Two 8-bit registers control service requests, the Status Byte Register and the Service Request Enable Register. The Status Byte Register topic describes the structure of these registers.

Service requests affect USB and VXI-11 connections. An SRQ event is generated.

Status Byte Register

The summary messages from the status registers and queues are used to set or clear the appropriate bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are dependent upon 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 will reset to 0, which will then reset the ESB bit in the Status Byte Register.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status, or MSS, bit.

Figure 113: Status byte and service request (SRQ)

The bits of the Status Byte Register are:

- **Bit B0, Measurement Summary Bit (MSB):** When set, indicates that an enabled measurement event has occurred.
- **Bit B1, System Summary Bit (SSB):** When set, indicates that an enabled system event has occurred.
- **Bit B2, Error Available (EAV):** When set, indicates that an error or status message is present in the error queue.
- **Bit B3, Questionable Summary Bit (QSB):** When set, indicates that an enabled questionable event has occurred.
- **Bit B4, Message Available (MAV):** When set, indicates that a response message is present in the output queue.

- **Bit B5, Event Summary Bit (ESB):** When set, indicates that an enabled standard event has occurred.
- **Bit B6, Request Service (RQS)/Master Summary Status (MSS):** When set, indicates that an enabled summary bit of the Status Byte Register is set. Depending on how it is used, bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit:
 - When using a USB or VXI-11 serial poll sequence of the 2606B to obtain the status byte (serial poll byte), B6 is the RQS bit. See [Serial polling and SRQ](#) (on page 10-16) for details on using the serial poll sequence.
 - When using the `*STB?` common command or `status.condition` to read the status byte, B6 is the MSS bit.
- **Bit B7, Operation Summary (OSB):** When set, indicates that an enabled operation event has occurred.

Service Request Enable Register

The Service Request Enable Register controls the generation of a service request. 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 Status Byte Register, 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). The commands to program and read the SRQ Enable Register are listed in [Status byte and service request commands](#) (on page 10-17).

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 services 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 10-13).

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.

Status byte and service request commands

The commands to program and read the Status Byte Register and Service Request Enable Register are listed in the table below. Note that the table includes both common commands and their script command equivalents. For details on programming and reading registers, see [Programming enable and transition registers](#) (on page 10-13) and [Reading registers](#) (on page 10-14).

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`).

Status Byte and Service Request Enable Register commands

Command	Description
<code>*STB?</code> or <code>print(status.condition)</code>	Read the Status Byte Register.
<code>*SRE mask</code> or <code>status.request_enable = mask</code>	Program the Service Request Enable Register where <i>mask</i> = 0 to 255.
<code>*SRE?</code> or <code>print(status.request_enable)</code>	Read the Service Request Enable Register.

Enable and transition registers

In general, there are three types of user-writable registers that are used to configure which bits feed the register summary bit and when it occurs. The registers are identified in each applicable command (as listed in [TSP commands](#) (on page 15-7)) as follows:

- **Enable register** (identified as `.enable` in the command listing of each attribute): Allows various associated events to be included in the summary bit for the register.
- **Negative-transition register** (identified as `.ntr` in the command listing of each attribute): A particular bit in the event register is set when the corresponding bit in the NTR is set, and the corresponding bit in the condition register transitions from 1 to 0.
- **Positive-transition register** (identified as `.ptr` in the command listing of each attribute): A particular bit in the event register is set when the corresponding bit in the PTR is set, and the corresponding bit in the condition register transitions from 0 to 1.

Controlling node and SRQ enable registers

Attributes to control system node and service request (SRQ) enable bits and read associated registers are summarized in the [Status byte and service request enable registers](#) (on page 10-4). For example, either of the following commands set the system node QSB enable bit:

```
status.node_enable = status.QSB
status.node_enable = 8
```

Status register sets

There are five status register sets in the status structure of a System SourceMeter® instrument:

- System Summary
- Standard Event Status
- Operation Status
- Questionable Status
- Measurement Event

System Summary Registers

As shown in [Status model diagrams](#) (on page 10-4), there are five register sets associated with system status events. These registers summarize the system status for various nodes connected to the TSP-Link® network (see [TSP-Link system expansion interface](#) (on page 14-52)). Note that all nodes on the TSP-Link network share a copy of the system summary registers once the TSP-Link system has been initialized. This feature allows all nodes to access the status models of other nodes, including service request (SRQ).

In a TSP-Link system, you can configure the status model so that a status event in any node in the system can set the RQS (request for service) bit of the Master Node Status Byte. See [TSP-Link system status](#) (on page 10-24) for details on using the status model in a TSP-Link system.

Commands for the system summary registers are summarized in the [Status function summary](#) (on page 10-3) table.

For example, either of the following commands sets the EXT enable bit:

```
status.system.enable = status.system.EXT
status.system.enable = 1
```

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 [Reading registers](#) (on page 10-14). For example, the following command reads the System Enable Register:

```
print(status.system.enable)
```

The used bits of the system event registers are described as follows:

- **Bit B0, Extension Bit (EXT):** Set bit indicates that an extension bit from another system status register is set.
- **Bits B1 to B14 NODE_{*n*}:** Indicates a bit on TSP-Link node *n* has been set (*n* = 1 to 64) (note that `status.system5` does not use bits B9 through B15).

Standard Event Register

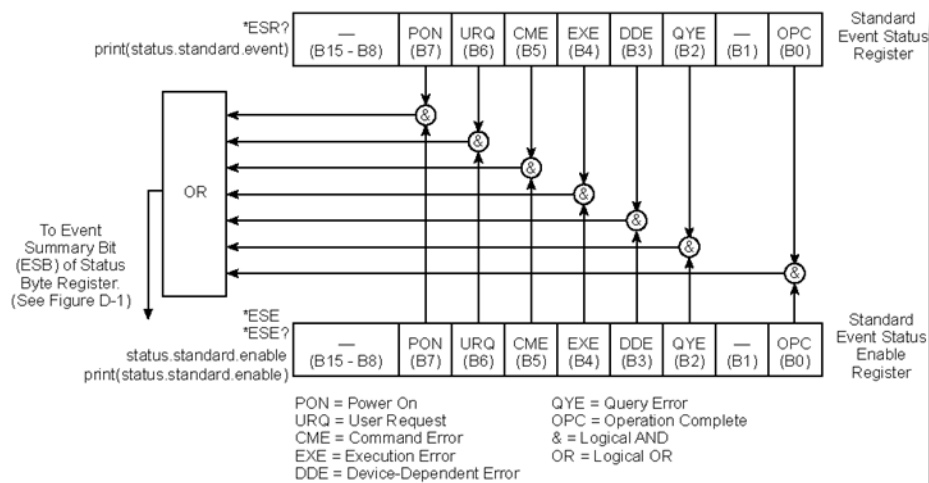
The bits used in the Standard Event Register are described as follows:

- **Bit B0, Operation Complete (OPC):** Set bit indicates that all pending selected device operations are completed and the 2606B instrument is ready to accept new commands. The bit is set in response to an *OPC command. The `opc()` function can be used in place of the *OPC command. See [Common commands](#) (on page 16-1) for details on the *OPC command.
- **Bit B1:** Not used.
- **Bit B2, Query Error (QYE):** Set bit indicates that you attempted to read data from an empty output queue.
- **Bit B3, Device-Dependent Error (DDE):** Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, Execution Error (EXE):** Set bit indicates that the 2606B instrument detected an error while trying to execute a command.

- **Bit B5, Command Error (CME):** Set bit indicates that a command error has occurred. Command errors include:
 - IEEE Std 488.2 syntax error: The 2606B instrument received a message that does not follow the defined syntax of IEEE Std 488.2.
 - Semantic error: 2606B instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented.
 - The instrument received a Group Execute Trigger (GET) inside a program message.
- **Bit B6, User Request (URQ):** Set bit indicates that the LOCAL key on the 2606B instrument front panel was pressed.
- **Bit B7, Power ON (PON):** Set bit indicates that the 2606B instrument has been turned off and turned back on since the last time this register was read.

Commands to program and read the register are summarized below and also in the [Status function summary](#) (on page 10-3) table.

Figure 114: Standard Event Register



Standard event commands

Command	Description
*ESR? or <code>print(status.standard.event)</code>	Read Standard Event Status Register.
*ESE <i>mask</i> or <code>status.standard.enable = mask</code>	Program the Event Status Enable Register: <i>mask</i> = 0 to 255 See Status register set contents (on page 10-1).
*ESE? or <code>print(status.standard.enable)</code>	Read Event Status Enable Register.

Operation Status Registers

As shown in the [Operation status registers](#) (on page 10-7) diagram of the status model, there are 22 register sets associated with operation status. Commands are summarized in [Status register set contents](#) (on page 10-1). You can also set bits using numeric parameter values. For details, see [Programming enable and transition registers](#) (on page 10-13).

For example, either of the following commands sets the CAL enable bit (B0):

```
status.operation.enable = status.operation.CAL
status.operation.enable = 1
```

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 [Reading registers](#) (on page 10-14). For example, the following command reads the Operation Status Enable Register:

```
print(status.operation.enable)
```

Commands to program and read the register are summarized in the table in [Status function summary](#) (on page 10-3).

This register set feeds to bit B7 (OSB) of the Status Byte. The bits used in the Operation Status Register set are described as follows:

- **Bit B0, Calibrating (CAL):** Set bit indicates that one or more channels are calibrating.
- **Bit B3, Sweeping (SWE):** Set bit indicates that one or more channels are sweeping.
- **Bit B4, Measuring (MEAS):** Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement.
- **Bit B10, Trigger Overrun (TRGOVR):** Set bit indicates that an enabled bit in the Operation Status Trigger Overrun Summary Register is set.
- **Bit B11, Remote Summary (REM):** Set bit indicates that an enabled bit in the Operation Status Remote Summary Register is set.
- **Bit B12, User (USER):** Set bit indicates that an enabled bit in the Operation Status User Register is set.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that an enabled bit in the Operation Status Instrument Summary Register is set.
- **Bit B14, Program Running (PROG):** Set bit indicates that a program is running.

For more information on the Operation Status Registers, refer to [Status register set contents](#) (on page 10-1) and the figures in this section.

Questionable Status Registers

This register set feeds to bit B3 (QSB) of the Status Byte. The bits used in the Questionable Status Register set are described as follows:

- **Bit B8, Calibration (CAL):** Set bit indicates that calibration is questionable.
- **Bit B9, Unstable Output (UO):** Set bit indicates that an unstable output condition was detected.
- **Bit B12, Over Temperature (OTEMP):** Set bit indicates that an over temperature condition was detected.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that a bit in the Questionable Status Instrument Summary Register is set.

Questionable Status Registers

As shown in the [Operation event, I/O, and TSP-Link registers](#) (on page 10-10) of the status model, there are seven register sets associated with Questionable Status. Commands are summarized in [Status byte and service request \(SRQ\)](#) (on page 10-14). You can also set bits by using numeric parameter values. For details, see [Programming enable and transition registers](#) (on page 10-13).

For example, either of the following commands sets the CAL enable bit (B8):

```
status.questionable.enable = status.questionable.CAL
status.questionable.enable = 256
```

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 [Reading registers](#) (on page 10-14). For example, the following command reads the Questionable Status Enable Register:

```
print(status.questionable.enable)
```

For more information about the Questionable Status Registers, refer to [Status register set contents](#) (on page 10-1) and the figures in this section.

Measurement Event Registers

As shown in the [Measurement event registers](#) (on page 10-22) of the status model, there are eight register sets associated with measurement event status. Commands are summarized in [Status register set contents](#) (on page 10-1). Note that bits can also be set by using numeric parameter values. For details, see [Programming enable and transition registers](#) (on page 10-13).

For example, either of the following commands sets the VOLTAGE_LIMIT enable bit:

```
status.measurement.enable = status.measurement.VOLTAGE_LIMIT
status.measurement.enable = 1
```

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 [Reading registers](#) (on page 10-14). For example, the following command reads the Measurement Event Enable Register:

```
print(status.measurement.enable)
```

This register set feeds to bit B0 (MSB) of the Status Byte. The bits used in the Measurement Event Registers are:

- **Bit B0, Voltage Limit (VLMT):** Set bit indicates that the voltage limit was exceeded. This bit is updated only when either a measurement is made or the `smuX.source.compliance` attribute is read.
- **Bit B1, Current Limit (ILMT):** Set bit indicates that the current limit was exceeded. This bit is updated only when either a measurement is made or the `smuX.source.compliance` attribute is read.
- **Bit B7, Reading Overflow (ROF):** Set bit indicates that an overflow reading has been detected.
- **Bit B8, Buffer Available (BAV):** Set bit indicates that there is at least one reading stored in either or both nonvolatile reading buffers.
- **Bit B11, Output Enable (OE):** Set bit indicates that output enable was asserted.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that a bit in the Measurement Instrument Summary Register is set.

Commands to program and read the register are summarized in the [Status function summary](#) (on page 10-3) table. For more information about the Measurement Event Registers, refer to [Status register set contents](#) (on page 10-1) and the figures in this section.

Register programming example

The command sequence below programs the instrument to generate a service request (SRQ) and set the system summary bit in all TSP-Link nodes when the current limit on channel A is exceeded.

```
-- Clear all registers.
status.reset()

-- Enable current limit bit in current limit register.
status.measurement.current_limit.enable = status.measurement.current_limit.SMUA

-- Enable status measure current limit bit.
status.measurement.enable = status.measurement.ILMT

-- Set system summary; enable MSB.
status.node_enable = status.MSB

-- Enable status SRQ MSB.
status.request_enable = status.MSB
```

TSP-Link system status

The TSP-Link® expansion interface allows instruments to communicate with each other. The test system can be expanded to include up to 32 TSP-enabled instruments. In a TSP-Link system, one node (instrument) is the master and the other nodes are the subordinates. The master can control the other nodes (subordinates) in the system. See [TSP-Link system expansion interface](#) (on page 14-52) for details about the TSP-Link system.

The system summary registers, shown in the [Status byte and service request enable register](#) (on page 10-4) and the [System summary and standard event registers](#) (on page 10-5), are shared by all nodes in the TSP-Link system. A status event that occurs at a subordinate node can generate an SRQ (service request) in the master node. After detecting the service request, your program can then branch to an appropriate subroutine that will service the request. See [Status byte and service request \(SRQ\)](#) (on page 10-14) for details.

Status model configuration example

In this example, a current limit (compliance) event in SMU A or B of node 15 sets the RQS bit of the Status Byte of the master node. The commands to configure the status model for this example are provided in [Status configuration \(enable\) commands](#) (on page 10-25).

When a current limit (compliance) condition occurs in SMU A or B of node 15, the following sequence of events occurs:

- Node 15: Bit B1 or B2 of the Measurement Event Current Limit Summary Register sets when the current limit (compliance) event occurs.
- Node 15: Bit B1 (ILMT) of the Measurement Event Register sets.
- Node 15: Bit B0 (MSB) of the Status Byte sets.
- System Summary Registers: Bit B1 (node 15) of the System Summary Register 2 sets.

NOTE

The System Summary Registers are shared by all nodes in the TSP-Link system. When a bit in a system register of node 15 sets, the same bit in the master node system register also sets.

- System Summary Registers: Bit B0 (Extension) of the System Summary Register sets.
- Master Node: Bit B0 (MSB) of the Status Byte sets.
- Master Node: With service request enabled, bit B6 (RQS) of the Status Byte sets. When your program performs the next serial poll of the master node, it detects the current limit event and can branch to a routine to service the request.

The figure in [Status configuration \(enable\) commands](#) (on page 10-25) demonstrates the flow of information through the status model of node 15 and the master node.

Status configuration (enable) commands

For the following registers, the commands listed, which are sent from the master node, enable the appropriate register bits for the status model configuration example.

Node 15 status registers: The following commands enable the current limit events for SMU A and B of node 15:

```
node[15].status.measurement.current_limit.enable = 6
node[15].status.measurement.enable = 2
node[15].status.node_enable = 1
```

The affected status registers for the above commands are indicated by labels A, B and C (see the following figure).

Master node system summary registers: The following commands enable the required system summary bits for node 15:

```
status.system2.enable = 2
status.system.enable = 1
```

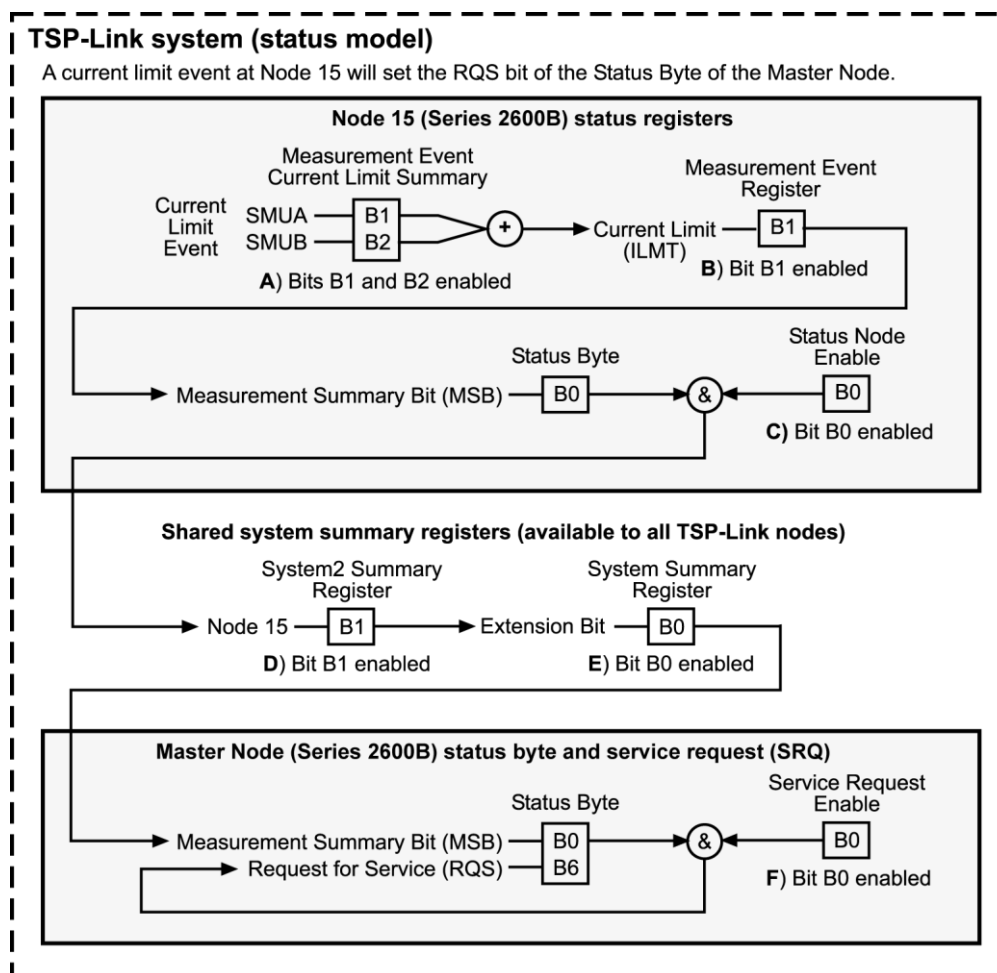
The affected system summary registers for the above commands are indicated by labels D and E (see the following figure).

Master node service request: The following command enables the service request for the measurement event:

```
status.request_enable = 1
```

The affected status register for the above command is indicated by label E (see the following figure).

Figure 115: TSP-Link status model configuration example



In this section:

Introduction	11-1
Line fuse replacement.....	11-1
Upgrading the firmware.....	11-2

Introduction

This section describes routine maintenance of the instrument that an operator can perform.

Line fuse replacement

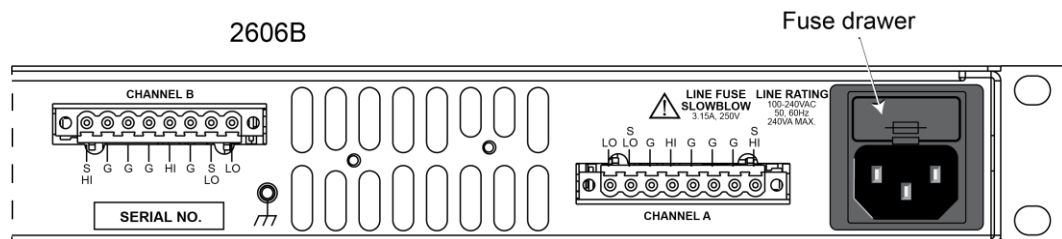
Fuses on the 2606B rear panel protect the power line input of the instrument.

⚠ 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.

The fuse drawer is above the AC receptacle on the rear panel of the 2606B, as shown in the following figure.

Figure 116: Fuse replacement



⚠ WARNING

To prevent injury, death, or instrument damage, use only the correct fuse type (see the following table).

To replace a line fuse:

1. Power off the instrument and remove the line cord.
2. There is a small tab at the bottom of the fuse drawer. Using a small screwdriver, pry this tab from the ac receptacle.
3. Slide the fuse drawer out to access to the fuses.
4. Snap the fuses out of the drawer and replace them with the same type, as specified in the following table.
5. Push the fuse drawer back into the module.

If the fuses continue to become damaged, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

Line fuses		
Line voltage	Rating	Keithley part number
100 V to 240 V	250 V, 6.3 A, Slow Blow 5 x 20 mm	FU-108-6.3, quantity 2

Upgrading the firmware

Use this procedure to upgrade the 2606B firmware for a module using a USB flash drive or a file on your computer. The upgrade process takes approximately five minutes. Each module must be updated independently.

You can upgrade the firmware using a USB flash drive on the virtual front panel, through the web interface, or using TSB.

Upgrade files are available for download from the Keithley [Product Support and Downloads web page \(tek.com/product-support\)](https://www.tek.com/product-support).

CAUTION

Disconnect the input and output terminals before you upgrade or downgrade.

Do not remove power from the instrument 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.

Do not initialize or reset TSP-Link before upgrading.

To upgrade or downgrade the firmware from the web interface:

1. Turn the instrument power off. Wait a few seconds.
2. Turn the instrument power on.
3. Open a web browser on the host computer.
4. Enter the IP address of the instrument in the web browser address box. For example, if the instrument IP address is 192.168.1.101, enter 192.168.1.101 in the browser address box.
5. Press **Enter** on the computer keyboard to open the web interface of the instrument.
6. From the left navigation area, select **Flash Upgrade**.
7. Choose **Select File**. A file selection dialog box is displayed.
8. Select the firmware file.
9. Select **Start**. A progress dialog box is displayed. When the upgrade begins, you can also see progress on the virtual front-panel display.

After the instrument automatically restarts, it is ready for use.

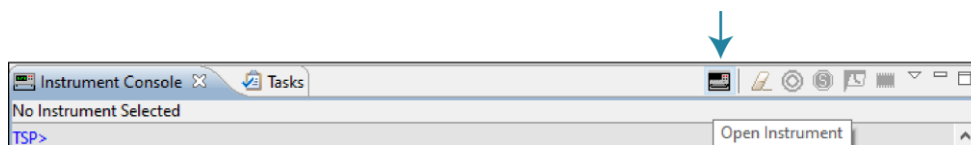
Using TSB to upgrade the firmware

After downloading the flash file from tek.com/keithley, you can use Test Script Builder (TSB) to upgrade the firmware of your 2606B.

To upgrade the firmware using Test Script Builder:

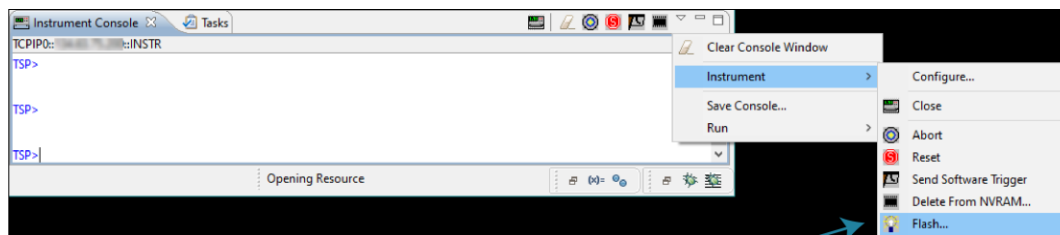
1. Start Test Script Builder.
2. On the Instrument Console toolbar, select the **Open Instrument** icon.

Figure 117: Open Instrument icon



3. Select your instrument from the **Select Instrument** dialog box.
4. On the Instrument Console toolbar, select the down arrow and select **Instrument > Flash**.

Figure 118: Flash menu option



5. For **Select or enter a firmware image file**, use the browser to select the new firmware.
6. For **Select a replacement mode**, select **Upgrade** to install a newer version of firmware or **Downgrade** to install an older of firmware.
7. Wait until the upgrade or downgrade is complete and the instrument shows the opening display.

Troubleshooting guide

In this section:

Introduction	12-1
Error levels	12-1
Effects of errors on scripts.....	12-1
Retrieving errors.....	12-2
Error summary list	12-2
LAN troubleshooting suggestions.....	12-6

Introduction

Troubleshooting information includes information on the Keithley Instruments 2606B errors (including a complete list of error messages) and LAN troubleshooting suggestions.

Error levels

Error messages are listed in [Error summary list](#) (on page 12-2). Errors have one of the following error levels:

Number	Error level	Description
0	NO_SEVERITY	The message is information only. This level is used when the error queue is empty; the message does not represent an error.
10	INFORMATIONAL	The message is information only. This level is used to indicate status changes; the message does not represent an error.
20	RECOVERABLE	The error was caused by improper use of the instrument or by conditions that can be corrected. This message indicates that an error occurred. The instrument is still operating normally.
30	SERIOUS	There is a condition that prevents the instrument from functioning properly. The message indicates that the instrument is presently operating in an error condition. If the condition is corrected, the instrument returns to normal operation.
40	FATAL	There is a condition that cannot be corrected that prevents the instrument from functioning properly. Disconnect the DUT and turn the power off and then on again. If the error is a hardware fault that persists after cycling the power, the instrument must be repaired.

Effects of errors on scripts

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

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

Retrieving errors

When errors occur, the error messages are placed in the error queue. Use `errorqueue` commands to request error message information. For example, the following commands request the complete set of information about the next message in the error queue. They return the code, message, severity, and node for that error:

```
errorCode, message, severity, errorNode = errorqueue.next()
print(errorCode, message, severity, errorNode)
```

The following table lists the commands associated with the error queue.

Remote commands associated with the error queue

Command	Description
errorqueue.clear() (on page 15-87)	Clear error queue of all errors
errorqueue.count (on page 15-88)	Number of messages in the error queue
errorqueue.next() (on page 15-88)	Request next error message from queue

Error summary list

Error summary

Error number	Error level	Error message
-430	RECOVERABLE	Query DEADLOCKED
-420	RECOVERABLE	Query UNTERMINATED
-410	RECOVERABLE	Query INTERRUPTED
-363	RECOVERABLE	Input buffer overrun
-360	RECOVERABLE	Communications error
-350	RECOVERABLE	Queue overflow
-315	RECOVERABLE	Configuration memory lost
-314	RECOVERABLE	Save/recall memory lost
-292	RECOVERABLE	Referenced name does not exist
-286	RECOVERABLE	TSP Runtime error
-285	RECOVERABLE	Program syntax
-282	RECOVERABLE	Illegal program name
-281	RECOVERABLE	Cannot create program
-225	RECOVERABLE	Out of memory or TSP Memory allocation error
-224	RECOVERABLE	Illegal parameter value
-222	RECOVERABLE	Parameter data out of range
-221	RECOVERABLE	Settings conflict
-220	RECOVERABLE	Parameter error
-211	RECOVERABLE	Trigger ignored
-203	RECOVERABLE	Command protected
-154	RECOVERABLE	String too long
-151	RECOVERABLE	Invalid string data
-144	RECOVERABLE	Character data too long
-141	RECOVERABLE	Invalid character data
-121	RECOVERABLE	Invalid character in number
-120	RECOVERABLE	Numeric data
-110	RECOVERABLE	Command header error
-109	RECOVERABLE	Missing parameter

Error summary

Error number	Error level	Error message
-108	RECOVERABLE	Parameter not allowed
-105	RECOVERABLE	Trigger not allowed
-104	RECOVERABLE	Data type error
0	NO_SEVERITY	Queue Is Empty
503	RECOVERABLE	Calibration overflow
601	RECOVERABLE	Reading buffer data lost
603	RECOVERABLE	Power on state lost
702	FATAL	Unresponsive digital FPGA
819	RECOVERABLE	Error parsing exponent
820	RECOVERABLE	Error parsing value
900	FATAL	Internal system error
1100	RECOVERABLE	Command unavailable
1101	RECOVERABLE	Parameter too big
1102	RECOVERABLE	Parameter too small
1103	RECOVERABLE	Min greater than max
1104	RECOVERABLE	Too many digits for param type
1105	RECOVERABLE	Too many parameters
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist
1109	RECOVERABLE	Menu name already exists
1110	FATAL	Analog supply failure: over temperature
1113	RECOVERABLE	Data too complex
1122	SERIOUS	Interlock or power supply failure
1200	RECOVERABLE	TSP-Link initialization failed
1202	RECOVERABLE	TSP-Link initialization failed
1203	RECOVERABLE	TSP-Link initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSP-Link initialization failed
1205	RECOVERABLE	TSP-Link initialization failed (no remote nodes found)
1206	RECOVERABLE	TSP-Link initialization failed
1207	RECOVERABLE	TSP-Link initialization failed
1208	RECOVERABLE	TSP-Link initialization failed
1209	RECOVERABLE	TSP-Link initialization failed
1210	RECOVERABLE	TSP-Link initialization failed (node ID conflict)
1211	RECOVERABLE	Node NN is inaccessible
1212	RECOVERABLE	Invalid node ID
1213	RECOVERABLE	TSP-Link session expired
1215	RECOVERABLE	Code execution requested within the local group
1216	RECOVERABLE	Remote execution requested on node in group with pending overlapped operations
1217	RECOVERABLE	Remote execution requested on node outside the local group
1218	RECOVERABLE	Operation allowed only when TSP-Link master
1219	RECOVERABLE	TSP-Link found fewer nodes than expected
1400	RECOVERABLE	Expected at least NN parameters
1401	RECOVERABLE	Parameter NN is invalid
1402	RECOVERABLE	User scripts lost
1403	RECOVERABLE	Factory scripts lost
1404	RECOVERABLE	Invalid byte order
1405	RECOVERABLE	Invalid ASCII precision
1406	RECOVERABLE	Invalid data format
1500	RECOVERABLE	Invalid baud rate setting

Error summary

Error number	Error level	Error message
1501	RECOVERABLE	Invalid parity setting
1502	RECOVERABLE	Invalid terminator setting
1503	RECOVERABLE	Invalid bits setting
1504	RECOVERABLE	Invalid flow control setting
1700	RECOVERABLE	Display area boundary exceeded
1800	RECOVERABLE	Invalid digital trigger mode
1801	RECOVERABLE	Invalid digital I/O line
2000	SERIOUS	Flash download error
2001	RECOVERABLE	Cannot flash with error in queue
2101	FATAL	Could not close socket
2102	RECOVERABLE	Lan configuration already in progress
2103	RECOVERABLE	Lan disabled
2104	RECOVERABLE	Socket error
2105	RECOVERABLE	Unreachable gateway
2106	RECOVERABLE	Could not acquire ip address
2110	RECOVERABLE	Could not resolve hostname
2111	RECOVERABLE	DNS name (FQDN) too long
2112	RECOVERABLE	Connection not established
2200	RECOVERABLE	File write error
2201	RECOVERABLE	File read error
2202	RECOVERABLE	Cannot close file
2203	RECOVERABLE	Cannot open file
2204	RECOVERABLE	Directory not found
2205	RECOVERABLE	File not found
2206	RECOVERABLE	Cannot read current working directory
2207	RECOVERABLE	Cannot change directory
2211	RECOVERABLE	File system error
2212	RECOVERABLE	File system command not supported
2213	RECOVERABLE	Too many open files
2214	RECOVERABLE	File access denied
2215	RECOVERABLE	Invalid file handle
2216	RECOVERABLE	Invalid drive
2217	RECOVERABLE	File system busy
2218	RECOVERABLE	Disk full
2219	RECOVERABLE	File corrupt
2220	RECOVERABLE	File already exists
2221	RECOVERABLE	File seek error
2222	RECOVERABLE	End-of-file error
2223	RECOVERABLE	Directory not empty
2400	RECOVERABLE	Invalid specified connection
2401	RECOVERABLE	Invalid timeout seconds (.001 to 30)
2402	RECOVERABLE	TSPnet remote error: XXX, where XXX explains the remote error
2403	RECOVERABLE	TSPnet failure
2404	RECOVERABLE	TSPnet read failure
2405	RECOVERABLE	TSPnet read failure, aborted
2406	RECOVERABLE	TSPnet read failure, timeout
2407	RECOVERABLE	TSPnet write failure
2408	RECOVERABLE	TSPnet write failure, aborted
2409	RECOVERABLE	TSPnet write failure, timeout
2410	RECOVERABLE	TSPnet max connections reached

Error summary

Error number	Error level	Error message
2411	RECOVERABLE	TSPnet connection failed
2412	RECOVERABLE	TSPnet invalid termination
2413	RECOVERABLE	TSPnet invalid reading buffer table
2414	RECOVERABLE	TSPnet invalid reading buffer index range
2415	RECOVERABLE	TSPnet feature only supported on TSP connections
2416	RECOVERABLE	TSPnet must specify both port and init
2417	RECOVERABLE	TSPnet disconnected by other side
2418	RECOVERABLE	TSPnet read input buffer overflow
2419	RECOVERABLE	Invalid format specifier
2420	RECOVERABLE	Termination locked while using TSP connection
2500	RECOVERABLE	Average delay must be at least NNN seconds
4900	RECOVERABLE	Reading buffer index NN is invalid
4903	RECOVERABLE	Reading buffer expired
4904	SERIOUS	ICX parameter count mismatch, %s (Line #%d)
4905	SERIOUS	ICX parameter invalid value, %s (Line #%d)
4906	SERIOUS	ICX invalid function id, %s (Line #%d)
5001	FATAL	SMU is unresponsive. Disconnect DUT and cycle power
5003	SERIOUS	Saved calibration constants corrupted
5004	RECOVERABLE	Operation conflicts with CALA sense mode
5005	RECOVERABLE	Value too big for range
5007	RECOVERABLE	Operation would exceed safe operating area of the instrument
5008	RECOVERABLE	Operation not permitted while OUTPUT is on
5009	SERIOUS	Unknown sourcing function
5010	SERIOUS	No such SMU function
5011	RECOVERABLE	Operation not permitted while cal is locked
5012	RECOVERABLE	Cal data not saved - save or restore before lock
5013	RECOVERABLE	Cannot save cal data - unlock before save
5014	RECOVERABLE	Cannot restore cal data - unlock before restore
5015	RECOVERABLE	Save to cal set disallowed
5016	RECOVERABLE	Cannot change cal date - unlock before operation
5017	RECOVERABLE	Cannot change cal constants - unlock before operation
5018	SERIOUS	Cal version inconsistency
5019	RECOVERABLE	Cannot unlock - invalid password
5021	SERIOUS	Cannot restore default calset. Using previous calset
5022	SERIOUS	Cannot restore previous calset. Using factory calset
5023	SERIOUS	Cannot restore factory calset. Using nominal calset
5024	SERIOUS	Cannot restore nominal calset. Using firmware defaults
5025	RECOVERABLE	Cannot set filter.count > 1 when measure.count > 1
5027	RECOVERABLE	Unlock cal data with factory password
5028	RECOVERABLE	Cannot perform requested operation while source autorange is enabled
5029	RECOVERABLE	Cannot save without changing cal adjustment date
5032	RECOVERABLE	Cannot change this setting unless buffer is cleared
5033	RECOVERABLE	Reading buffer not found within device
5038	RECOVERABLE	Index exceeds maximum reading
5040	RECOVERABLE	Cannot use same reading buffer for multiple overlapped measurements
5041	SERIOUS	Output Enable not asserted
5042	RECOVERABLE	Cannot perform requested action while an overlapped operation is in progress
5043	RECOVERABLE	Cannot perform requested operation while voltage measure autorange is enabled

Error summary

Error number	Error level	Error message
5044	RECOVERABLE	Cannot perform requested operation while current measure autorange is enabled
5045	RECOVERABLE	Cannot perform requested operation while filter is enabled
5046	SERIOUS	SMU too hot
5047	RECOVERABLE	Minimum timestamp resolution is 1us
5048	RECOVERABLE	Contact check not valid with HIGH-Z OUTPUT off
5049	RECOVERABLE	Contact check not valid while an active current source
5050	RECOVERABLE	I limit too low for contact check
5051	FATAL	Model number/SMU hardware mismatch. Disconnect DUT and cycle power
5053	RECOVERABLE	Unstable output detected - Measurements may not be valid
5055	RECOVERABLE	Cannot change adjustment date - change cal constants before operation
5059	RECOVERABLE	trigger.source.action enabled without configuration
5060	RECOVERABLE	trigger.measure.action enabled without configuration
5061	RECOVERABLE	Operation not permitted while OUTPUT is off
5062	SERIOUS	SMU overload. Automatic OUTPUT off.
5063	RECOVERABLE	Cannot perform requested operation while measure autozero is on
5064	RECOVERABLE	Cannot use reading buffer that collects source values
5065	RECOVERABLE	I range too low for contact check
5066	RECOVERABLE	source.offlimiti too low for contact check
5069	SERIOUS	Autorange locked for HighC mode

LAN troubleshooting suggestions

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

- Verify that the network cable is in the LAN port on the rear panel of the instrument, not one of the TSP-Link® ports (see the description in [Rear panel](#) (on page 3-2)).
- Verify that 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.
- Verify that the configuration information for the correct ethernet card was used during the setup procedure.
- Verify that the network card in the computer is enabled.
- Verify that the IP address of the instrument is compatible with the IP address on the computer.
- Verify that the subnet mask address of the instrument is the same as the subnet mask address of the computer.
- Turn the instrument power off, and then on. Wait at least 60 seconds for the network configuration to be completed. Verify that an IP address has been assigned to the instrument:
 1. Press the **MENU** key to display the MAIN MENU.
 2. Use the navigation wheel to select **LAN**. The LAN CONFIG menu is displayed.
 3. Select **STATUS**.
 4. Select **IP-ADDRESS**.
- Restart your computer.
- For more detail on LAN settings, see [Connecting to the LAN](#) (on page 9-2).

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

Introduction to TSP operation

In this section:

Introduction to TSP operation.....	13-1
About TSP commands	13-3
Factory scripts.....	13-19

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 do 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. 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 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 (). 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 the results to variables 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 or a predefined constant.

Example 1: Set an attribute using a number

```
beeper.enable = 0
```

This attribute controls the beeps that occur when front-panel controls are selected. Setting this attribute to 0 turns off the beeper.

Example 2: Set an attribute using a constant

```
format.data = format.REAL64
```

Using the constant `REAL64` sets the print format to double-precision floating-point format.

To read an attribute, you can use the attribute as the parameter of a function or assign it to another variable.

Example 3: Read an attribute using a function

```
print(format.data)
```

Reads the data format by passing the attribute to the print function. If the data format is set to 3, the output is:
3.000000e+00

This shows that the data format is set to double precision floating point.

Example 4: Read an attribute using a variable

```
fd = format.data
```

This reads the data format by assigning the attribute to a variable named `fd`.

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.

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 14-1).

If you need information about using the Lua programming language with the instrument, see [Fundamentals of programming for TSP](#) (on page 14-14).

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 15-7)).

Beeper control

The beeper commands allow you to enable or disable and sound the instrument beeper.

[beeper.beep\(\)](#) (on page 15-7)

[beeper.enable](#) (on page 15-8)

Bit manipulation and logic operations

The bit functions perform bitwise logic operations on two given numbers, and bit operations on one given number. Logic and bit operations truncate the fractional part of given numbers to make them integers.

Logic operations

The `bit.bitand()`, `bit.bitor()`, and `bit.bitxor()` functions in this group perform bitwise logic operations on two numbers. The Test Script Processor (TSP®) scripting engine performs the indicated logic operation on the binary equivalents of the two integers. This bitwise logic operation is performed on all corresponding bits of the two numbers. The result of a logic operation is returned as an integer.

Bit operations

The rest of the functions in this group are used for operations on the bits of a given number. You can use these functions to:

- Clear a bit
- Toggle a bit
- Test a bit
- Set a bit or bit field
- Retrieve the weighted value of a bit or field value

All these functions use an index parameter to specify the bit position of the given number. The least significant bit of a given number has an index of 1, and the most significant bit has an index of 32.

NOTE

The Test Script Processor (TSP) scripting engine stores all numbers internally as IEEE Std 754 double-precision floating-point values. The logical operations work on 32-bit integers. Any fractional bits are truncated. For numbers larger than 4294967295, only the lower 32 bits are used.

[bit.bitand\(\)](#) (on page 15-8)

[bit.bitor\(\)](#) (on page 15-9)

[bit.bitxor\(\)](#) (on page 15-10)

[bit.clear\(\)](#) (on page 15-10)

[bit.get\(\)](#) (on page 15-11)

[bit.getfield\(\)](#) (on page 15-12)

[bit.set\(\)](#) (on page 15-13)

[bit.setfield\(\)](#) (on page 15-14)

[bit.test\(\)](#) (on page 15-15)

[bit.toggle\(\)](#) (on page 15-16)

Data queue

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

The data queue in the Test Script Processor (TSP®) scripting engine is first-in, first-out (FIFO).

You can access data from the data queue even if a remote group or a node has overlapped operations in process.

[dataqueue.add\(\)](#) (on page 15-49)

[dataqueue.CAPACITY](#) (on page 15-50)

[dataqueue.clear\(\)](#) (on page 15-50)

[dataqueue.count](#) (on page 15-51)

[dataqueue.next\(\)](#) (on page 15-52)

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 14 lines. Each line can be at TTL logic state 1 (high) or 0 (low). See the pinout diagram in [Digital I/O port](#) (on page 7-37) for additional information.

There are commands to read and write to each individual bit, and commands to read and write to the entire port.

[digio.readbit\(\)](#) (on page 15-54)

[digio.readport\(\)](#) (on page 15-54)

[digio.trigger\[N\].assert\(\)](#) (on page 15-55)

[digio.trigger\[N\].clear\(\)](#) (on page 15-56)

[digio.trigger\[N\].EVENT_ID](#) (on page 15-56)

[digio.trigger\[N\].mode](#) (on page 15-57)

[digio.trigger\[N\].overrun](#) (on page 15-58)

[digio.trigger\[N\].pulsewidth](#) (on page 15-59)

[digio.trigger\[N\].release\(\)](#) (on page 15-59)

[digio.trigger\[N\].reset\(\)](#) (on page 15-60)

[digio.trigger\[N\].stimulus](#) (on page 15-61)

[digio.trigger\[N\].wait\(\)](#) (on page 15-62)

[digio.writebit\(\)](#) (on page 15-63)

[digio.writeport\(\)](#) (on page 15-63)

[digio.writeprotect](#) (on page 15-64)

Display

[display.clear\(\)](#) (on page 15-65)
[display.getannunciators\(\)](#) (on page 15-65)
[display.getcursor\(\)](#) (on page 15-67)
[display.getlastkey\(\)](#) (on page 15-68)
[display.gettext\(\)](#) (on page 15-69)
[display.inputvalue\(\)](#) (on page 15-70)
[display.loadmenu.add\(\)](#) (on page 15-72)
[display.loadmenu.catalog\(\)](#) (on page 15-73)
[display.loadmenu.delete\(\)](#) (on page 15-74)
[display.locallockout](#) (on page 15-74)
[display.menu\(\)](#) (on page 15-75)
[display.numpad](#) (on page 15-76)
[display.prompt\(\)](#) (on page 15-76)
[display.screen](#) (on page 15-78)
[display.sendkey\(\)](#) (on page 15-78)
[display.setcursor\(\)](#) (on page 15-79)
[display.settext\(\)](#) (on page 15-80)
[display.smuX.digits](#) (on page 15-82)
[display.smuX.limit.func](#) (on page 15-82)
[display.smuX.measure.func](#) (on page 15-83)
[display.trigger.clear\(\)](#) (on page 15-84)
[display.trigger.EVENT_ID](#) (on page 15-84)
[display.trigger.overrun](#) (on page 15-84)
[display.trigger.wait\(\)](#) (on page 15-85)
[display.waitkey\(\)](#) (on page 15-86)

Error queue

When errors and events occur, the error and status messages are placed in the error queue. Use the error queue commands to request error and status message information.

[errorqueue.clear\(\)](#) (on page 15-87)
[errorqueue.count](#) (on page 15-88)
[errorqueue.next\(\)](#) (on page 15-88)

Event log

You can use the event log to view specific details about LAN triggering events.

[eventlog.all\(\)](#) (on page 15-89)
[eventlog.clear\(\)](#) (on page 15-90)
[eventlog.count](#) (on page 15-90)
[eventlog.enable](#) (on page 15-91)
[eventlog.next\(\)](#) (on page 15-92)
[eventlog.overwritemethod](#) (on page 15-93)

File I/O

You can use the file I/O commands to open and close directories and files, write data, or to read a file on an installed USB flash drive. File I/O commands are organized into two groups:

- Commands that reside in the `fs` and `io` table, for example: `io.open()`, `io.close()`, `io.input()`, and `io.output()`. Use these commands to manage file system directories; open and close file descriptors; and perform basic I/O operations on a pair of default files (one input and one output).
- Commands that reside in the file descriptors (for example: `fileVar.seek()`, `fileVar.write()`, and `fileVar.read()`) operate exclusively on the file with which they are associated.

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 generally need to use a double backslash (\\) when you are creating scripts or sending commands to the instrument.

For basic information about navigation and directory listing of files on a flash drive, see [File system navigation](#) (on page 4-17).

File descriptor commands for file I/O use a colon (:) to separate the command parts rather than a period (.), like the `io` commands.

File descriptors cannot be passed between nodes in a TSP-Link® system, so the `io.open()`, `fileVar::read()`, and `fileVar::write` commands are not accessible to the TSP-Link system. However, the default input and output files mentioned above allow for the execution of many file I/O operations without any reference to a file descriptor.

[fileVar.close\(\)](#) (on page 15-94)

[fileVar.flush\(\)](#) (on page 15-95)

[fileVar.read\(\)](#) (on page 15-96)

[fileVar.seek\(\)](#) (on page 15-97)

[fileVar.write\(\)](#) (on page 15-99)

[fs.chdir\(\)](#) (on page 15-103)

[fs.cwd\(\)](#) (on page 15-104)

[fs.is_dir\(\)](#) (on page 15-104)

[fs.is_file\(\)](#) (on page 15-105)

[fs.mkdir\(\)](#) (on page 15-106)

[fs.readdir\(\)](#) (on page 15-107)

[fs.rmdir\(\)](#) (on page 15-107)

[io.close\(\)](#) (on page 15-117)

[io.flush\(\)](#) (on page 15-117)

[io.input\(\)](#) (on page 15-118)

[io.open\(\)](#) (on page 15-119)

[io.output\(\)](#) (on page 15-119)

[io.read\(\)](#) (on page 15-121)

[io.type\(\)](#) (on page 15-122)

[io.write\(\)](#) (on page 15-123)

[os.remove\(\)](#) (on page 15-167)

[os.rename\(\)](#) (on page 15-167)

The following standard I/O commands are not supported:

File	I/O
▪ <code>fileVar:lines()</code> ▪ <code>fileVar:setvbuf()</code>	▪ <code>io.lines()</code> ▪ <code>io.popen()</code>

Instrument identification

These commands store strings that describe the instrument.

[localnode.description](#) (on page 15-154)

[localnode.model](#) (on page 15-155)

[localnode.revision](#) (on page 15-160)

[localnode.serialno](#) (on page 15-161)

LAN and LXI

The LAN commands have options that allow you to review and configure network settings. The `lan.config.*` commands allow you to configure LAN settings over the remote interface.

NOTE

You must send `lan.applysettings()` for the configuration settings to take effect.

The `lan.status.*` commands help you determine the status of the LAN.

The `lan.trigger[N].*` commands allow you to set up and assert trigger events that are sent over the LAN.

Other LAN commands allow you to reset the LAN, restore defaults, check LXI domain information, and enable or disable the Nagle algorithm.

[lan.applysettings\(\)](#) (on page 15-125)

[lan.autoconnect](#) (on page 15-125)

[lan.config.dns.address\[N\]](#) (on page 15-126)

[lan.config.dns.domain](#) (on page 15-127)

[lan.config.dns.dynamic](#) (on page 15-127)

[lan.config.dns.hostname](#) (on page 15-128)

[lan.config.dns.verify](#) (on page 15-129)

[lan.config.duplex](#) (on page 15-129)

[lan.config.gateway](#) (on page 15-130)

[lan.config.ipaddress](#) (on page 15-131)

[lan.config.method](#) (on page 15-131)
[lan.config.speed](#) (on page 15-132)
[lan.config.subnetmask](#) (on page 15-133)
[lan.linktimeout](#) (on page 15-133)
[lan.lxidomain](#) (on page 15-134)
[lan.nagle](#) (on page 15-135)
[lan.reset\(\)](#) (on page 15-135)
[lan.restoredefaults\(\)](#) (on page 15-136)
[lan.status.dns.address\[N\]](#) (on page 15-137)
[lan.status.dns.name](#) (on page 15-137)
[lan.status.duplex](#) (on page 15-138)
[lan.status.gateway](#) (on page 15-139)
[lan.status.ipaddress](#) (on page 15-139)
[lan.status.macaddress](#) (on page 15-140)
[lan.status.port.dst](#) (on page 15-140)
[lan.status.port.rawsocket](#) (on page 15-141)
[lan.status.port.telnet](#) (on page 15-141)
[lan.status.port.vxi11](#) (on page 15-142)
[lan.status.speed](#) (on page 15-142)
[lan.status.subnetmask](#) (on page 15-143)
[lan.timedwait](#) (on page 15-143)
[lan.trigger\[N\].assert\(\)](#) (on page 15-144)
[lan.trigger\[N\].clear\(\)](#) (on page 15-144)
[lan.trigger\[N\].connect\(\)](#) (on page 15-145)
[lan.trigger\[N\].connected](#) (on page 15-146)
[lan.trigger\[N\].disconnect\(\)](#) (on page 15-146)
[lan.trigger\[N\].EVENT_ID](#) (on page 15-147)
[lan.trigger\[N\].ipaddress](#) (on page 15-147)
[lan.trigger\[N\].mode](#) (on page 15-148)
[lan.trigger\[N\].overrun](#) (on page 15-149)
[lan.trigger\[N\].protocol](#) (on page 15-150)
[lan.trigger\[N\].pseudostate](#) (on page 15-150)
[lan.trigger\[N\].stimulus](#) (on page 15-151)
[lan.trigger\[N\].wait\(\)](#) (on page 15-152)
[localnode.description](#) (on page 15-154)
[localnode.password](#) (on page 15-156)
[localnode.passwordmode](#) (on page 15-157)

Miscellaneous

[delay\(\)](#) (on page 15-53)
[exit\(\)](#) (on page 15-93)
[localnode.autolinefreq](#) (on page 15-153)
[localnode.linefreq](#) (on page 15-155)
[makegetter\(\)](#) (on page 15-162)
[makesetter\(\)](#) (on page 15-163)
[meminfo\(\)](#) (on page 15-163)
[opc\(\)](#) (on page 15-166)
[waitcomplete\(\)](#) (on page 15-407)

Parallel script execution

[dataqueue.add\(\)](#) (on page 15-49)
[dataqueue.CAPACITY](#) (on page 15-50)
[dataqueue.clear\(\)](#) (on page 15-50)
[dataqueue.count](#) (on page 15-51)
[dataqueue.next\(\)](#) (on page 15-52)
[node\[N\].execute\(\)](#) (on page 15-164)
[node\[N\].getglobal\(\)](#) (on page 15-165)
[node\[N\].setglobal\(\)](#) (on page 15-166)
[tsplink.group](#) (on page 15-377)
[tsplink.master](#) (on page 15-378)
[tsplink.node](#) (on page 15-378)

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 15-100)
[format.byteorder](#) (on page 15-100)
[format.data](#) (on page 15-102)
[localnode.prompts](#) (on page 15-158)
[localnode.prompts4882](#) (on page 15-159)
[localnode.showerrors](#) (on page 15-161)
[print\(\)](#) (on page 15-169)
[printbuffer\(\)](#) (on page 15-170)
[printnumber\(\)](#) (on page 15-171)

Reading buffer

Reading buffers capture measurements, ranges, instrument status, and output states of the instrument.

[bufferVar.appendmode](#) (on page 15-17)
[bufferVar.basetimestamp](#) (on page 15-18)
[bufferVar.cachemode](#) (on page 15-19)
[bufferVar.capacity](#) (on page 15-20)
[bufferVar.clear\(\)](#) (on page 15-21)
[bufferVar.clearcache\(\)](#) (on page 15-21)
[bufferVar.collectsourcevalues](#) (on page 15-22)
[bufferVar.collecttimestamps](#) (on page 15-23)
[bufferVar.fillcount](#) (on page 15-24)
[bufferVar.fillmode](#) (on page 15-25)

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statuses](#) (on page 15-34)
[bufferVar.timestampresolution](#) (on page 15-35)
[bufferVar.timestamps](#) (on page 15-36)
[savebuffer\(\)](#) (on page 15-176)
[smuX.buffer.getstats\(\)](#) (on page 15-193)
[smuX.buffer.recalculatestats\(\)](#) (on page 15-195)
[smuX.makebuffer\(\)](#) (on page 15-210)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.savebuffer\(\)](#) (on page 15-231)

Reset

Resets settings to their default settings.

[digio.trigger\[N\].reset\(\)](#) (on page 15-60)
[lan.reset\(\)](#) (on page 15-135)
[localnode.reset\(\)](#) (on page 15-159)
[reset\(\)](#) (on page 15-176)
[smuX.reset\(\)](#) (on page 15-231)
[timer.reset\(\)](#) (on page 15-361)
[trigger.blender\[N\].reset\(\)](#) (on page 15-364)
[trigger.timer\[N\].reset\(\)](#) (on page 15-373)
[tsplink.trigger\[N\].reset\(\)](#) (on page 15-387)

Saved setups

Use the saved setups commands to save or restore the configurations to or from the nonvolatile memory of the instrument or an installed USB flash drive. You can use the `setup.poweron` attribute to specify which setup is recalled when the instrument is turned on.

[setup.poweron](#) (on page 15-190)
[setup.recall\(\)](#) (on page 15-191)
[setup.save\(\)](#) (on page 15-192)

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 14-1).

- [exit\(\)](#) (on page 15-93)
- [makegetter\(\)](#) (on page 15-162)
- [makesetter\(\)](#) (on page 15-163)
- [script.anonymous](#) (on page 15-177)
- [script.delete\(\)](#) (on page 15-178)
- [script.factory.catalog\(\)](#) (on page 15-178)
- [script.load\(\)](#) (on page 15-179)
- [script.new\(\)](#) (on page 15-180)
- [script.newautorun\(\)](#) (on page 15-181)
- [script.restore\(\)](#) (on page 15-182)
- [script.run\(\)](#) (on page 15-182)
- [script.user.catalog\(\)](#) (on page 15-183)
- [scriptVar.autorun](#) (on page 15-184)
- [scriptVar.list\(\)](#) (on page 15-185)
- [scriptVar.name](#) (on page 15-185)
- [scriptVar.run\(\)](#) (on page 15-186)
- [scriptVar.save\(\)](#) (on page 15-187)
- [scriptVar.source](#) (on page 15-188)

SMU

- [localnode.linefreq](#) (on page 15-155)
- [smuX.abort\(\)](#) (on page 15-193)
- [smuX.buffer.getstats\(\)](#) (on page 15-193)
- [smuX.buffer.recalculatestats\(\)](#) (on page 15-195)
- [smuX.contact.check\(\)](#) (on page 15-207)
- [smuX.contact.r\(\)](#) (on page 15-208)
- [smuX.contact.speed](#) (on page 15-209)
- [smuX.contact.threshold](#) (on page 15-210)
- [smuX.makebuffer\(\)](#) (on page 15-210)
- [smuX.measure.autorangeY](#) (on page 15-211)
- [smuX.measure.autozero](#) (on page 15-212)
- [smuX.measure.count](#) (on page 15-215)
- [smuX.measure.delay](#) (on page 15-216)
- [smuX.measure.delayfactor](#) (on page 15-217)
- [smuX.measure.filter.count](#) (on page 15-217)
- [smuX.measure.filter.enable](#) (on page 15-218)
- [smuX.measure.filter.type](#) (on page 15-219)
- [smuX.measure.highcrangedelayfactor](#) (on page 15-220)
- [smuX.measure.interval](#) (on page 15-221)
- [smuX.measure.lowrangeY](#) (on page 15-222)
- [smuX.measure.nplc](#) (on page 15-223)

[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.measure.rangeY](#) (on page 15-224)
[smuX.measure.rel.enableY](#) (on page 15-226)
[smuX.measure.rel.levelY](#) (on page 15-227)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.measureYandstep\(\)](#) (on page 15-229)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.reset\(\)](#) (on page 15-231)
[smuX.savebuffer\(\)](#) (on page 15-231)
[smuX.sense](#) (on page 15-232)
[smuX.source.autorangeY](#) (on page 15-233)
[smuX.source.compliance](#) (on page 15-235)
[smuX.source.delay](#) (on page 15-236)
[smuX.source.func](#) (on page 15-237)
[smuX.source.highc](#) (on page 15-237)
[smuX.source.levelY](#) (on page 15-238)
[smuX.source.limitY](#) (on page 15-239)
[smuX.source.lowrangeY](#) (on page 15-240)
[smuX.source.offlimitY](#) (on page 15-241)
[smuX.source.offmode](#) (on page 15-242)
[smuX.source.output](#) (on page 15-243)
[smuX.source.outputenableaction](#) (on page 15-244)
[smuX.source.rangeY](#) (on page 15-245)
[smuX.source.settling](#) (on page 15-246)
[smuX.trigger.arm.count](#) (on page 15-247)
[smuX.trigger.arm.set\(\)](#) (on page 15-247)
[smuX.trigger.arm.stimulus](#) (on page 15-248)
[smuX.trigger.ARMED_EVENT_ID](#) (on page 15-249)
[smuX.trigger.autoclear](#) (on page 15-250)
[smuX.trigger.count](#) (on page 15-250)
[smuX.trigger.endpulse.action](#) (on page 15-252)
[smuX.trigger.endpulse.set\(\)](#) (on page 15-252)
[smuX.trigger.endpulse.stimulus](#) (on page 15-254)
[smuX.trigger.endsweep.action](#) (on page 15-255)
[smuX.trigger.IDLE_EVENT_ID](#) (on page 15-256)
[smuX.trigger.initiate\(\)](#) (on page 15-256)
[smuX.trigger.measure.action](#) (on page 15-257)
[smuX.trigger.measure.set\(\)](#) (on page 15-258)
[smuX.trigger.measure.stimulus](#) (on page 15-259)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)
[smuX.trigger.MEASURE_COMPLETE_EVENT_ID](#) (on page 15-262)
[smuX.trigger.PULSE_COMPLETE_EVENT_ID](#) (on page 15-262)
[smuX.trigger.source.action](#) (on page 15-263)
[smuX.trigger.source.limitY](#) (on page 15-264)
[smuX.trigger.source.linearY\(\)](#) (on page 15-265)
[smuX.trigger.source.listY\(\)](#) (on page 15-266)
[smuX.trigger.source.logY\(\)](#) (on page 15-267)
[smuX.trigger.source.set\(\)](#) (on page 15-268)
[smuX.trigger.source.stimulus](#) (on page 15-269)

[smuX.trigger.SOURCE_COMPLETE_EVENT_ID](#) (on page 15-270)

[smuX.trigger.SWEEP_COMPLETE_EVENT_ID](#) (on page 15-271)

[smuX.trigger.SWEEPING_EVENT_ID](#) (on page 15-271)

SMU calibration

[smuX.cal.adjustdate](#) (on page 15-196)

[smuX.cal.date](#) (on page 15-197)

[smuX.cal.due](#) (on page 15-198)

[smuX.cal.lock\(\)](#) (on page 15-199)

[smuX.cal.password](#) (on page 15-199)

[smuX.cal.polarity](#) (on page 15-200)

[smuX.cal.restore\(\)](#) (on page 15-201)

[smuX.cal.save\(\)](#) (on page 15-202)

[smuX.cal.state](#) (on page 15-203)

[smuX.cal.unlock\(\)](#) (on page 15-204)

[smuX.contact.calibratehi\(\)](#) (on page 15-204)

[smuX.contact.calibratelo\(\)](#) (on page 15-206)

[smuX.measure.calibrateY\(\)](#) (on page 15-214)

[smuX.source.calibrateY\(\)](#) (on page 15-234)

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.condition](#) (on page 15-273)

[status.measurement.*](#) (on page 15-275)

[status.measurement.buffer_available.*](#) (on page 15-277)

[status.measurement.current_limit.*](#) (on page 15-278)

[status.measurement.instrument.*](#) (on page 15-280)

[status.measurement.instrument.smuX.*](#) (on page 15-281)

[status.measurement.reading_overflow.*](#) (on page 15-284)

[status.measurement.voltage_limit.*](#) (on page 15-285)

[status.node_enable](#) (on page 15-286)

[status.node_event](#) (on page 15-288)

[status.operation.*](#) (on page 15-290)

[status.operation.calibrating.*](#) (on page 15-292)

[status.operation.instrument.*](#) (on page 15-293)

[status.operation.instrument.digio.*](#) (on page 15-296)

[status.operation.instrument.digio.trigger_overrun.*](#) (on page 15-297)

[status.operation.instrument.lan.*](#) (on page 15-299)

[status.operation.instrument.lan.trigger_overrun.*](#) (on page 15-301)

[status.operation.instrument.smuX.*](#) (on page 15-303)

[status.operation.instrument.smuX.trigger_overrun.*](#) (on page 15-306)

[status.operation.instrument.trigger_blender.*](#) (on page 15-308)

[status.operation.instrument.trigger_blender.trigger_overrun.*](#) (on page 15-309)

[status.operation.instrument.trigger_timer.*](#) (on page 15-311)

[status.operation.instrument.trigger_timer.trigger_overrun.*](#) (on page 15-312)

[status.operation.instrument.tsplink.*](#) (on page 15-315)
[status.operation.instrument.tsplink.trigger_overrun.*](#) (on page 15-316)
[status.operation.measuring.*](#) (on page 15-318)
[status.operation.remote.*](#) (on page 15-319)
[status.operation.sweeping.*](#) (on page 15-320)
[status.operation.trigger_overrun.*](#) (on page 15-322)
[status.operation.user.*](#) (on page 15-324)
[status.questionable.*](#) (on page 15-326)
[status.questionable.calibration.*](#) (on page 15-328)
[status.questionable.instrument.*](#) (on page 15-330)
[status.questionable.instrument.smuX.*](#) (on page 15-331)
[status.questionable.over_temperature.*](#) (on page 15-333)
[status.questionable.unstable_output.*](#) (on page 15-334)
[status.request_enable](#) (on page 15-336)
[status.request_event](#) (on page 15-338)
[status.reset\(\)](#) (on page 15-339)
[status.standard.*](#) (on page 15-340)
[status.system.*](#) (on page 15-342)
[status.system2.*](#) (on page 15-344)
[status.system3.*](#) (on page 15-347)
[status.system4.*](#) (on page 15-349)
[status.system5.*](#) (on page 15-351)

Time

[bufferVar.basetimestamp](#) (on page 15-18)
[bufferVar.collecttimestamps](#) (on page 15-23)
[bufferVar.timestampresolution](#) (on page 15-35)
[bufferVar.timestamps](#) (on page 15-36)
[delay\(\)](#) (on page 15-53)
[gettimezone\(\)](#) (on page 15-108)
[os.time\(\)](#) (on page 15-168)
[settime\(\)](#) (on page 15-188)
[settimezone\(\)](#) (on page 15-189)
[timer.measure.t\(\)](#) (on page 15-360)
[timer.reset\(\)](#) (on page 15-361)

Triggering

Triggering allows you to capture measurements when an input signal or combination of input signals meet a set of conditions that you set.

[digio.trigger\[N\].assert\(\)](#) (on page 15-55)
[digio.trigger\[N\].clear\(\)](#) (on page 15-56)
[digio.trigger\[N\].EVENT_ID](#) (on page 15-56)
[digio.trigger\[N\].mode](#) (on page 15-57)
[digio.trigger\[N\].overrun](#) (on page 15-58)
[digio.trigger\[N\].pulsewidth](#) (on page 15-59)
[digio.trigger\[N\].release\(\)](#) (on page 15-59)

[digio.trigger\[N\].reset\(\)](#) (on page 15-60)
[digio.trigger\[N\].stimulus](#) (on page 15-61)
[digio.trigger\[N\].wait\(\)](#) (on page 15-62)
[display.trigger.clear\(\)](#) (on page 15-84)
[display.trigger.EVENT_ID](#) (on page 15-84)
[display.trigger.overrun](#) (on page 15-84)
[display.trigger.wait\(\)](#) (on page 15-85)
[lan.trigger\[N\].assert\(\)](#) (on page 15-144)
[lan.trigger\[N\].clear\(\)](#) (on page 15-144)
[lan.trigger\[N\].connect\(\)](#) (on page 15-145)
[lan.trigger\[N\].connected](#) (on page 15-146)
[lan.trigger\[N\].disconnect\(\)](#) (on page 15-146)
[lan.trigger\[N\].EVENT_ID](#) (on page 15-147)
[lan.trigger\[N\].ipaddress](#) (on page 15-147)
[lan.trigger\[N\].mode](#) (on page 15-148)
[lan.trigger\[N\].overrun](#) (on page 15-149)
[lan.trigger\[N\].protocol](#) (on page 15-150)
[lan.trigger\[N\].pseudostate](#) (on page 15-150)
[lan.trigger\[N\].stimulus](#) (on page 15-151)
[lan.trigger\[N\].wait\(\)](#) (on page 15-152)
[smuX.trigger.arm.count](#) (on page 15-247)
[smuX.trigger.arm.set\(\)](#) (on page 15-247)
[smuX.trigger.arm.stimulus](#) (on page 15-248)
[smuX.trigger.ARMED_EVENT_ID](#) (on page 15-249)
[smuX.trigger.autoclear](#) (on page 15-250)
[smuX.trigger.count](#) (on page 15-250)
[smuX.trigger.endpulse.action](#) (on page 15-252)
[smuX.trigger.endpulse.set\(\)](#) (on page 15-252)
[smuX.trigger.endpulse.stimulus](#) (on page 15-254)
[smuX.trigger.endsweep.action](#) (on page 15-255)
[smuX.trigger.IDLE_EVENT_ID](#) (on page 15-256)
[smuX.trigger.initiate\(\)](#) (on page 15-256)
[smuX.trigger.measure.action](#) (on page 15-257)
[smuX.trigger.measure.set\(\)](#) (on page 15-258)
[smuX.trigger.measure.stimulus](#) (on page 15-259)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)
[smuX.trigger.MEASURE_COMPLETE_EVENT_ID](#) (on page 15-262)
[smuX.trigger.PULSE_COMPLETE_EVENT_ID](#) (on page 15-262)
[smuX.trigger.source.action](#) (on page 15-263)
[smuX.trigger.source.limitY](#) (on page 15-264)
[smuX.trigger.source.linearY\(\)](#) (on page 15-265)
[smuX.trigger.source.listY\(\)](#) (on page 15-266)
[smuX.trigger.source.logY\(\)](#) (on page 15-267)
[smuX.trigger.source.set\(\)](#) (on page 15-268)
[smuX.trigger.source.stimulus](#) (on page 15-269)
[smuX.trigger.SOURCE_COMPLETE_EVENT_ID](#) (on page 15-270)
[smuX.trigger.SWEEP_COMPLETE_EVENT_ID](#) (on page 15-271)
[smuX.trigger.SWEEPING_EVENT_ID](#) (on page 15-271)
[trigger.blender\[N\].clear\(\)](#) (on page 15-362)

[trigger.blender\[N\].EVENT_ID](#) (on page 15-362)
[trigger.blender\[N\].orenable](#) (on page 15-363)
[trigger.blender\[N\].overrun](#) (on page 15-363)
[trigger.blender\[N\].reset\(\)](#) (on page 15-364)
[trigger.blender\[N\].stimulus\[M\]](#) (on page 15-365)
[trigger.blender\[N\].wait\(\)](#) (on page 15-366)
[trigger.clear\(\)](#) (on page 15-367)
[trigger.EVENT_ID](#) (on page 15-367)
[trigger.generator\[N\].assert\(\)](#) (on page 15-368)
[trigger.generator\[N\].EVENT_ID](#) (on page 15-368)
[trigger.timer\[N\].clear\(\)](#) (on page 15-369)
[trigger.timer\[N\].count](#) (on page 15-369)
[trigger.timer\[N\].delay](#) (on page 15-370)
[trigger.timer\[N\].delaylist](#) (on page 15-371)
[trigger.timer\[N\].EVENT_ID](#) (on page 15-372)
[trigger.timer\[N\].overrun](#) (on page 15-372)
[trigger.timer\[N\].passthrough](#) (on page 15-373)
[trigger.timer\[N\].reset\(\)](#) (on page 15-373)
[trigger.timer\[N\].stimulus](#) (on page 15-374)
[trigger.timer\[N\].wait\(\)](#) (on page 15-376)
[trigger.wait\(\)](#) (on page 15-376)
[tsplink.trigger\[N\].assert\(\)](#) (on page 15-381)
[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)
[tsplink.trigger\[N\].EVENT_ID](#) (on page 15-383)
[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)
[tsplink.trigger\[N\].pulsewidth](#) (on page 15-386)
[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)
[tsplink.trigger\[N\].reset\(\)](#) (on page 15-387)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)
[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

TSP-Link

These functions and attributes allow you to set up and work with a system that is connected by a TSP-Link® network.

[tsplink.group](#) (on page 15-377)
[tsplink.master](#) (on page 15-378)
[tsplink.node](#) (on page 15-378)
[tsplink.readbit\(\)](#) (on page 15-379)
[tsplink.readport\(\)](#) (on page 15-379)
[tsplink.reset\(\)](#) (on page 15-380)
[tsplink.state](#) (on page 15-381)
[tsplink.trigger\[N\].assert\(\)](#) (on page 15-381)
[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)
[tsplink.trigger\[N\].EVENT_ID](#) (on page 15-383)
[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)

[tsplink.trigger\[N\].pulsewidth](#) (on page 15-386)

[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)

[tsplink.trigger\[N\].reset\(\)](#) (on page 15-387)

[tsplink.trigger\[N\].stimulus](#) (on page 15-388)

[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

[tsplink.writebit\(\)](#) (on page 15-390)

[tsplink.writeport\(\)](#) (on page 15-390)

[tsplink.writeprotect](#) (on page 15-391)

TSP-Net

The TSP-Net module provides a simple socket-like programming interface to Test Script Processor (TSP®) enabled instruments.

[tspnet.clear\(\)](#) (on page 15-392)

[tspnet.connect\(\)](#) (on page 15-393)

[tspnet.disconnect\(\)](#) (on page 15-394)

[tspnet.execute\(\)](#) (on page 15-395)

[tspnet.idn\(\)](#) (on page 15-396)

[tspnet.read\(\)](#) (on page 15-397)

[tspnet.readavailable\(\)](#) (on page 15-398)

[tspnet.reset\(\)](#) (on page 15-399)

[tspnet.termination\(\)](#) (on page 15-399)

[tspnet.timeout](#) (on page 15-400)

[tspnet.tsp.abort\(\)](#) (on page 15-401)

[tspnet.tsp.abortonconnect](#) (on page 15-401)

[tspnet.tsp.rtablecopy\(\)](#) (on page 15-402)

[tspnet.tsp.runscript\(\)](#) (on page 15-403)

[tspnet.write\(\)](#) (on page 15-404)

Userstrings

Use the functions in this group to store and retrieve user-defined strings in nonvolatile memory. These strings are stored as key-value pairs. The key is a unique identifier such as a part number or identification string.

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 15-404)

[userstring.catalog\(\)](#) (on page 15-405)

[userstring.delete\(\)](#) (on page 15-406)

[userstring.get\(\)](#) (on page 15-407)

Factory scripts

The 2606B is shipped with one or more factory scripts saved in its flash firmware memory. A factory script is made up of a number of functions. Some of them can be called from the virtual front-panel LOAD TEST menu. All of them can be called using remote programming.

As Keithley Instruments develops additional factory scripts, they will be made available on tek.com/keithley as a flash firmware upgrade for the 2606B. See [Upgrading the firmware](#) (on page 11-2) for instructions on upgrading the flash firmware of your 2606B instrument.

A factory script is similar to a user script, except a factory script is created by Keithley Instruments at the factory and is permanently stored in nonvolatile memory. The differences between a user script and a factory script include the following:

- A factory script cannot be deleted from nonvolatile memory.
- The script listing for a factory script can be retrieved and modified, but it will then be treated as a user script. A user script cannot be saved as a factory script.
- Factory scripts are not stored in global variables. The only references to factory scripts are in the `script.factory.scripts` attribute.
- The `script.factory.catalog()` function returns an iterator that can be used in a `for` loop to iterate over all the factory scripts.

Example

To retrieve the catalog listing for factory scripts, send:

```
for name in script.factory.catalog() do print(name) end
```

Running a factory script

Use either of the following commands to run a factory script:

```
script.factory.scripts.name()  
script.factory.scripts.name.run()
```

Where: *name* is the name of the factory script.

Example:

Run the factory script named `KISweep`.

```
script.factory.scripts.KISweep()
```

Running a factory script function from the virtual front-panel controls

1. Press the **LOAD** key.
2. Select **FACTORY**.
3. Select the function to run and press the **ENTER** key or navigation wheel.
4. Press the **RUN** key.
5. Follow the prompts on the virtual front panel to run the script.

Retrieving and modifying a factory script listing

The script listing for a factory script can be retrieved and modified. However, it cannot be saved as a factory script. The modified script can be saved as a user script using the same name or a new name.

An imported factory script can only be reloaded into the 2606B as a user script.

The following function retrieves a script listing. The script code is output with the shell keywords (`loadscript` or `loadandrunscript` and `endscript`):

```
script.factory.scripts.name.list()
```

Where: *name* is the name of the factory script.

An example that retrieves the script listing for a factory script named `KISweep`:

```
script.factory.scripts.KISweep.list()
```

KISweep factory script

The KISweep factory script provides simple sweep test programming and shows how to use the sweeping function.

This script is made up of the following functions. Access these functions from the virtual front panel or the remote interfaces. The following functions make up the KISweep factory script:

[SweepILinMeasureV\(\)](#) (on page 15-353)

[SweepIListMeasureV\(\)](#) (on page 15-354)

[SweepILogMeasureV\(\)](#) (on page 15-355)

[SweepVLinMeasureI\(\)](#) (on page 15-356)

[SweepVListMeasureI\(\)](#) (on page 15-358)

[SweepVLogMeasureI\(\)](#) (on page 15-359)

KIPulse factory script

The KIPulse factory script provides examples of how to generate pulses and to provide a simple pulsing interface. Pulses can be generated using the functions listed below.

NOTE

Please note the following information about the KIPulse factory script:

- This factory script only operates on the channels present in the instrument executing the pulse functions. These functions do not operate correctly if you attempt to access instrument channels over the TSP-Link® interface.
 - The KIPulse factory scripts are general purpose examples that may not be suitable for all use cases. Very short pulses (less than 1 ms pulse width) may require optimization of the examples provided by the factory script in order to achieve settled measurements.
 - The `PulseIMeasureV()` and `PulseVMeasureI()` functions may be accessed from the front panel. The remaining functions may only be accessed remotely.
-

Use the configuration [KIPulse tag parameter pulse functions](#) (on page 13-21) to configure a pulse train and assign the configuration to the *tag* parameter (use `QueryPulseConfig()` to inspect configured pulse trains). Use the initiation `InitiatePulseTest()` function to execute the pulse trains assigned to its *tag* arguments. The conditions listed in the table below must be true for these functions to execute successfully.

Conditions that must be true for successful function execution

Conditions for Config functions:	Conditions for InitiatePulseTest functions	Conditions for InitiatePulseTestDual functions
Source autorange (I and V) off	Output on	Output on
Measure autorange (I and V) off	There is enough free space in the buffer	There is enough free space in the buffer
Measure NPLC < ton	Buffer append mode is on when pulse train is >1 point	Buffer append mode is on when pulse train is >1 point
Measure autozero OFF or ONCE		Separate unique source-measure units (SMUs) for each tag Same NPLC setting for each tag Same toff for each tag

Use the [KIPulse simple pulse functions](#) (on page 13-21) to specify and perform a specified number of pulse-measure cycles.

The following functions make up the KIPulse factory script.

KIPulse tag parameter pulse functions

[ConfigPulseIMeasureV\(\)](#) (on page 15-37)

[ConfigPulseVMeasureI\(\)](#) (on page 15-43)

[ConfigPulseIMeasureVSweepLin\(\)](#) (on page 15-39)

[ConfigPulseVMeasureISweepLin\(\)](#) (on page 15-45)

[ConfigPulseIMeasureVSweepLog\(\)](#) (on page 15-41)

[ConfigPulseVMeasureISweepLog\(\)](#) (on page 15-47)

[InitiatePulseTest\(\)](#) (on page 15-113)

[InitiatePulseTestDual\(\)](#) (on page 15-115)

[QueryPulseConfig\(\)](#) (on page 15-174)

KIPulse simple pulse functions

[PulseIMeasureV\(\)](#) (on page 15-172)

[PulseVMeasureI\(\)](#) (on page 15-173)

Advanced features for KIPulse tag parameter pulse functions

Variable off time between pulses in a pulse train

The KIPulse “Configure” functions accept the *toff* parameter as a table or as a number. The table allows you to define different off times to be used after each pulse. Note the following:

- If *toff* is passed as a number or only a single value is used in the table, it is used for all points in a multiple point pulse.
- The number of times specified in the table must match the number of points called for in the sweep.
- The times used in tables must match for dual channel pulsing.
- Each specified off time must adhere to the duty cycle limits for the specified pulsing region.

Simultaneous IV measurement during pulse

The KIPulse “Configure” functions optionally accept an extra reading buffer to activate simultaneous IV measurements during pulsing. Previous usage of passing in a reading buffer or a nil (for no measurement) is still supported.

KIHighC factory script

The KIHighC factory script is made up of two functions: `i_leakage_measure()` and `i_leakage_threshold()`. These functions are intended to be used when high capacitance mode is active. Output is generally at a nonzero voltage before calling these functions. These functions can also be used to step the voltage to zero volts in order to measure the leakage current.

[i_leakage_measure\(\)](#) (on page 15-111)

[i_leakage_threshold\(\)](#) (on page 15-112)

KIParlib factory script

The KIParlib factory script is made up of two functions: `gm_vswEEP()` and `gm_ismEEP()`.

[gm_vswEEP\(\)](#) (on page 15-110)

[gm_ismEEP\(\)](#) (on page 15-109)

KISavebuffer factory script

The KISavebuffer script has one function: `savebuffer()`.

[savebuffer\(\)](#) (on page 15-176)

Instrument programming

In this section:

Fundamentals of scripting for TSP	14-1
Fundamentals of programming for TSP	14-14
Test Script Builder	14-32
Password management	14-36
Working with TSB Embedded	14-38
Advanced scripting for TSP	14-42
TSP-Link system expansion interface	14-52
TSP-Net	14-64

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 virtual front panel of the instrument that prompt the operator to enter parameters.

Runtime and nonvolatile memory storage of scripts

Scripts are loaded into the runtime environment of the instrument. From there, they can be stored in nonvolatile memory in the instrument.

The runtime 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 runtime environment, you can run and manage them from the virtual front panel of the instrument or from a computer. Information in the runtime 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 runtime environment when the instrument is turned on.

Scripts are placed in the runtime environment when:

- The instrument is turned on. All scripts that are saved to nonvolatile memory are copied to the runtime environment when the instrument is turned on.
- Loaded over a remote command interface.

For detail on the amount of memory available in the runtime environment, see [Memory considerations for the runtime environment](#) (on page 14-50).

NOTE

If you make changes to a script in the runtime environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See [Working with scripts in nonvolatile memory](#) (on page 14-10).

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 15-7)). Lua is a scripting language that is described in [Fundamentals of programming for TSP](#) (on page 14-14).

Commands that cannot be used in scripts

Though the instrument accepts the following commands, you cannot use these commands in scripts.

Commands that cannot be used in scripts

General commands	IEEE Std 488.2 common commands	
abort	*CLS	*RST
endflash	*ESE	*SRE
endscript	*ESE?	*SRE?
flash	*ESR?	*STB?
loadscript	*IDN?	*TRG
loadandrunscript	*OPC	*TST?
password	*OPC?	*WAI
restoreglobals		

Manage scripts

This section describes how to create scripts by sending commands over the remote interface.

Tools for managing scripts

To manage scripts, you can send messages to the instrument, use your own development tool or program, use Keithley Instruments Test Script Builder (TSB) software, or use TSB Embedded on the web interface of the instrument.

TSB software is a programming tool that you can download from the [Product Support and Downloads web page](https://www.tek.com/product-support) ([tek.com/product-support](https://www.tek.com/product-support)). You can use it to create, modify, debug, and store Test Script Processor (TSP®) scripting engine scripts. For more information about using the TSB software, see [Using Test Script Builder \(TSB\)](#) (on page 14-33).

TSB Embedded is a tool with a reduced set of features than the complete Keithley TSB software. TSB Embedded has both script-building functionality and console functionality (single-line commands). It is accessed from a web browser. Refer to [Working with TSB Embedded](#) (on page 14-38) for additional information.

NOTE

If you are using TSB or TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

Create and load a script

You create scripts by loading them into the runtime environment of the instrument. You can load a script as a named script or as the anonymous script.

Once a script is loaded into the instrument, you can execute it remotely or from the virtual front panel.

Anonymous scripts

If a script is created with the `loadscript` or `loadandrunscript` command with no name defined, it is called the anonymous script. There can only be one anonymous script in the runtime environment. If another anonymous script is loaded into the runtime environment, it replaces the existing anonymous script.

Named scripts

A named script is a script with a unique name. You can have as many named scripts as needed in the instrument (within the limits of the memory available to the runtime environment). When a named script is loaded into the runtime environment with the `loadscript` or `loadandrunscript` commands, a global variable with the same name is created to reference the script.

Key points regarding named scripts:

- If you load a new script with the same name as an existing script, the existing script becomes an unnamed script, which in effect removes the existing script if there are no variables that reference it.
- Sending revised scripts with different names does not remove previously loaded scripts.
- Named scripts can be saved to internal nonvolatile memory. Saving a named script to nonvolatile memory allows the instrument to be turned off without losing the script. See [Working with scripts in nonvolatile memory](#) (on page 14-10).

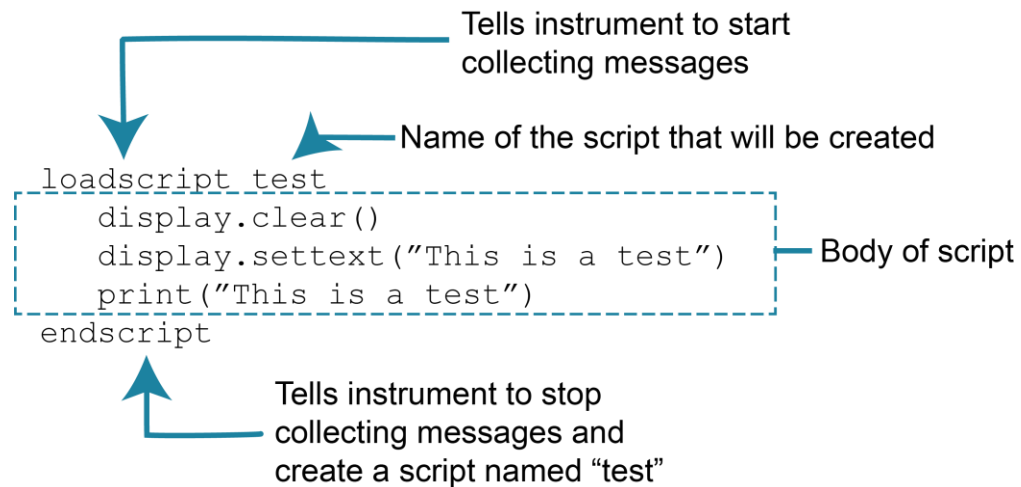
Load a script by sending commands over the remote interface

To load a script over the remote interface, you can use the `loadscript`, `loadandrunscript`, and `endscript` commands.

The `loadscript` and `loadandrunscript` commands start the collection of messages that make up the script. When the instrument receives either of these commands, it starts collecting all subsequent messages. Without these commands, the instrument runs them immediately as individual commands.

The `endscript` command tells the instrument to compile the collection of messages. It compiles the messages into one group of commands. This group of commands is loaded into the runtime environment.

The following figure shows an example of how to load a script named “test.” The first command tells the instrument to start collecting the messages for the script named “test.” The last command marks the end of the script. When this script is run, the message `This is a test` is displayed on the instrument and sent to the computer.

Figure 119: Loadscript and endscript example**To load a named 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. See [Run scripts](#) (on page 14-7).

NOTE

To run the script immediately, use `loadandrunscript scriptName` instead of `loadscript`.

Create a script using TSB Embedded**NOTE**

If you are using TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

You can create a script from the instrument web interface with TSB Embedded. When you save the script, it is loaded into the runtime environment and saved in the nonvolatile memory of the instrument.

To create a script using TSB Embedded:

1. In the **TSP Script** box, enter a name for the script.
2. In the input area, enter the sequence of commands to be included in the script.
3. Select **Save Script**. The name is added to the User Scripts list on the left.

Load a script from the instrument front panel

You can also load scripts from a USB flash drive to the runtime environment of the instrument. Depending on the content of the TSP file on the drive, the script can be loaded either as an anonymous script without a designated name, or as a named script with a user-defined name. Only named scripts can be saved to internal nonvolatile memory. Only one anonymous script can exist in the runtime environment.

To load a script into the instrument with a specific name, the TSP file must include the shell keywords `loadscript` and `endscript`, along with the specified script name, as shown in the example file `MyScript1.tsp`, which contains the script:

```
loadscript Beeper
reset()
beeper.enable = beeper.ON
beeper.beep(2, 2400)
endscript
```

When you load the file `MyScript1.tsp` from the flash drive, a script named `Beeper` is created in the runtime environment. Note that the script is named using the name that follows the `loadscript` keyword, not the name of the TSP file on the flash drive. After the script is loaded, you can choose to save it to nonvolatile memory.

If the loaded file does not contain `loadscript` and `endscript` keywords, or if no name is included after the `loadscript` keyword, the code is loaded as the anonymous script. Loading an unnamed script overwrites the existing anonymous script. For example, if a file named `MyScript2.tsp` contains only the following code, the script is loaded as the anonymous script:

```
reset()
beeper.enable = beeper.ON
beeper.beep(2, 2400)
```

The file must be a valid script file. If not, an error message is posted and no further action is taken. You can view the errors on the virtual front panel of the instrument.

To load a script from a USB flash drive:

1. Insert the flash drive into the USB port on the instrument.
2. Select the **MENU** key.
3. Select the **SCRIPT** option.
4. Select the **LOAD** option.
5. Select the **USB1** option. A menu is displayed that lists the TSP files and directories on the flash drive.
6. If the files are in a directory, use the navigation wheel to select the directory. A new menu is displayed that lists the TSP files and directories in that directory.
7. Use the navigation wheel to select the TSP file you want to load.

8. If the script has the same name as a script that is already in memory, you are prompted to overwrite the script.
 - Select **Yes** to continue.
 - Select **No** to return to the list of files. You must select a file to continue.
9. The SCRIPT ACTION menu is displayed. You can select:
 - **SAVE-INTERNAL:** Save the file to nonvolatile memory. This is the same as sending `scriptVar.save()` with no parameters.
 - **ACTIVE-FOR-RUN:** Set the script to run from the RUN button.
10. Loading is complete. To return to the MAIN menu, press **EXIT (LOCAL)** until the MAIN menu is displayed.
11. If you selected **ACTIVE-FOR-RUN**, you can select **RUN** to run the script.

NOTE

The entries in the SCRIPT ACTION menu depend on whether the script that was loaded is a named script or the anonymous script. If it is a named script, both SAVE-INTERNAL and ACTIVE-FOR-RUN appear in the menu. If it is the anonymous script, then only ACTIVE-FOR-RUN appears in the menu.

Run scripts

This section describes how to run the anonymous and named scripts.

On the virtual front panel, items are available through the USER menu if you explicitly add them to the menu. The items the menu selections represent can be scripts, function calls, or instrument commands. Items in the menus are referred to as scripts in this section.

The SCRIPTS menu lists the names of scripts in nonvolatile memory or scripts that have been added to the runtime environment. The anonymous script also appears in this menu.

NOTE

If the instrument is in local control when the script is started, it switches to remote control (**REM** is displayed) while the script is running. The instrument is returned to local control when the script completes. If you press the virtual front-panel **EXIT (LOCAL)** key while the script is running, the script is stopped.

Run the anonymous script

The anonymous script can be run many times without reloading it. It remains in the runtime environment until a new anonymous script is created or until the instrument is turned off.

To run the anonymous script, use any one of these commands:

- `run()`
- `script.run()`
- `script.anonymous()`
- `script.anonymous.run()`

Run a named script

You can run any named script that is in the runtime environment using one of the following commands:

- `scriptVar()`
- `scriptVar.run()`

Where `scriptVar` is the user-defined name of the script.

When a script is named, it can be accessed using the global variable `scriptVar`.

To run a named script from TSB Embedded, select the script from the User Scripts list and select **Run**.

Example: Run a named script

```
test3()
```

If the script `test3` is loaded into the runtime environment, the instrument executes `test3`.

Run a user script from the instrument front panel

From the virtual front panel, you can load and run a script that was previously added to the USER menu.

To run the code from the virtual front panel and add it to the USER menu:

1. Select the **LOAD** key.
2. Select **USER**.
3. Select the script from list and press the **ENTER** key. The script is loaded into the runtime environment.

NOTE

If you are used to using `print` in Test Script Builder, note that the output of the print commands using this procedure do not function the same as when you are in Test Script Builder. You may find that it makes more sense to use Test Script Builder to get the output you need.

4. Press the **RUN** key to execute.

To run a script directly without adding it to the USER menu:

1. Select the **LOAD** key.
2. Select **SCRIPTS** and select the **ENTER** key. There may be a short pause before a menu is displayed that represents the scripts in the instrument.
3. Select the script from the list and select the **ENTER** key. The script is now loaded for front-panel execution.
4. Press the **RUN** key to execute.

NOTE

If you are used to using `print` in Test Script Builder, note that the output of the prints using this procedure do not function the same as when you are in Test Script Builder. You may find that it makes more sense to use Test Script Builder to get the output you need.

Scripts that run automatically

You can set up scripts to run automatically when you power on the instrument. To do this, either set the `autorun` attribute for the script to `yes` (see [Autorun scripts](#) (on page 14-9)) or create a script with the script name `autoexec` (see [Autoexec script](#) (on page 14-10)).

Autorun scripts

Autorun scripts run automatically when the instrument is turned on. You can set any number of scripts to autorun. The run order for autorun scripts is arbitrary, so make sure the run order is not important.

As shown in the example below, you can set a script to run automatically by setting the `.autorun` attribute of the script to `"yes"` and then saving the script.

Example:

```
scriptVar.autorun = "yes"
scriptVar.save()
```

Where: `scriptVar` is the user-defined name of the script.

To disable autorun, set the `autorun` attribute of the script to `no` and then save the script.

NOTE

The `scriptVar.save()` command saves the script to nonvolatile memory, which makes the change persistent through a power cycle. See [Save a user script to nonvolatile memory](#) (on page 14-11) for more detail.

Example: Set a script to run automatically

```
test5.autorun = "yes"
test5.save()
```

Assume a script named `test5` is in the runtime environment.

The next time the instrument is turned on, `test5` script automatically loads and runs.

Autoexec script

The autoexec script runs automatically when the instrument is turned on. It runs after all the scripts have loaded and any scripts defined as autorun have run.

To create a script that executes automatically, create and load a new script and name it `autoexec`. See [Create and load a script](#) (on page 14-3).

NOTE

You must save the `autoexec` script to nonvolatile memory if you want to use it after instrument power has been turned off and then turned on again. See [Save a user script to nonvolatile memory](#) (on page 14-11) for more detail.

Example: Creating an autoexec script with the loadscript command

```
loadscript autoexec
display.clear()
display.settext("Hello from autoexec")
endscript
autoexec.save()
```

Creates the script `autoexec`.

Saves the `autoexec` script to nonvolatile memory. The next time the instrument is turned on, `Hello from autoexec` is displayed.

Example: Creating an autoexec script using TSB Embedded

```
display.clear()
display.settext("Hello from autoexec")
```

In the TSP Script box, enter `autoexec`.

Enter the code in the entry box.

Select **Save Script**.

Creates a new script that clears the display when the instrument is turned on and displays `Hello from autoexec`.

Working with scripts in nonvolatile memory

The [Fundamentals of scripting for TSP](#) (on page 14-1) section in this manual describes working with scripts, primarily in the runtime environment. You can also work with scripts in nonvolatile memory.

The runtime environment and nonvolatile memory are separate storage areas in the instrument. The information in the runtime environment is lost when the instrument is turned off. The nonvolatile memory remains intact when the instrument is turned off. When the instrument is turned on, information in nonvolatile memory is loaded into the runtime environment.

Save a user script

You can save scripts to nonvolatile memory using commands or TSB Embedded.

Only named scripts can be saved to nonvolatile memory. The anonymous script must be named before it can be saved to nonvolatile memory.

NOTE

If a script is not saved to nonvolatile memory, the script is lost when the instrument is turned off.

To save a script to nonvolatile memory from a remote interface:

1. Create and load a named script (see [Create and load a script](#) (on page 14-3)).
2. Send the command `scriptVar.save()`, where `scriptVar` is the name of the script.

To save a script to nonvolatile memory using TSB Embedded:

Select **Save 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.

To save a script to an external USB flash drive using a remote interface:

NOTE

When you save a script to a USB flash drive, you do not need to specify a file extension. The extension `.tsp` is automatically added. If you do specify a file extension, it must be `.tsp`. An error occurs if you use any other file extension.

1. Load a script (see [Create and load a script](#) (on page 14-3)).
2. Send the command `scriptVar.save("/usb1/filename.tsp")`, where `scriptVar` is the variable referencing the script and `filename.tsp` is the name of the file.

To save a script to an external USB flash drive or other accessible drive using TSB Embedded:

Load the script and select **Export to PC**.

Save the anonymous script as a named script

To save the anonymous script to nonvolatile memory, you must name it first.

To save the anonymous script as a named script:

1. To name the script, send the command `script.anonymous.name = "myTest"` (where `myTest` is the name of the script).
2. Send the `script.anonymous.save()` command to save `myTest` to nonvolatile memory.

Save a script from the instrument front panel

You can save scripts from the runtime environment to nonvolatile memory or the USB port on the instrument front panel.

NOTE

If you want to save the anonymous script to nonvolatile memory, you must name it first. See [Save the anonymous script as a named script](#) (on page 14-11).

To save a script to nonvolatile memory from the front panel:

1. Select the **MENU** key.
2. Select the **SCRIPT** option.
3. Select the **SAVE** option.
A list of the scripts available to save is displayed. It may take a few seconds to display. The displayed list is from the `script.user.scripts` table in the instrument.
4. Turn the navigation wheel to select the script that you want to save.
5. Select **INTERNAL**. Press the navigation wheel. The script is saved to nonvolatile memory using the name attribute of the script.
6. Press **EXIT (LOCAL)** several times to return to the Main Menu.

Delete user scripts

NOTE

These steps remove a script from nonvolatile memory. To completely remove a script from the instrument, there are additional steps you must take. See [Delete user scripts from the instrument](#) (on page 14-49).

To delete a script from nonvolatile memory using a remote interface:

You can delete the script from nonvolatile memory by sending either of the following commands:

- `script.delete("name")`
- `script.user.delete("name")`

Where: *name* is the user-defined name of the script.

Example: Delete a user script from nonvolatile memory

```
script.delete("test8")
```

Delete a user script named `test8` from nonvolatile memory.

To delete a script from nonvolatile memory using TSB Embedded:

1. In TSB Embedded, select the script from the **User Scripts** list.
2. Select **Delete**. There is no confirmation message.

Programming example: Interactive script

An interactive script prompts the operator to input values using virtual front panel controls. The following example script uses display messages to prompt the operator to:

- Enter the voltage level to source
- Enable or disable measurements
- Set the number of readings if measurements are enabled

After the operator completes entering values, the output turns on and sources the specified value. If measurements were enabled, a message indicates that measurements are in progress. Another message is displayed when the source-measure operation is complete. If measurements were not enabled, the message indicates that the source operation is complete.

When an input prompt is displayed, the script waits until the operator inputs the parameter or presses the ENTER key. The example shown here assumes that you are using TSB or TSB Embedded. If you are using a remote interface, you need to add the `loadscript` and `endscript` commands to the example code. See [Load a script by sending commands over the remote interface](#) (on page 14-4) for details.

```

reset()
-- Clear the display.
display.clear()
-- Prompt the user for a voltage value to source.
srcVoltage = display.prompt("+000.00", " V", "Enter source voltage", 5, -20, 20)
-- Prompt the user to enable measurements.
measEnable = display.menu("Measurements?", "ENABLE DISABLE")
if measEnable == "ENABLE" then
    -- Prompt the user for the number of measurements.
    numMeas = display.prompt("0000", " Rdgs", "Enter the number of readings", 10,
        0, 9999)
    smua.measure.count = numMeas
    smua.nvbuffer1.clear()
end
-- Convert user input to the source level.
smua.source.levelv = tonumber(srcVoltage)
smua.source.output = smua.OUTPUT_ON
if measEnable == "ENABLE" then
    -- Inform the user that measurements are in progress
    display.setcursor(1, 1)
    display.settext("$BPlease wait.$R$NMeasure operation in progress.")
    smua.measure.i(smua.nvbuffer1)
    display.clear()
    -- Inform the user that the Source Measure operation has finished
    display.settext("Operation Finished$NSource-Measure Complete")
else
    -- Inform the user that the Source operation has finished
    display.settext("Operation Finished$NSource Complete")
end
-- Wait 5 seconds then return to the main screen.
delay(5)
display.screen = display.SMUA

```

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 use 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 \(lua.org\)](http://lua.org). Another source of useful information is the [Lua users group \(lua-users.org\)](http://lua-users.org), created for and by users of Lua programming language.

Comments

You can start a comment anywhere outside a string by typing a double hyphen (--). If the text immediately after -- is anything other than double left brackets ([[), the comment is a short comment, which continues until the end of the line.

If -- is followed by [[, the following characters are a long comment, which continues until double right brackets (]]) close the comment. Long comments may continue for several lines and may contain nested [[...]] pairs. The example below shows how to use code comments.

An example of a short comment is:

```
-- Turn off the virtual front-panel display.
```

An example of a long comment is:

```
--[[Display a menu with three menu items. If the second menu item is selected,  
the selection is given the value Test2.]]
```

Function and variable name restrictions

You cannot use factory script names, functions created by factory scripts, Lua reserved words and top-level command names for function or variable names.

For information on factory script names, see [Factory scripts](#) (on page 13-19).

You cannot use the following Lua reserved words for function or variable names.

Lua reserved words

and	for	or
break	function	repeat
do	if	return
else	in	then
elseif	local	true
end	nil	until
false	not	while

You also cannot use top-level command names as variable names. If you use these names, it results in the loss of use of the commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until you turn the instrument power off and then turn it on again.

These names include:

Top level command names			
beeper	gcinfo	os	smub
bit	gettimezone	print	status
collectgarbage	gpib	printbuffer	string
dataqueue	io	printnumber	timer
delay	lan	reset	tonumber
digio	localnode	savebuffer	tostring
display	makegetter	script	trigger
errorqueue	makesetter	serial	tsplink
eventlog	math	settime	tspnet
exit	meminfo	settimezone	type
format	node	setup	userstring
fs	opc	smua	waitcomplete

Values and variable types

In Lua, you use variables to store values in the runtime 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 <code>nil</code> , 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 <code>false</code> and <code>true</code> . In Lua, both <code>nil</code> and <code>false</code> make a condition <code>false</code> ; any other value makes it <code>true</code> .
"number"	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers.
"string"	sequence of words or characters	
"function"	a block of code	Functions perform a task or compute and return values.
"table"	an array	New tables are created with <code>{ }</code> braces. For example: <code>{1, 2, 3.00e0}</code>
"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: 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

<pre>function add_two(first_value, second_value) return first_value + second_value end print(add_two(3, 4), type(add_two))</pre>	<pre>7 function</pre>

Example: Table

<pre>atable = {1, 2, 3, 4} print(atable, type(atable)) print(atable[1]) print(atable[4])</pre>	Defines a table with four numeric elements. Note that the <i>table</i> value (shown here as a096cd30) varies. <pre>table: a096cd30 table 1 4</pre>

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the runtime environment.

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 runtime environment. Like all data that is stored in the runtime environment, the function persists until it is removed from the runtime 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

0.666666666666667

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.

NOTE

The following steps use TSB Embedded. You can also use the `loadscript` and `endscript` commands to create the script over the remote interface. See [Load a script by sending commands over the remote interface](#) (on page 14-4).

Steps to create a function using a script:

1. In TSB Embedded, enter a name into the TSP Script box. For example, type `MakeMyFunction`.
2. Enter the function as the body of the script. This example concatenates two strings:

```
MyFunction = function (who)
    print ("Hello".. who)
end
```

3. Select **Save Script**.
4. `MakeMyFunction` is now on the instrument in a global variable with the same name as the script (`MakeMyFunction`). However, the function defined in the script does not yet exist because the script has not been executed.
5. Run the script as a function. For this example, send:

```
MakeMyFunction()
```

This instructs the instrument to run the script, which creates the `MyFunction` global variable. This variable is of the type "function" (see [Variable types](#) (on page 14-16)).

6. Run the new function with a value.

```
MyFunction("world")
```

The response message is:

```
Hello world.
```

Group commands using the function keyword

The following script contains instrument commands that display the name of the person that is using the script on the virtual front panel of the instrument. It takes one parameter to represent this name. When this script is run, the function is loaded in memory. Once loaded into memory, you can call the function outside of the script to execute it.

When calling the function, you must specify a string for the *name* argument of the function. For example, to set the name to **John**, call the function as follows:

```
myDisplay("John")
```

Example: User script

User script created in Test Script Builder or TSB Embedded	User script created in a different program
<pre>function myDisplay(name) display.clear() display.settext(name .. "\$N is here!") end</pre>	<pre>loadscript function myDisplay(name) display.clear() display.settext(name .. " \$N is here!") end endscript</pre>

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

Logical 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

<code>print(10 or errorqueue.next())</code>	1.00000e+01
<code>print(nil or "a")</code>	a
<code>print(nil and 10)</code>	nil
<code>print(false and errorqueue.next())</code>	false
<code>print(false and nil)</code>	false
<code>print(false or nil)</code>	nil
<code>print(10 and 20)</code>	2.00000e+01

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)
- *, /
- +, -
- .. (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

<code>reading + offset < testValue/2+0.5</code>	<code>= (reading + offset) < ((testValue/2)+0.5)</code>
<code>3+reading^2*4</code>	<code>= 3+((reading^2)*4)</code>
<code>Rdg < maxRdg and lastRdg <= expectedRdg</code>	<code>= (Rdg < maxRdg) and (lastRdg <= expectedRdg)</code>
<code>-reading^2</code>	<code>= -(reading^2)</code>
<code>reading^testAdjustment^2</code>	<code>= reading^(testAdjustment^2)</code>

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 14-22) 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 terminates 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 \(lua.org\)](http://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
<code>collectgarbage()</code> <code>collectgarbage(<i>limit</i>)</code>	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 14-29) for more information.
<code>gcinfo()</code>	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 14-29) for more information.
<code>tonumber(<i>x</i>)</code> <code>tonumber(<i>x</i>, <i>base</i>)</code>	Returns <i>x</i> converted to a number. If <i>x</i> is already a number, or a convertible string, the number is returned; otherwise, it returns <code>nil</code> . 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 and an optional exponent. In other bases, only unsigned integers are accepted.
<code>tostring(<i>x</i>)</code>	Receives an argument of any type and converts it to a string in a reasonable format.
<code>type(<i>v</i>)</code>	Returns (as a string) the type of its only argument. The possible results of this function are "nil" (a string, not the value <code>nil</code>), "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
<code>string.byte(s)</code> <code>string.byte(s, i)</code> <code>string.byte(s, i, j)</code>	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 .
<code>string.char(...)</code>	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.
<code>string.format(formatstring, ...)</code>	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 <code>printf</code> family of standard C functions. The only differences are that the modifiers <code>*</code> , <code>l</code> , <code>L</code> , <code>n</code> , <code>p</code> , and <code>h</code> are not supported and there is an extra option, <code>q</code> . The <code>q</code> 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: <code>string.format('%q', 'a string with "quotes" and \n new line')</code> produces the string: <code>"a string with \"quotes\" and \n new line"</code> The options <code>c</code> , <code>d</code> , <code>E</code> , <code>e</code> , <code>f</code> , <code>g</code> , <code>G</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>X</code> , and <code>x</code> all expect a number as argument. <code>q</code> and <code>s</code> expect a string. This function does not accept string values containing embedded zeros, except as arguments to the <code>q</code> option.
<code>string.len(s)</code>	Receives a string and returns its length. The empty string <code>""</code> has length 0. Embedded zeros are counted, so <code>"a\000bc\000"</code> has length 5.
<code>string.lower(s)</code>	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
<code>string.rep(s, n)</code>	Returns a string that is the concatenation of n copies of the string s .
<code>string.sub(s, i)</code> <code>string.sub(s, i, j)</code>	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 <code>string.sub(s, 1, j)</code> returns a prefix of s with length j , and <code>string.sub(s, -i)</code> returns a suffix of s with length i .
<code>string.upper(s)</code>	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
<code>math.abs(x)</code>	Returns the absolute value of x .
<code>math.acos(x)</code>	Returns the arc cosine of x .
<code>math.asin(x)</code>	Returns the arc sine of x .
<code>math.atan(x)</code>	Returns the arc tangent of x .
<code>math.atan2(y, x)</code>	Returns the arc tangent of y/x but uses the signs of both parameters to find the quadrant of the result (it also correctly handles the case of x being zero).
<code>math.ceil(x)</code>	Returns the smallest integer larger than or equal to x .
<code>math.cos(x)</code>	Returns the cosine of x .
<code>math.deg(x)</code>	Returns the angle x (given in radians) in degrees.
<code>math.exp(x)</code>	Returns the value e^x .
<code>math.floor(x)</code>	Returns the largest integer smaller than or equal to x .
<code>math.frexp(x)</code>	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).
<code>math.ldexp(m, e)</code>	Returns $m2^e$ (e should be an integer).
<code>math.log(x)</code>	Returns the natural logarithm of x .
<code>math.log10(x)</code>	Returns the base-10 logarithm of x .
<code>math.max(x, ...)</code>	Returns the maximum value among its arguments.
<code>math.min(x, ...)</code>	Returns the minimum value among its arguments.
<code>math.pi</code>	The value of π (3.141592654).
<code>math.pow(x, y)</code>	Returns x^y (you can also use the expression x^y to compute this value).
<code>math.rad(x)</code>	Returns the angle x (given in degrees) in radians.
<code>math.random()</code> <code>math.random(m)</code> <code>math.random(m, n)</code>	This function is an interface to the simple pseudorandom generator function <code>rand</code> 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 , <code>math.random()</code> returns a uniform pseudorandom integer in the range $[1, m]$. When called with two integer numbers m and n , <code>math.random()</code> returns a uniform pseudorandom integer in the range $[m, n]$.
<code>math.randomseed(x)</code>	Sets x as the seed for the pseudorandom generator; equal seeds produce equal sequences of numbers.
<code>math.sin(x)</code>	Returns the sine of x .
<code>math.sqrt(x)</code>	Returns the square root of x . You can also use the expression $x^{0.5}$ to compute this value.
<code>math.tan(x)</code>	Returns the tangent of x .

Programming example: User script

The following script puts a message on the virtual front panel display slowly, one character at a time. The intent of this example is to demonstrate:

- The use of a `for` loop
- Simple display remote commands
- Simple Lua string manipulation

NOTE

When creating a script using the TSB Embedded, you do not need the shell commands `loadscript` and `endscript`, as shown in the examples below.

Example: User script

User script created in TSB Embedded	User script created in user's own program
	<code>loadscript</code>
<pre>display.clear() myMessage = "Hello World!" for k = 1, string.len(myMessage) do x = string.sub(myMessage, k, k) display.settext(x) print(x) delay(1) end</pre>	<pre>display.clear() myMessage = "Hello World!" for k = 1, string.len(myMessage) do x = string.sub(myMessage, k, k) display.settext(x) print(x) delay(1) end</pre>
	<code>endscript</code>

Test Script Builder

Keithley Instruments Test Script Builder (TSB) is a software tool you can use to develop scripts for TSP-enabled instruments.

Installing the TSB software

The installation files for the TSB software are available at tek.com/keithley.

To install the 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.

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

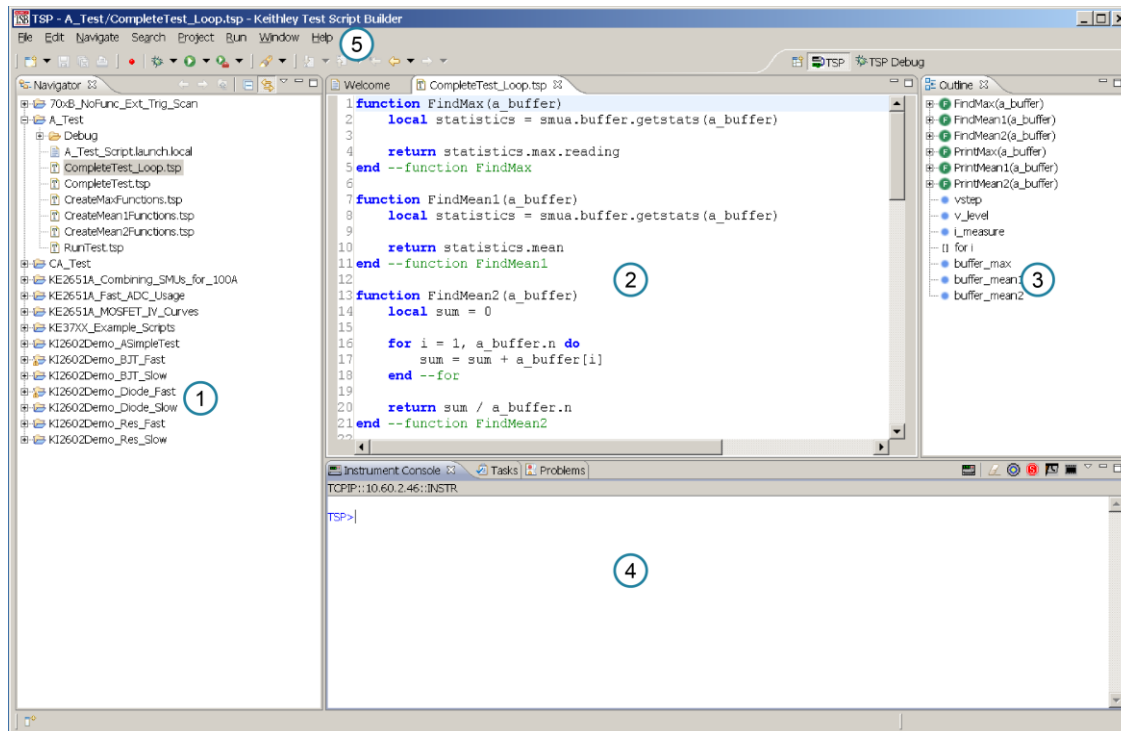
The Keithley Instruments Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands. For more information about using the Lua scripting language with Keithley TSP-enabled instruments, refer to the [Fundamentals of programming for TSP](#) (on page 14-14).

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 120: 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

The Outline tab is visible by default in the TSP perspective.

Icon	Name	Examples
	Global function variable	<code>function gFunction() end</code>
	Local function variable	<code>local function lFunction() end</code>
	Anonymous function	<code>myTest(function() return 1 end)</code>
	Global table variable	<code>gTable = { }</code>
	Local table variable	<code>local lTable = { }</code>
	Other table field	<code>testTable.unit1 = "This is unit 1" testTable.unit2 = "This is unit 2"</code>
	Global variable	<code>gVariable = 3</code>
	Local variable	<code>local lVariable = 5</code>
	Table method	<code>gTable = { } function gTable:testmethod() end</code>
	Nonfunction block statement (example 1)	<code>if true == true then local var end</code>
	Nonfunction block statement (example 2)	<code>for index = 1, 10 do end</code>

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. For additional information, refer to the online help that is accessible from Test Script Builder (TSB).

Password management

The 2606B has password capabilities that let you decide how to password protect the instrument. Password protection prevents unauthorized access to any remote interface and reserves the instrument exclusively for your use.

When password usage is enabled, you must supply a password to change the configuration or to control an instrument from a remote command interface.

Setting the password from a command or web interface

The attribute `localnode.passwordmode` enables passwords and sets the mode. The password mode identifies which interface to password protect.

Set this attribute to one of the values below to enable password checking:

- `localnode.PASSWORD_NONE` or 0: Disable passwords everywhere
- `localnode.PASSWORD_WEB` or 1: Use passwords on the web interface only
- `localnode.PASSWORD_LAN` or 2: Use passwords on the web interface and all LAN interfaces
- `localnode.PASSWORD_ALL` or 3: Use passwords on the web interface and all remote command interfaces

NOTE

When a password is set for the web interface, you cannot make changes using the web interface options Virtual Front Panel, Flash Upgrade, or TSB Embedded.

The password lock feature on 2606B is similar to the lock feature on your computer.

NOTE

You must assign a password to use this feature. Passwords can be up to 255 characters.

To set the password using the web interface:

1. From the web interface, select **Set Password**.
The LXI - Keithley Instruments - 2606B - Administration page is displayed.
2. In **Current Password**, type the existing password. The default is `admin`.
3. In **New Password**, type the new password.
4. Retype the new password in **Confirm New Password**.
5. Select **Submit**.

The LXI Welcome page is displayed.

To enable the password from a command interface:

To lock the instrument when you are away from the testing area, send the following command:

```
password
```

The remote interface is locked. The module of the 2606B does not respond to commands issued from the command interface until you unlock the interface. This reserves the instrument and protects the test script running on the instrument.

Unlocking the remote interface

If the remote interface is locked, you must enter the password before the module of the 2606B responds to any command issued over a remote interface.

NOTE

The password for the example below is `Keithley`.

To unlock the remote interface, send the following command:

```
password Keithley
```

The module of the 2606B is unlocked and communicates with any remote interface.

Resetting the password

You can reset the password from the virtual front panel. Once you enable the password feature, the 2606B stores this password until the LAN configuration is reset or until you reset the password.

To reset the password:

1. From the virtual front panel, press the **MENU** key.
2. Select **RESET-PASSWORD**.

NOTE

Resetting the LAN settings also resets the password feature. If you reset the LAN settings, you must re-enable the password feature.

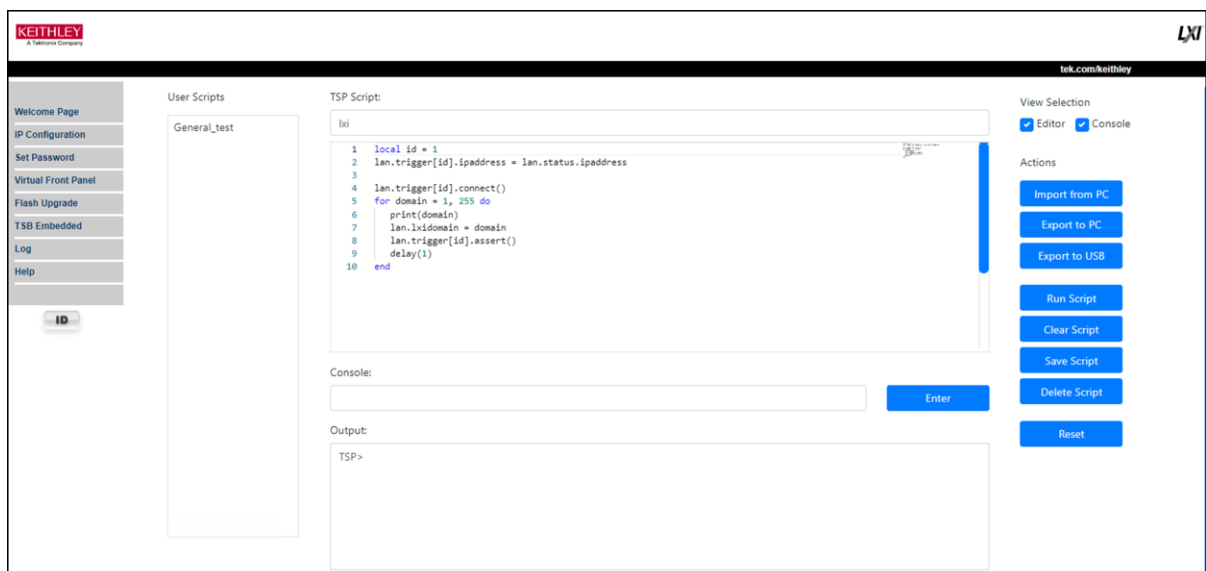
Working with TSB Embedded

TSB Embedded is a script management tool that is available through the web interface of the instrument. You can use TSB Embedded to create, modify, and save test scripts, and to send individual commands. TSB Embedded provides some of the features of Test Script Builder (TSB). TSB is a software tool that simplifies building test scripts for Keithley Instruments that are enabled to use the Test Script Processor (TSP®) scripting engine. You can also use TSB Embedded to send individual commands to the instrument.

NOTE

For more information on scripts, refer to [Fundamentals of scripting for TSP](#) (on page 14-1).

Figure 121: TSB Embedded interface



Simplify the display

You can simplify the TSB Embedded display to show only the TSP Script Editor or only the instrument Console.

The TSP Script Editor includes the list of User Scripts, the script entry area, and the script management buttons.

The Console includes only the Console and Output boxes and controls. You can use these to send individual commands to the instrument.

To display only the TSP Script Editor features, under View Selection, select **Editor**. To display only the Console features, select **Console**.

Create a script using TSB Embedded

NOTE

If you are using TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

You can create a script from the instrument web interface with TSB Embedded. When you save the script, it is loaded into the runtime environment and saved in the nonvolatile memory of the instrument.

To create a script using TSB Embedded:

1. If there is an existing script, select **Clear Script**.
2. In the **TSP Script** box, enter a name for the script.
3. In the input area, enter the sequence of commands to be included in the script.
4. Select **Save Script**. The name is added to the User Scripts list.

NOTE

If there is an error in the code, a message is displayed in the Output area and the script is not saved. Resolve the error and select Save Script again.

Copy an existing script

You can copy an existing script to a script with a different name.

To copy an existing script:

1. Select the script from the **User Scripts** list.
2. In the **TSP Script** box, delete the existing name and enter a name for the script.
3. Select **Save Script**. The script with the new name is added to the User Scripts list.

Run a script

Running a script executes the script on the instrument.

To run a script:

1. Select a script from the User Scripts list.
2. Select **Run Script**.

To stop a running script, select **Abort**.

The Abort button is only displayed while a script is running.

Delete a script

NOTE

You cannot retrieve a deleted script. Be sure to back up your script to your computer before deleting.

To delete a script from TSB Embedded:

1. Select the script from the User Scripts list.
2. Select **Delete Script**.
3. Select **Delete** on the confirmation message.

Modify a script

You can modify the script in TSB Embedded.

To modify a script:

1. Select a script from the User Scripts list.
2. Modify the code in the editor.
3. Select **Save Script**.

Import a script from a computer

You can import a script from any drive that you can access from the host computer, including USB flash drives.

When you import a script with a `loadscript` command, the TSP Script is assigned the `loadscript` name (not the file name). If the script does not include a `loadscript` command, the TSP Script name is not assigned.

TSP files have the extension `.tsp`.

To import a script from the host computer:

1. Select **Import from PC**.
2. Choose **Select File** to select a file. You can also drag the file into the **Select File** box.
3. Select **Import**.
4. In the TSP Script box, modify the script name as needed.
5. Select **Save Script**.

The script is added to the User Scripts list.

Export a script to a computer

You can download a script from TSB Embedded to the host computer.

TSP scripts have the extension `.tsp`.

To export a script to a computer:

1. Select the script from the User Scripts list.
2. Select **Export to PC**. The file is saved as a download.
3. Use the procedure for your browser to work with the file.

Export a script to the instrument USB

You can save a script to a USB flash drive inserted into the USB connector on the front panel of the instrument.

To export a script to a USB flash drive:

1. Insert a flash drive into the USB port on the front panel of the instrument.
2. Select the script from the User Scripts list.
3. Select **Export to USB**.
4. In the **Export File Name** box, enter the file name.
5. Select **Export**.

Reset the instrument using TSB Embedded

The reset option in TSB Embedded performs an abort operation followed by a `reset()` command.

Refer to [reset\(\)](#) (on page 15-176) for additional information.

Send individual instrument commands with TSB Embedded

You can send individual commands to the instrument using TSB Embedded. The response from the instrument appears in the Output box.

To send commands from the console:

1. Type the command in **Console**.
2. Press the **Enter** key to send the command to the instrument. The command is displayed in the Output box. If there is a response to the command, it is displayed after the command.

To clear information from the Output box:

1. Right-click in the **Output** box.
2. Select **Clear**.

To copy information from the Output box:

1. Right-click in the **Output** box.
2. Select **Copy**. The information is copied to the clipboard.

Advanced scripting for TSP

The following topics describe advanced information that can help you understand how the Test Script Processor (TSP®) scripting engine works.

Global variables and the `script.user.scripts` table

When working with script commands, it is helpful to understand how scripts are handled in the instrument.

Scripts are loaded into the runtime environment from nonvolatile memory when you turn the instrument on. They are also added to the runtime environment when you load them into the instrument.

A script in the runtime environment can be:

- A named script
- An unnamed script
- The anonymous script (which is a special unnamed script)

Script names can be assigned by using the `loadscript` command or by defining the `scriptVar` parameter of the `script.new()` function. When a named script is loaded into the runtime environment:

- A global variable with the same name is created so that you can reference the script more conveniently.
- An entry for the script is added to the `script.user.scripts` table.

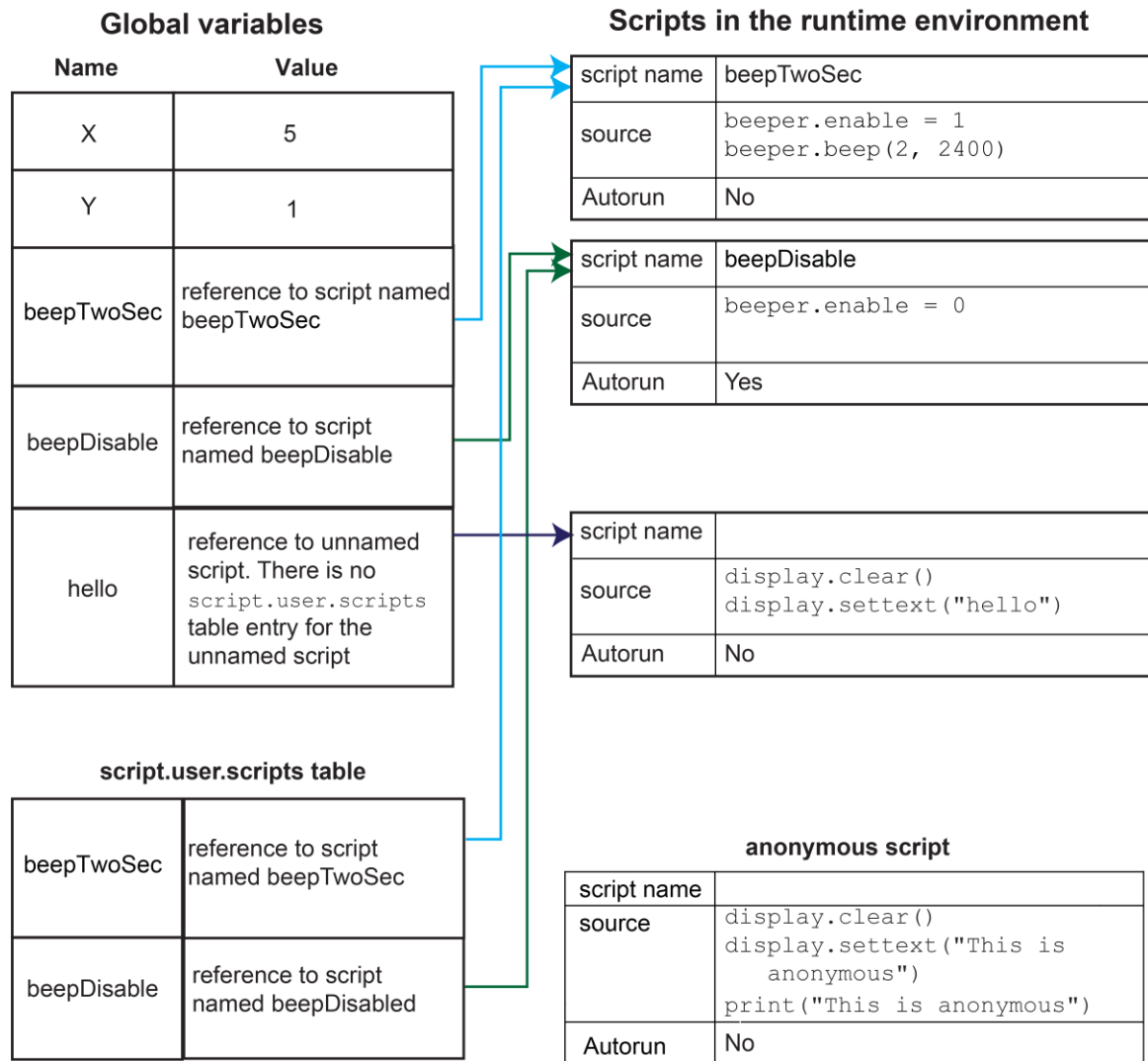
When you create a script using the `script.new()` function without providing a name, the script is added to the runtime environment as an unnamed script. The `script.new()` function returns the script, but the script is not added to the `script.user.scripts` table.

When the anonymous script is loaded, it does not have a global variable or an entry in the `script.user.scripts` table. If there is an existing anonymous script, it is replaced by the new one.

When the instrument is turned off, everything in the runtime environment is deleted, including the scripts and global variables.

See the figure below to see how the scripts, global variables, and `script.user.scripts` table interrelate.

Figure 122: Global variables and scripts in the runtime environment



Create a script using the `script.new()` command

Use the `script.new()` function to copy an existing script from the local node to a remote node. This enables parallel script execution.

You can create a script with the `script.new()` function using the command:

```
scriptVar = script.new(code, name)
```

Where:

<code>scriptVar</code>	=	Name of the variable created when the script is loaded into the runtime environment
<code>code</code>	=	Content of the script
<code>name</code>	=	Name that is added to the <code>script.user.scripts</code> table

For example, to set up a two-second beep, you can send the command:

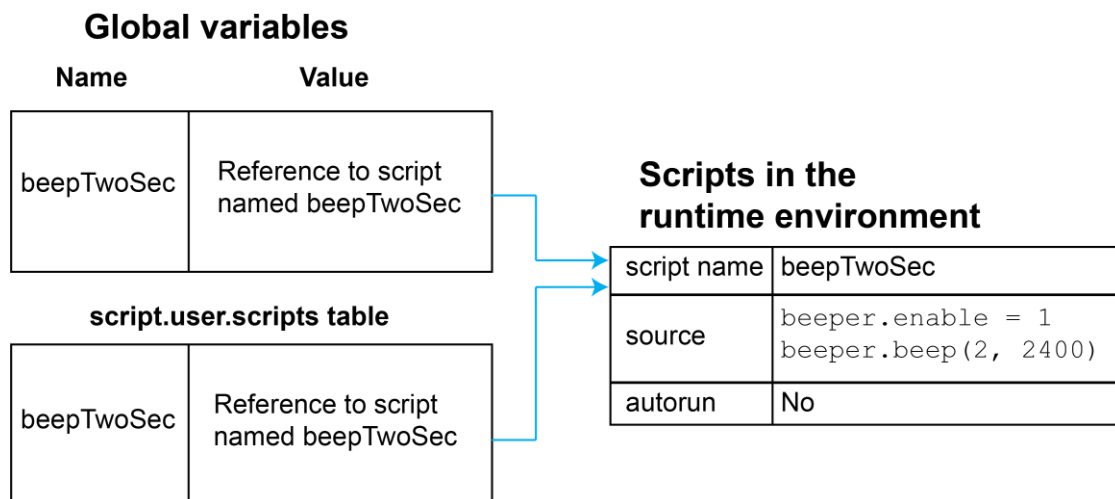
```
beepTwoSec = script.new("beeper.enable = 1 beeper.beep(2, 2400)", "beepTwoSec")
```

To run the new script, send the command:

```
beepTwoSec()
```

When you add `beepTwoSec`, the global variable and `script.user.scripts` table entries are made to the runtime environment, as shown in the following figure.

Figure 123: Runtime environment after creating a script



Create an unnamed script using `script.new()`

NOTE

Unnamed scripts are not available from the virtual front-panel display of the instrument. Only the anonymous script and named scripts are available from the virtual front-panel display.

When you create a script using `script.new()`, if you do not include *name*, the script is added to the runtime environment as an unnamed script. The `script.new()` function returns the script. You can assign it to a global variable, a local variable, or ignore the return value. A global variable is not automatically created.

For example, send the following command:

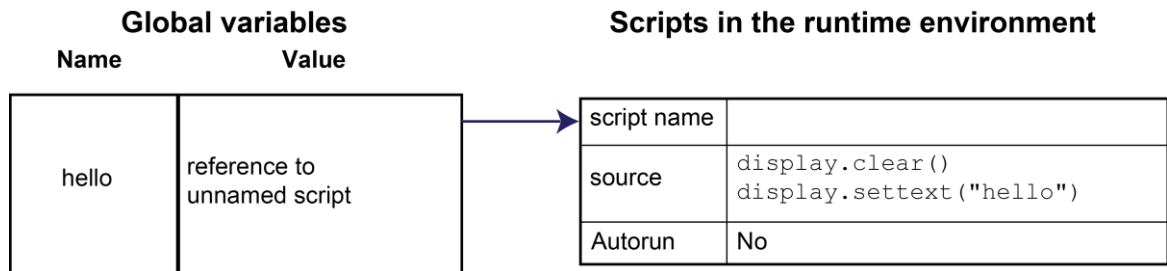
```
hello = script.new('display.clear() display.settext("hello")')
```

A script is created in the runtime environment and a global variable is created that references the script.

To run the script, send the command:

```
hello()
```

Figure 124: Create an unnamed script



A script becomes unnamed if you create a new script with the same name. In this circumstance, the name of the script in the `script.user.scripts` table is set to an empty string before it is replaced by the new script.

For example, if `beepTwoSec` already exists in the `script.user.scripts` table and you sent:

```
beepTwoSec1200 = script.new("beeper.enable = 1 beeper.beep(2, 1200)", "beepTwoSec")
```

The following actions occur:

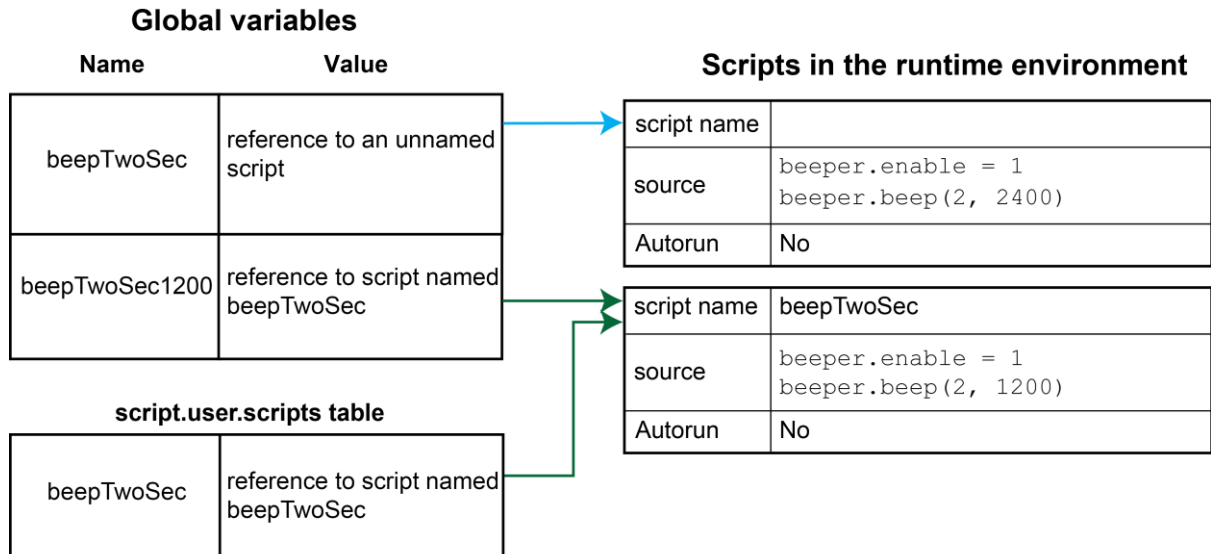
- `beepTwoSec1200` is added as a global variable.
- The script that was in the runtime environment as `beepTwoSec` is changed to an unnamed script (the name attribute is set to an empty string).
- The global variable `beepTwoSec` remains in the runtime environment unchanged (it points to the now unnamed script).
- A new script named `beepTwoSec` is added to the runtime environment.

In this example, you can access the new script by sending either of the following commands:

```
beepTwoSec1200 ()
script.user.scripts.beepTwoSec ()
```

To access the unnamed script, you can send the command:

```
beepTwoSec ()
```

Figure 125: Create a new script with the name of an existing script

Note that the `script.user.scripts` table entry referencing `beepTwoSec` was removed and a new entry for `beepTwoSec` has been added

Rename a script

You can rename a script. You might want to rename a script if you need to name another script the same name as the existing script. You could also rename an existing script to be the autoexec script.

To change the name of a script, use the command:

```
scriptVar.name = "renamedScript"
```

Where:

<code>scriptVar</code>	=	The global variable name
<code>"renamedScript"</code>	=	The new name of the user script that was referenced by the <code>scriptVar</code> global variable

After changing the name, you need to save the original script to save the change to the name attribute.

For example:

```
beepTwoSec.name = "beep2sec"
beepTwoSec.save()
```

Run the `beep2sec` script using the following command:

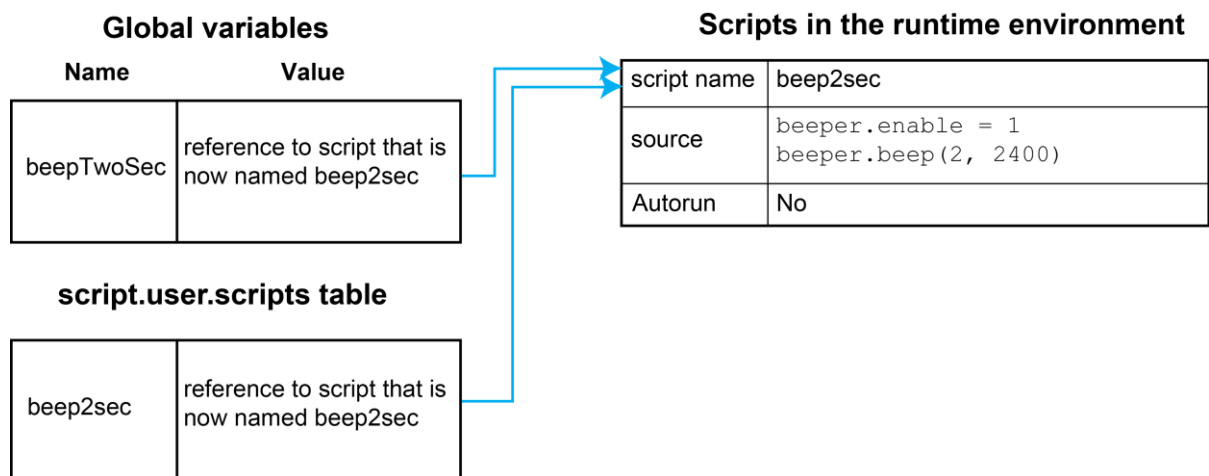
```
script.user.scripts.beep2sec()
```

NOTE

If the new name is the same as a name that is already used for a script, the name of the existing script is removed and that script becomes unnamed. This removes the existing script if there are no other variables that reference the previous script. If variables do reference the existing script, the references remain intact.

Changing the name of a script does not change the name of any variables that reference that script. After changing the name, the script is in the `script.user.scripts` table under its new name.

Figure 126: Rename script



For example, to change the name of the script named `test2` to be `autoexec`:

```
test2.name = "autoexec"
test2.save()
```

The `autoexec` script runs automatically when the instrument is turned on. It runs after all the scripts have loaded and any scripts marked as `autorun` have run.

NOTE

You can also use the `script.new()` and the `scriptVar.source` attribute commands to create a script with a new name. For example, if you had an existing script named `test1`, you could create a new script named `test2` by sending the command:

```
test2 = script.new(test1.source, "test2")
```

See [script.new\(\)](#) (on page 15-180).

Retrieve a user script

There are several ways to retrieve the source code of a user script:

- One line at a time: Use `scriptVar.list()` to retrieve the source code one line at a time
- Entire script: Use the `print(scriptVar.source)` command to retrieve the script source code as a single string
- Use TSB Embedded; refer to [Working with TSB Embedded](#) (on page 14-38) for more information

See [Create and load a script](#) (on page 14-3) for information about recreating the script and loading it back into the instrument.

NOTE

To get a list of scripts that are in nonvolatile memory, use the [script.user.catalog\(\)](#) (on page 15-183) function.

Retrieve source code one line at a time

To retrieve the source code one line at a time, send the `scriptVar.list()` command. When this command is received, the instrument sends the entire script. Each line of the script is sent as a separate response message. The output includes the `loadscript` or `loadandrunscript` and `endscript` keywords.

After retrieving the source code, you can modify and save the command lines as a user script under the same name or a new name.

To retrieve the source code of a script one line at a time, send the command:

```
scriptVar.list()
```

Where `scriptVar` is the name of the script.

NOTE

To retrieve the commands in the anonymous script, use `script.anonymous.list()`.

Example: Retrieve source code one line at a time

```
test.list()
```

Retrieve the source of a script named "test".
The output looks similar to:
`loadscript test`
`display.clear()`
`display.settext("This is a test")`
`print("This is a test")`
`endscript`

Retrieve a script as a single string

To retrieve the entire user script source code as a single string, use the `scriptVar.source` attribute. The `loadscript` or `loadandrunscript` and `endscript` keywords are not included.

To retrieve the source code as a single string, send the command:

```
print(scriptVar.source)
```

Where `scriptVar` is the name of the script.

Example: Retrieve the source code as a single string

```
print(test.source)
```

Retrieve the source of a script named "test".

Output looks similar to:

```
display.clear() display.settext("This is a test") print("This  
is a test")
```

Retrieve a script using TSB Embedded

In TSB Embedded, from the User Scripts list, select the script you want to retrieve. The contents of the script are displayed. See [Working with TSB Embedded](#) (on page 14-38) for more information.

Delete user scripts from the instrument

In most circumstances, you can delete a script using `script.delete()` (as described in [Delete user scripts](#) (on page 14-12)), and then turn the instrument off and back on again. However, if you cannot turn the instrument off, you can use the following steps to completely remove a script from the instrument.

When you completely remove a script, you delete all references to the script from the runtime environment, the `script.user.scripts` table, and nonvolatile memory.

To completely remove a script:

1. **Remove the script from the runtime environment.** Set any variables that refer to the script to `nil` or assign the variables a different value. For example, to remove the script "beepTwoSec" from the runtime environment, send the following code:

```
beepTwoSec = nil
```
2. **Remove the script from the `script.user.scripts` table.** Set the name attribute to an empty string (""). This makes the script nameless, but does not make the script become the anonymous script. For example, to remove the script named "beepTwoSec", send the following code:

```
script.user.scripts.beepTwoSec.name = ""
```

3. **Remove the script from nonvolatile memory.** To delete the script from nonvolatile memory, send the command:

```
script.delete("name")
```

Where *name* is the name that the script was saved as. For example, to delete "beepTwoSec", send:

```
script.delete("beepTwoSec")
```

Restore a script to the runtime environment

You can retrieve a script that was removed from the runtime environment but is still saved in nonvolatile memory.

To restore a script from nonvolatile memory into the runtime environment, you can use `script.restore("scriptName")`, where *scriptName* is the user-defined name of the script to be restored.

For example, to restore a user script named "test9" from nonvolatile memory:

```
script.restore("test9")
```

Memory considerations for the runtime environment

The 2606B reserves 32 MB of memory for dynamic runtime use. Approximate allocation of this memory is shown below:

5 MB	Firmware general operation
1 MB	Reserve for instrument internal operation
2 MB	Reserve for future firmware updates
24 MB	Runtime environment, user-created reading buffers, and active sweep configuration

Note that the runtime environment, user-created reading buffers, and active sweep configuration must fit in the 24 MB of memory that is available. The amount of memory used by a reading buffer is approximately 15 bytes for each entry requested.

Reading buffers also use a small amount of memory for reading buffer management, which is not significant when making memory utilization calculations. For example, assume two reading buffers were created. One of them was created to store up to 1,000 readings and the other to store up to 2,500 readings. The memory reserved for the reading buffers is calculated as follows:

$$(1000 * 15) + (2500 * 15) = 52,500 \text{ bytes or } 52.5 \text{ kilobytes}$$

Note that the dedicated reading buffers do not consume memory that is needed by the runtime environment; do not include them in your memory consumption calculations. Also, reading buffers for remote nodes consume memory on the remote node, not the local node. Make sure the total reading buffer memory for any particular remote node does not exceed 24 MB, but do not include that amount in your local memory consumption calculations.

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 the source-measure unit (SMU) settings, 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 runtime environment, sweep configuration and reading buffers to exceed 24 MB. When this occurs, there is a risk that memory allocation errors will occur and commands will not be executed as expected.

CAUTION

If the instrument encounters memory allocation errors when the memory used is above 95 percent, 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 data in reading buffers will be lost.

The amount of memory in use can be checked using the `meminfo()` function. The first value returned by `meminfo()` is the number of kilobytes of memory in use.

If the amount of memory used is over 95 percent or if you receive out-of-memory errors, you should reduce the amount of memory that is used.

Some suggestions for increasing the available memory:

- Turn the instrument off and on. This deletes scripts that have not been saved and reloads only scripts that have been stored in nonvolatile memory.
- Remove unneeded scripts from nonvolatile memory. Scripts are loaded from nonvolatile memory into the runtime environment when the instrument is turned on. See [Delete user scripts from the instrument](#) (on page 14-49).
- Reduce the number of TSP-Link® nodes.
- Delete unneeded global variables from the runtime environment by setting them to `nil`.
- Set the source attribute of all scripts to `nil`.
- Adjust the `collectgarbage()` settings in Lua. See [Lua memory management](#) (on page 14-29) for more information.
- Review scripts to optimize 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.

TSP-Link system expansion interface

The TSP-Link® expansion interface allows the 2606B instrument to communicate with other Test Script Processor (TSP®) enabled instruments. The test system can be expanded to include up to 32 TSP-Link enabled instruments.

CAUTION

Combining two modules to achieve greater currents in both source voltage and source current applications requires specific precautions, including configuration settings. Make sure that you adequately understand the risks involved and the measures needed to accommodate the combination of two modules. To prevent damage to the 2606B, connected instruments, and the device under test, make sure proper procedures are used. For further information, visit the Keithley website at tek.com/keithley for application notes on combining channels from two modules.

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.

When any node transitions from local operation to remote operation, it becomes the master of the system. All other nodes also transition to remote operation and become its subordinates. When any node transitions from remote operation to local, all other nodes also transition to local operation, and the master/subordinate relationship between nodes is dissolved.

The expanded system can be stand-alone or computer-based.

Stand-alone system: You can run a script from the virtual front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation (REM indicators turn on). The node running the script becomes the master and can control all other nodes, which become its subordinates. When the script is finished running, all the nodes in the system return to local operation (REM indicators turn off), and the master/subordinate relationship between nodes is dissolved.

Computer-based system: You can use a computer and a remote communications interface to any 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 (REM indicators turn on). 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.

TSP-Link nodes

Each instrument (node) attached to the TSP-Link® network must be identified by assigning it a unique TSP-Link node number.

Commands for remote nodes are stored in the `node` table. An individual node is accessed as `node[N]`, where *N* is the node number assigned to the node.

All TSP-accessible remote commands can be accessed as elements of the specific node. The following attributes are examples of items you can access:

- `node[N].model`: The product model number string of the node.
- `node[N].revision`: The product revision string of the node.
- `node[N].serialno`: The product serial number string of the node.

You do not need to know the node number of the node that is running a script. The variable `localnode` is an alias for the node entry of the node where the script is running. For example, if a script is running on node 5, you can use the global variable `localnode` as an alias for `node[5]`. To access the product model number for this example, use `localnode.model`.

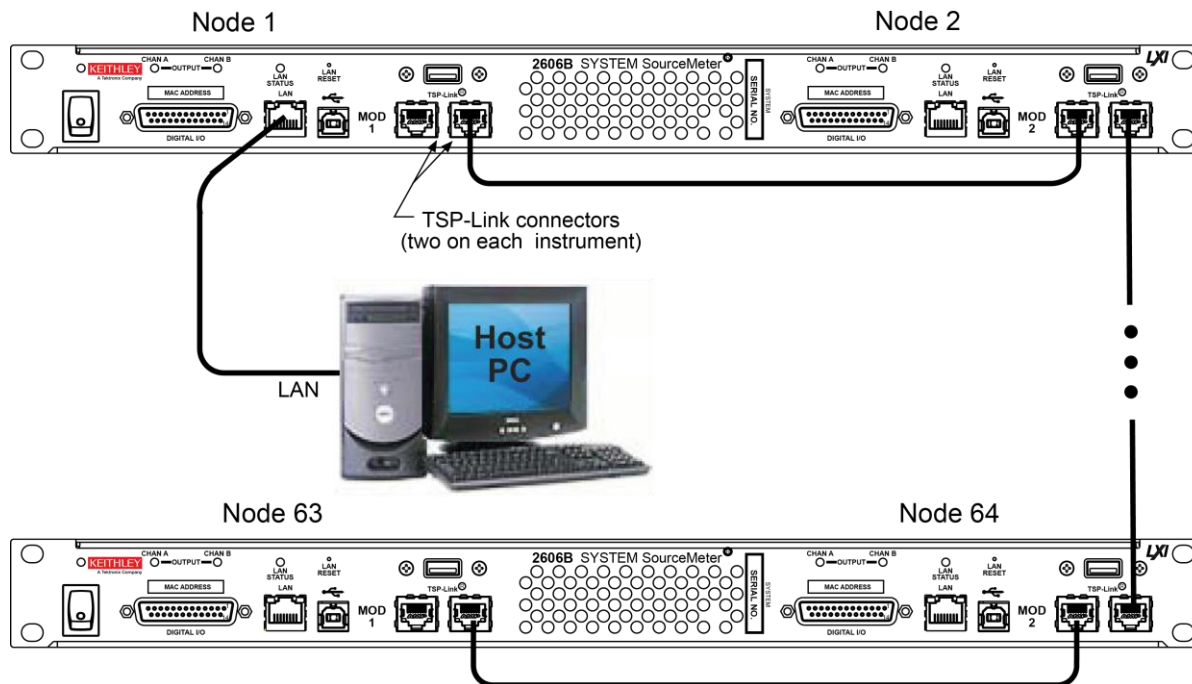
Connections

Connections for an expanded system are shown in the following figure. As shown, one instrument is optionally connected to the computer using the LAN or USB interface. Details about these computer communication connections are described in Remote communications interfaces.

All the instruments in the system are connected in a sequence (daisy-chained) using LAN crossover cables. Cables must be:

- LAN crossover cables, category 5e or higher
- Less than 3 m maximum between nodes

Figure 127: 2606B TSP-Link connections



Initialization

Before you can use a TSP-Link® system, it must be initialized. For initialization to succeed, each instrument in a TSP-Link system must be assigned a different node number.

Assigning node numbers

At the factory, each module of the 2606B instrument is assigned as node 1. The node number is stored in nonvolatile memory and remains in storage when the instrument is turned off. You can assign a node number to a module of the 2606B using the virtual front panel or by using a remote command. There can only be 32 physical nodes, but you can assign node numbers from 1 to 64.

To assign a node number from the virtual front panel of the instrument:

1. Press the **MENU** key, then select **TSPLINK > NODE**.
2. Press the navigation wheel and select the node number.
3. Press the **ENTER** key to save the number.

To assign a node number using a remote command:

Set the `tsplink.node` attribute of the instrument:

```
tsplink.node = N
```

Where $N = 1$ to 64

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.

Resetting the TSP-Link network

After all the node numbers are set, you must initialize the system by performing a TSP-Link® network reset.

NOTE

If you change the system configuration after initialization, you must reinitialize the system by performing a TSP-Link network reset. Changes that require that you reinitialize the TSP-Link network include turning off power or rebooting any instrument in the system, or rearranging or disconnecting the TSP-Link cable connections between instruments.

Front-panel operation

To reset the TSP-Link® network from the virtual front panel:

1. Power on all instruments connected to the TSP-Link network.
2. Press the **MENU** key, select **TSPLINK**, and then press the **ENTER** key.
3. Turn the navigation wheel to select **RESET**, and then press the **ENTER** key.

Remote programming

The commands associated with the TSP-Link® system reset are listed in the following table.

TSP-Link reset commands

Command	Description
<code>tsplink.reset()</code>	Initializes the TSP-Link network
<code>tsplink.state</code>	Reads the state of the TSP-Link network: ▪ “online” if the most recent TSP-Link reset was successful ▪ “offline” if the reset operation failed

An attempted TSP-Link reset operation fails if any of the following conditions are true:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument performing the reset (only if the expected number of nodes was not provided in the reset call)
- One or more of the instruments in the system is turned off
- If the actual number of nodes is less than the expected number

The programming example below illustrates a TSP-Link reset operation and displays its state:

```
tsplink.reset()  
print(tsplink.state)
```

If the reset operation is successful, `online` is output to indicate that communications with all nodes have been established.

Accessing nodes

A TSP-Link® reset command populates the node table. Each instrument in the system corresponds to an entry in this table. Each entry is indexed by the node number of the instrument. The variable `node[N]` (where N is the node number) is used to access any node in the system. For example, node 1 is represented as entry `node[1]` in the node table.

You can access all the remote commands for a specific node by adding `node[N]` to the beginning of the remote command, where N is the node number. For example, to set the NPLC value for the source-measure unit (SMU) A on node 1 to 0.1, you could send this command:

```
node[1].smua.measure.nplc = 0.1
```

The variable `localnode` is an alias for `node[N]`, where N is the node number of the node on which the code is running. For example, if node 1 is running the code, you can use `localnode` instead of `node[1]`.

The following programming examples illustrate how to access instruments in the TSP-Link system (shown in TSP-Link connections):

- Any one of the following commands reset SMU A of node 1 (which, in this example, is the master). The other nodes in the system are not affected.
 - `smua.reset()`
 - `localnode.smua.reset()`
 - `node[1].smua.reset()`
- The following command resets SMU A of node 4, which is a subordinate. The other nodes are not affected.
 - `node[4].smua.reset()`

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.reset()` command. The `tsplink.reset()` 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()
```

Using the abort command

An `abort` command terminates an executing script and returns all nodes to local operation (REM indicators turn off). This dissolves the master/subordinate relationships between nodes. To invoke an abort operation, either send an `abort` command to a specific node or press the EXIT (LOCAL) key on any node in the system.

You can also perform an abort operation by pressing the OUTPUT ON/OFF control on any node. The results are the same as above, with the addition that all source-measure unit (SMU) outputs in the system are turned off.

Triggering with TSP-Link

The TSP-Link® expansion interface has three trigger lines that function similarly to the digital I/O synchronization lines. See [Digital I/O](#) (on page 7-37) and [Triggering](#) (on page 7-1) for more information.

TSP advanced features

Use the Test Script Processor (TSP®) scripting engine's advanced features to:

- Run test scripts simultaneously
- Manage resources allocated to test scripts that are running simultaneously
- Use the data queue to facilitate real-time communication between nodes on the TSP-Link® network

When test scripts are run simultaneously, it improves functional testing, provides higher throughput, and expands system flexibility.

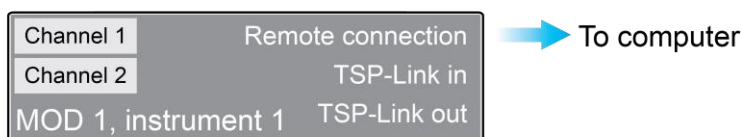
There are two methods you can use to run test scripts simultaneously:

- Create multiple TSP-Link networks
- Use a single TSP-Link network with groups

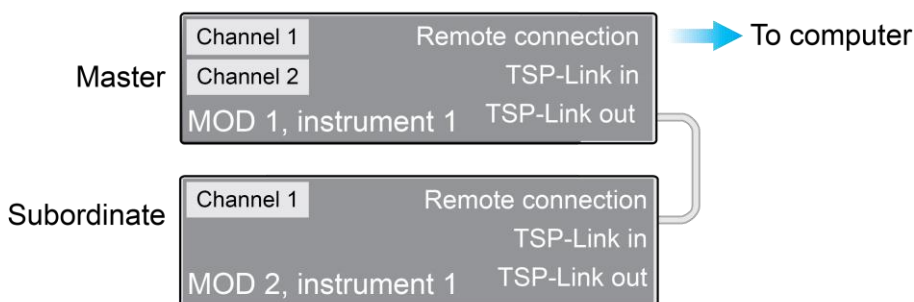
The following figure displays the first method, which consists of multiple TSP-Link networks. Each TSP-Link network has a master node and a remote connection to the computer.

Figure 128: Multiple TSP-Link networks

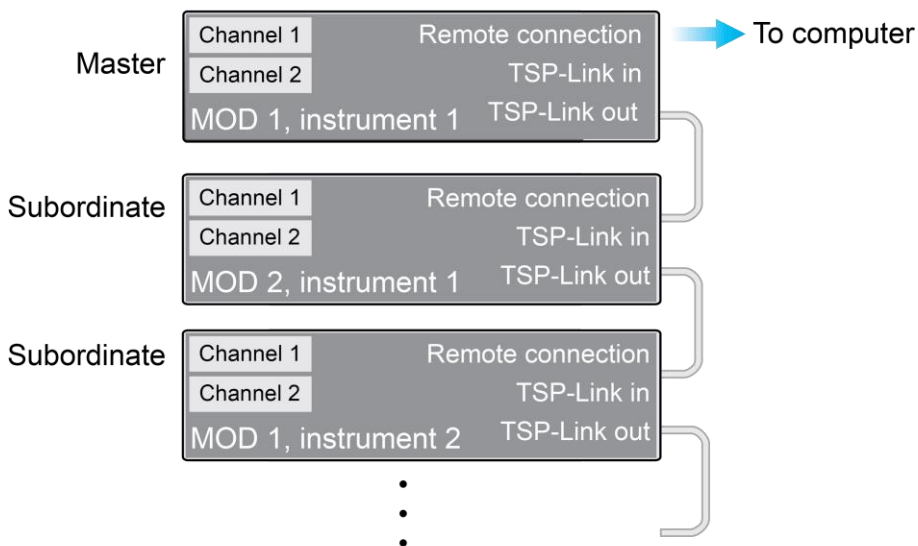
System with two channels



System with three channels



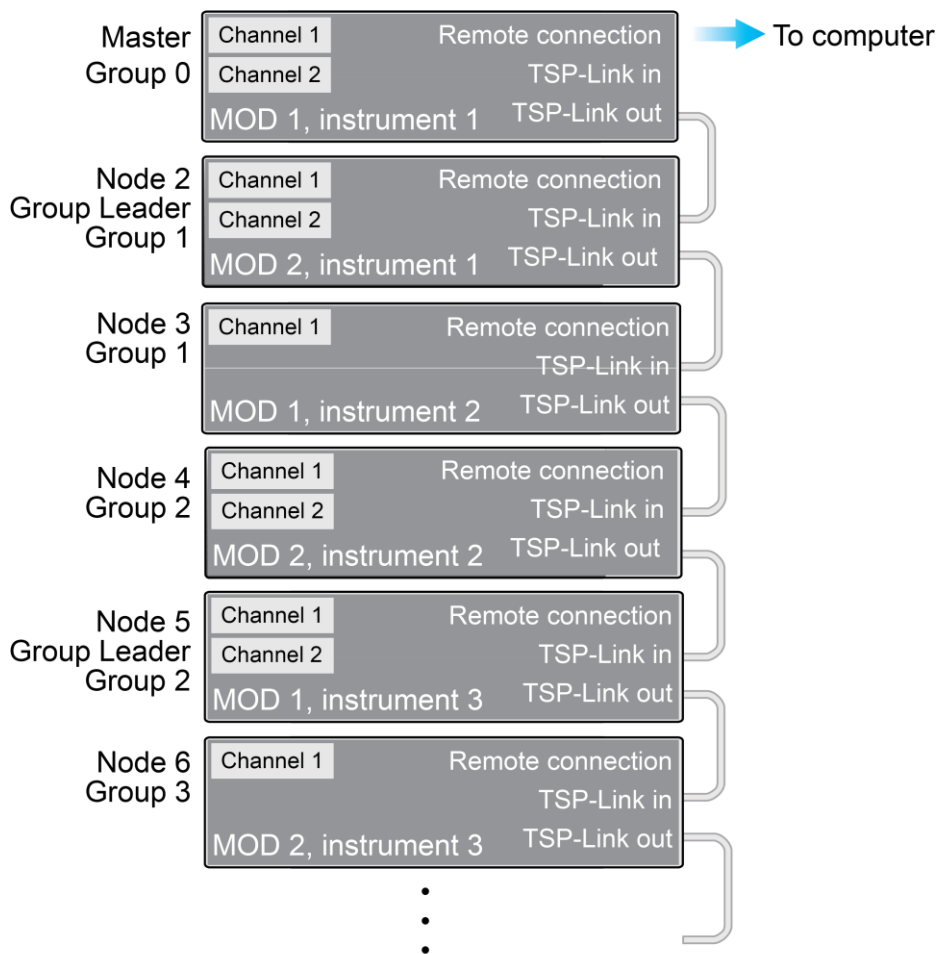
System with 16 or more channels



Another method you can use to run simultaneous test scripts is to use groups with a single TSP-Link network. Each group on the TSP-Link network can run a test while other groups are running different tests.

A group consists of one or more nodes with the same group number. The following figure displays a single TSP-Link network with groups. This method requires one TSP-Link network and a single remote communications connection to the computer.

Figure 129: Single TSP-Link network with groups



The following table shows an example of the functions of 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 5 Initiates and runs a test script on node 6
1	Group leader Node 2	Runs the test script initiated by the master node Initiates remote operations on node 3
	Node 3	Performs remote operations initiated by node 2
2	Group leader Node 5	Runs the test script initiated by the master node Initiates remote operations on node 4
	Node 4	Performs remote operations initiated by node 5
3	Group leader Node 6	Runs the test script initiated by the master node

Using groups to manage nodes on TSP-Link network

The primary purpose of groups is to allow each group to run a different test script simultaneously.

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 is always grouped with the master node (regardless of the group to which the master node is assigned).

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 simultaneous test scripts

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. See [Copying test scripts across the TSP-Link network](#) (on page 14-63) for more information.

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 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 15-407).

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 for more information.

Copying test scripts across the TSP-Link network

To run a large script on a remote node, copy the test script to the remote node to increase the speed of test script initiation.

The code in the example below copies a test script across the TSP-Link® network, creating a copy of the script on the remote node with the same name.

```
-- Add the source code from the script
-- testScript to the data queue.
node[2].dataqueue.add(testScript.source)
-- Create a new script on the remote node
-- using the source code from testScript.
node[2].execute(testScript.name ..
    "= script.new(dataqueue.next(), [{" .. testScript.name .. "}]")
```

Removing stale values from the reading buffer cache

The node that acquires the data also stores the data for the reading buffer. To optimize data access, all nodes can cache data from the node that stores the reading buffer data.

When you run Lua code remotely, it can cause reading buffer data that is held in the cache to become stale. If the values in the reading buffer change while the Lua code runs remotely, another node can hold stale values. Use the `clearcache()` command to clear the cache. For additional detail on the reading buffer cache commands, see [bufferVar.cachemode](#) (on page 15-19) and [bufferVar.clearcache\(\)](#) (on page 15-21).

The following example code demonstrates how stale values occur and how to use the `clearcache()` command to clear the cache on node 2, which is part of group 7.

```
-- Create a reading buffer on a node in a remote group.
node[2].tsplink.group = 7
node[2].execute("rbremote = smua.makebuffer(20)" ..
               "smua.measure.count = 20 " ..
               "smua.measure.v(rbremote)")
-- Create a variable on the local node to
-- access the reading buffer.
rblocal = node[2].getglobal("rbremote")
-- Access data from the reading buffer.
print(rblocal[1])
-- Run code on the remote node that updates the reading buffer.
node[2].execute("smua.measure.v(rbremote)")
-- Use the clearcache command if the reading buffer contains cached data.
rblocal.clearcache()
-- If you do not use the clearcache command, the data buffer
-- values never update. Every time the print command is
-- issued after the first print command, the same data buffer
-- values print.
print(rblocal[1])
```

TSP-Net

The TSP-Net® library allows the 2606B to control LAN-enabled devices directly through its LAN port. This enables the 2606B to communicate directly with a device that is not TSP® enabled without the use of a controlling computer.

TSP-Net capabilities

The TSP-Net library permits the 2606B to control a remote instrument through the LAN port for both Test Script Processor (TSP®) and non-TSP instruments. 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.

You can use TSP-Net commands to communicate with any ethernet-enabled instrument. However, 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 2606B can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The 2606B 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 simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the 2606B.

Using TSP-Net with any ethernet-enabled instrument

NOTE

Refer to [TSP command reference](#) (on page 15-1) for details about the commands presented in this section.

The 2606B 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 2606B 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 is 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 15-393) 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 is 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 15-393) for additional detail.

To communicate to a remote ethernet device from the 2606B:

1. Connect to the remote device using one of the above procedures. If the 2606B 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 2606B 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 2606B 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 2606B,04089762,1.6.3d
```

TSP-Net compared to TSP-Link to communicate with TSP-enabled devices

The TSP-Link® network interface is the preferred communication method for most applications where communication occurs between the 2606B and another TSP-enabled instrument.

One of the advantages of using the TSP-Link network interface is that TSP-Link connections have three trigger lines that are available to each device on the TSP-Link network. You can use any one of the trigger lines to perform hardware triggering between devices on the TSP-Link network. Refer to [Hardware trigger modes](#) (on page 7-29) for details.

However, if the distance between the 2606B and the TSP-enabled device is longer than 15 feet, use TSP-Net commands.

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

[tspnet.clear\(\)](#) (on page 15-392)
[tspnet.connect\(\)](#) (on page 15-393)
[tspnet.disconnect\(\)](#) (on page 15-394)
[tspnet.execute\(\)](#) (on page 15-395)
[tspnet.idn\(\)](#) (on page 15-396)
[tspnet.read\(\)](#) (on page 15-397)
[tspnet.readavailable\(\)](#) (on page 15-398)
[tspnet.reset\(\)](#) (on page 15-399)
[tspnet.termination\(\)](#) (on page 15-399)
[tspnet.timeout](#) (on page 15-400)
[tspnet.write\(\)](#) (on page 15-404)

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

[tspnet.tsp.abort\(\)](#) (on page 15-401)
[tspnet.tsp.abortonconnect](#) (on page 15-401)
[tspnet.tsp.rhtablecopy\(\)](#) (on page 15-402)
[tspnet.tsp.runscript\(\)](#) (on page 15-403)

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()

```

TSP command reference

In this section:

TSP command programming notes.....	15-1
Using the TSP command reference	15-3
TSP commands.....	15-7

TSP command programming notes

This section contains general information about using TSP commands.

Placeholder text

This manual uses italicized text to represent the parts of remote commands that must be replaced by user specified values. The following examples show typical uses of italicized text.

Example 1:

```
beeper.enable = state
```

Where *state* can be a value (`beeper.ON` or `beeper.OFF`) or an integer (1 or 0) that you specify. For example, to set this attribute on, you send one of the following commands:

```
beeper.enable = beeper.ON
```

```
beeper.enable = 1
```

Example 2:

```
digio.trigger[N].assert()
```

Where *N* is an integer (1 to 14) that you specify. For example, to assert trigger line 7, you send:

```
digio.trigger[7].assert()
```

To assert a trigger line with a variable as the integer, you send:

```
triggerline = 7
```

```
digio.trigger[triggerline].assert()
```

Example 3:

```
smuX.trigger.measure.Y(rbuffer)
```

Where:

X refers to the source-measure unit (SMU) channel (use *a* for SMU A).

Y is the measurement type that you specify (*v*, *i*, *r*, or *p*).

rbuffer is the reading buffer object where the readings are stored.

For example, to use SMU A to make voltage measurements and store them in buffer *vbuffername*, you send:

```
smua.trigger.measure.v(vbuffername)
```

Syntax rules

Use these syntax requirements to build well-formed instrument control commands.

Instrument commands are case sensitive. Refer to the command reference descriptions for the correct case.

The white space in lists of parameters in functions is optional. For example, the following functions are equivalent:

```
digio.writebit(3,0)
digio.writebit (3, 0)
```

All functions must have a set of parentheses () immediately following the function, even if there are no parameters specified. For example:

```
waitcomplete(G)
timezone = localtime.gettimezone()
```

If there are multiple parameters, they must be separated by commas (,). For example:

```
beeper.beep(0.5, 2400)
```

Time and date values

Time and date values are represented as the number of seconds since some base. Representing time as a number of seconds is referred to as “standard time format.” There are three time bases:

- **UTC 12:00 am Jan 1, 1970.** Some examples of UTC time are reading buffer base timestamps, adjustment dates, and the value returned by `os.time()`.
- **Instrument on.** References time to when the instrument was turned on. The value returned by `os.clock()` is referenced to the turn-on time.
- **Event.** Time referenced to an event, such as the first reading stored in a reading buffer.

Using the TSP command reference

The Test Script Processor (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 130: Example instrument command description

beeper.enable				
This attribute allows you to turn the beeper on or off.				
Type	TSP Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Recall setup Instrument reset	Saved setup Create configuration script	1 (beeper.ON)
Usage				
<pre>state = beeper.enable beeper.enable = state</pre>				
state beeper.OFF or 0: Beeper disabled beeper.ON or 1: Beeper enabled				
Details				
Disabling the beeper also disables front panel key clicks.				
Example				
<pre>beeper.enable = beeper.ON beeper.beep(2, 2400)</pre>				
Enables the beeper and generates a two-second, 2400 Hz tone				
Also see				
beeper.beep() (on page 8-10)				

The subsections contain information about the command. The subsections are:

- 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 brief description and a table with relevant information for each command. Definitions for the numbered items in the figure below are listed following the figure.

Figure 131: TSP command name and summary table

1. **feature.enable**

2. This command is an example of a typical TSP command that turns an instrument on or off.

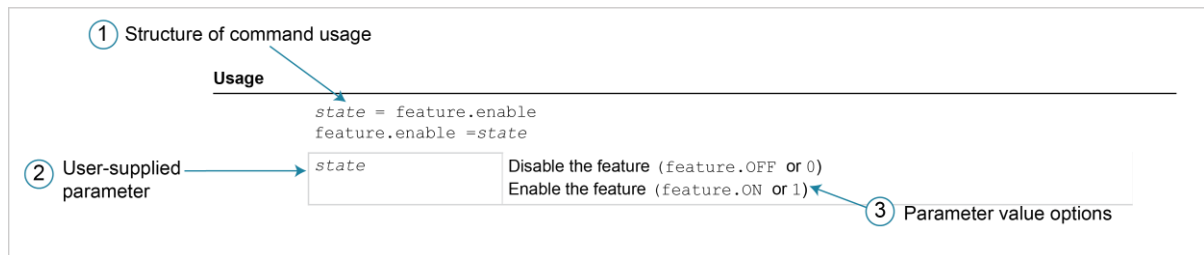
Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	1 (feature.ENABLE)

1. **Instrument command name.** Indicates 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). For detail on commands, see [Introduction to TSP operation](#) (on page 13-1).
3. **TSP-Link accessible.** **Yes** or **No**; indicates whether or not the command can be accessed through a TSP-Link network.
4. **Affected by.** Commands or actions that may change the setting of this command.
 - **LAN restore defaults:** This command is reset to the default value when `lan.restoredefaults()` is sent.
 - **Digital I/O trigger N reset:** This command is reset to the default value when `digio.trigger[N].reset()` is sent.
 - **Recall setup:** This command is stored as part of the saved setup and is changed to the value stored in the saved setup when the setup is recalled.
 - **Instrument reset:** This command is reset to the default value when `reset()`, `localnode.reset()`, or `*RST` is sent.
 - **SMU reset:** This command is reset to the default value when `smuX.reset()` is sent.
 - **Power cycle:** This command is set to the default value when the instrument power is cycled.
5. **Where saved.** Indicates where the command settings reside once they are used on an instrument. Options include:
 - **Not saved:** Command is not saved anywhere and must be typed each time you use it.
 - **Nonvolatile memory:** Storage area in the instrument where information is saved when the instrument is turned off.
 - **Saved setup:** Command is saved as part of the saved setup.
6. **Default value:** Lists the default value or constant 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.

Figure 132: TSP usage description



- 1 Structure of command usage:** Shows how to organize the parts of the command. 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 133: TSP Details description

Details
This command is a typical example of a command that enables or disables a feature.

Example section

The Example section of the remote command description shows examples of how you can use the command.

Figure 134: 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 135: TSP Also see description

Also see
exampleUnit.enable() (on page 7-8)

TSP commands

The TSP commands available for the instrument are listed in alphabetical order.

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)
```

<i>duration</i>	The amount of time to play the tone (0.001 s to 100 s)
<i>frequency</i>	The frequency of the tone in Hertz (Hz)

Details

You can use the beeper of the 2606B to provide an audible signal at a specified frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

The beeper does not sound if it is disabled. It can be disabled or enabled with the beeper enable command, or through the virtual front panel.

Example

```
beeper.enable = beeper.ON
beeper.beep(2, 2400)
```

Enables the beeper and generates a two-second, 2400 Hz tone.

Also see

[beeper.enable](#) (on page 15-8)

beeper.enable

This command allows you to turn the beeper on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Recall setup Instrument reset	Saved setup	1 (beeper.ON)

Usage

```
state = beeper.enable
beeper.enable = state
```

<i>state</i>	Disable the beeper: <code>beeper.OFF</code> or 0 Enable the beeper: <code>beeper.ON</code> or 1

Details

This command enables or disables the beeper. When enabled, a beep signals that a front-panel key has been pressed. Disabling the beeper also disables virtual front-panel key clicks.

Example

```
beeper.enable = beeper.ON
beeper.beep(2, 2400)
```

Enables the beeper and generates a two-second, 2400 Hz tone.

Also see

[beeper.beep\(\)](#) (on page 15-7)

bit.bitand()

This function performs a bitwise logical AND operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitand(value1, value2)
```

<i>result</i>	Result of the logical AND operation
<i>value1</i>	Operand for the logical AND operation
<i>value2</i>	Operand for the logical AND operation

Details

Any fractional parts of *value1* and *value2* are truncated to form integers. The returned *result* is also an integer.

Example

```
testResult = bit.bitand(10, 9)
print(testResult)
```

Performs a logical AND operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 8 (binary 1000).

Output:
8.00000e+00

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.bitor\(\)](#) (on page 15-9)

[bit.bitxor\(\)](#) (on page 15-10)

bit.bitor()

This function performs a bitwise logical OR operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitor(value1, value2)
```

<i>result</i>	Result of the logical OR operation
<i>value1</i>	Operand for the logical OR operation
<i>value2</i>	Operand for the logical OR operation

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

Example

```
testResult = bit.bitor(10, 9)
print(testResult)
```

Performs a bitwise logical OR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 11 (binary 1011).

Output:
1.10000e+01

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.bitand\(\)](#) (on page 15-8)

[bit.bitxor\(\)](#) (on page 15-10)

bit.bitxor()

This function performs a bitwise logical XOR (exclusive OR) operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitxor(value1, value2)
```

<i>result</i>	Result of the logical XOR operation
<i>value1</i>	Operand for the logical XOR operation
<i>value2</i>	Operand for the logical XOR operation

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

Example

```
testResult = bit.bitxor(10, 9)
print(testResult)
```

Performs a logical XOR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 3 (binary 0011).

Output:
3.000000e+00

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.bitand\(\)](#) (on page 15-8)

[bit.bitor\(\)](#) (on page 15-9)

bit.clear()

This function clears a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.clear(value, index)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within value to clear (1 to 32)

Details

Any fractional part of *value* is truncated to make it an integer. The returned *result* is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Example

```
testResult = bit.clear(15, 2)
print(testResult)
```

The binary equivalent of decimal 15 is 1111. If you clear the bit at *index* position 2, the returned decimal value is 13 (binary 1101).

Output:

```
1.30000e+01
```

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.set\(\)](#) (on page 15-13)

[bit.test\(\)](#) (on page 15-15)

[bit.toggle\(\)](#) (on page 15-16)

bit.get()

This function retrieves the weighted value of a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.get(value, index)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> to get (1 to 32)

Details

This function returns the value of the bit in *value* at *index*. This is the same as returning *value* with all other bits set to zero (0).

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for the number is set to zero (0), the result is zero (0).

Example

```
testResult = bit.get(10, 4)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. If you get the bit at index position 4, the returned decimal value is 8.

Output:

```
8.00000e+00
```

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.clear\(\)](#) (on page 15-10)

[bit.set\(\)](#) (on page 15-13)

[bit.test\(\)](#) (on page 15-15)

[bit.toggle\(\)](#) (on page 15-16)

bit.getfield()

This function returns a field of bits from the value starting at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.getfield(value, index, width)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> to get (1 to 32)
<i>width</i>	The number of bits to include in the field (1 to 32)

Details

A field of bits is a contiguous group of bits. This function retrieves a field of bits from *value* starting at *index*.

The *index* position is the least significant bit of the retrieved field. The number of bits to return is specified by *width*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Example

```
myResult = bit.getfield(13, 2, 3)
print(myResult)
```

The binary equivalent of decimal 13 is 1101.

The field at *index* position 2 and *width* 3 consists of the binary bits 110. The returned value is decimal 6 (binary 110).

Output:

```
6.000000e+00
```

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.get\(\)](#) (on page 15-11)

[bit.set\(\)](#) (on page 15-13)

[bit.setfield\(\)](#) (on page 15-14)

bit.set()

This function sets a bit at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.set(value, index)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> to set (1 to 32)

Details

This function returns *result*, which is *value* with the indexed bit set. The *index* must be between 1 and 32.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Any fractional part of *value* is truncated to make it an integer.

Example

```
testResult = bit.set(8, 3)
```

```
print(testResult)
```

The binary equivalent of decimal 8 is 1000. If the bit at *index* position 3 is set to 1, the returned value is decimal 12 (binary 1100).

Output:

```
1.20000e+01
```

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.clear\(\)](#) (on page 15-10)

[bit.get\(\)](#) (on page 15-11)

[bit.getfield\(\)](#) (on page 15-12)

[bit.setfield\(\)](#) (on page 15-14)

[bit.test\(\)](#) (on page 15-15)

[bit.toggle\(\)](#) (on page 15-16)

bit.setfield()

This function overwrites a bit field at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.setfield(value, index, width, fieldValue)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position in <i>value</i> to set (1 to 32)
<i>width</i>	The number of bits to include in the field (1 to 32)
<i>fieldValue</i>	Value to write to the field

Details

This function returns *result*, which is *value* with a field of bits overwritten, starting at *index*. The *index* specifies the position of the least significant bit of *value*. The *width* bits starting at *index* are set to *fieldValue*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Before setting the field of bits, any fractional parts of *value* and *fieldValue* are truncated to form integers.

If *fieldValue* is wider than *width*, the most significant bits of the *fieldValue* that exceed the width are truncated. For example, if *width* is 4 bits and the binary value for *fieldValue* is 11110 (5 bits), the most significant bit of *fieldValue* is truncated and a binary value of 1110 is used.

Example

```
testResult = bit.setfield(15, 2, 3, 5)
print(testResult)
```

The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at *index* position 2, the returned *value* is decimal 11 (binary 1011).

Output:

```
1.10000e+01
```

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.get\(\)](#) (on page 15-11)

[bit.set\(\)](#) (on page 15-13)

[bit.getfield\(\)](#) (on page 15-12)

bit.test()

This function returns the Boolean value (`true` or `false`) of a bit at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.test(value, index)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within value to test (1 to 32)

Details

This function returns *result*, which is the result of the tested bit.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for *value* is 0, *result* is `false`. If the bit of *value* at *index* is 1, the returned value is `true`.

If *index* is bigger than the number of bits in *value*, the result is `false`.

Example

```
testResult = bit.test(10, 4)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. Testing the bit at *index* position 4 returns a Boolean value of `true`.

Output:
`true`

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.clear\(\)](#) (on page 15-10)

[bit.get\(\)](#) (on page 15-11)

[bit.set\(\)](#) (on page 15-13)

[bit.toggle\(\)](#) (on page 15-16)

bit.toggle()

This function toggles the value of a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.toggle(value, index)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> to toggle (1 to 32)

Details

This function returns *result*, which is the result of toggling the bit *index* in *value*.

Any fractional part of *value* is truncated to make it an integer. The returned value is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

The indexed bit for *value* is toggled from 0 to 1, or 1 to 0.

Example

```
testResult = bit.toggle(10, 3)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. Toggling the bit at *index* position 3 returns a decimal value of 14 (binary 1110).

Output:
1.40000e+01

Also see

[Bit manipulation and logic operations](#) (on page 13-4)

[bit.clear\(\)](#) (on page 15-10)

[bit.get\(\)](#) (on page 15-11)

[bit.set\(\)](#) (on page 15-13)

[bit.test\(\)](#) (on page 15-15)

bufferVar.appendmode

This attribute sets the state of the append mode of the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See Details	0 (disabled)

Usage

```
state = bufferVar.appendmode
bufferVar.appendmode = state
```

<i>state</i>	The reading buffer append mode; set to one of the following: 0: Append mode off; new measurement data overwrites the previous buffer content 1: Append mode on; appends new measurement data to the present buffer content
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute enables or disables the buffer append mode. This value can only be changed with an empty buffer. Use `bufferVar.clear()` to empty the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

If the append mode is set to 0, any stored readings in the buffer are cleared before new ones are stored. If append mode is set to 1, any stored readings remain in the buffer and new readings are added to the buffer after the stored readings.

With append mode on, the first new measurement is stored at `rb[n+1]`, where *n* is the number of readings stored in buffer `rb`.

Example

```
buffer1.appendmode = 1
```

Append new readings to contents of the reading buffer named `buffer1`.

Also see

[bufferVar.clear\(\)](#) (on page 15-21)

[Reading buffers](#) (on page 6-1)

bufferVar.basetimestamp

This attribute contains the timestamp that indicates when the first reading was stored in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	See Details	0

Usage

```
basetime = bufferVar.basetimestamp
```

<i>basetime</i>	The timestamp of the first stored reading
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)

Details

This read-only attribute contains the timestamp (in seconds) of the first reading stored in a buffer (`rb[1]` stored in reading buffer `rb`). The timestamp is the number of seconds since 12:00 am January 1, 1970 (UTC) that the measurement was performed and stored.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

See the `smuX.nvbufferY` attribute for details on accessing dedicated reading buffers.

Example

```
basetime = smua.nvbuffer1.basetimestamp
print(basetime)
```

Read the timestamp for the first reading stored in dedicated reading buffer 1.

Output:

```
1.57020e+09
```

This output indicates that the timestamp is 1,570,200,000 seconds (which is Friday, October 4, 2019 at 14:40:00 pm).

Also see

[Reading buffers](#) (on page 6-1)

[smuX.measure.overlappedY\(\)](#) (on page 15-223)

[smuX.measure.Y\(\)](#) (on page 15-228)

[smuX.nvbufferY](#) (on page 15-230)

[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

bufferVar.cachemode

This attribute enables or disables the reading buffer cache (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not saved	1 (enabled)

Usage

```
cacheMode = bufferVar.cachemode  
bufferVar.cachemode = cacheMode
```

<i>cacheMode</i>	The reading buffer cache mode; set to one of the following: 0: Cache mode disabled (off) 1: Cache mode enabled (on)
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute enables or disables the reading buffer cache. When enabled, the reading buffer cache improves access speed to reading buffer data.

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when initiating successive sweeps without reconfiguring the sweep measurements or when overwriting data in the reading buffer by setting the *bufferVar.fillmode* attribute to `smuX.FILL_WINDOW`. To avoid this, make sure that you include commands that automatically invalidate the cache as needed (for example, explicit calls to the *bufferVar.clearcache()* function) or disable the cache using this attribute (*bufferVar.cachemode*).

Example

<code>smua.nvbuffer1.cachemode = 1</code>	Enables reading buffer cache of dedicated reading buffer 1 (source-measure unit (SMU) channel A).

Also see

[bufferVar.clearcache\(\)](#) (on page 15-21)

[bufferVar.fillmode](#) (on page 15-25)

[Reading buffers](#) (on page 6-1)

bufferVar.capacity

This attribute sets the number of readings a buffer can store.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	See Details	Not applicable

Usage

```
bufferCapacity = bufferVar.capacity
```

<i>bufferCapacity</i>	The maximum number of readings the buffer can store
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This read-only attribute reads the number of readings that can be stored in the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

The capacity of the buffer does not change as readings fill the buffer. A dedicated reading buffer that only collects basic items can store over 140,000 readings. Turning on additional collection items, such as timestamps and source values, decreases the capacity of a dedicated reading buffer (for example, `smua.nvbuffer1`), but does not change the capacity of a user-defined dynamically allocated buffer. A user-defined dynamically allocated buffer has a fixed capacity that is set when the buffer is created.

See the `smuX.nvbufferY` attribute for details on accessing dedicated reading buffers. See the `smuX.makebuffer()` function for information on creating user-defined dynamically allocated reading buffers.

Example

```
bufferCapacity = smua.nvbuffer1.capacity
print(bufferCapacity)
```

Reads the capacity of dedicated reading buffer 1 (source-measure unit (SMU) channel A).

Output:

```
1.49789e+05
```

The above output indicates that the buffer can hold 149789 readings.

Also see

[Reading buffers](#) (on page 6-1)
[smuX.makebuffer\(\)](#) (on page 15-210)
[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

bufferVar.clear()

This function empties the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar.clear()
```

<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This function clears all readings and related recall attributes from the buffer (for example, *bufferVar.timestamps* and *bufferVar.statuses*) from the specified buffer.

Example

<code>smua.nvbuffer1.clear()</code>	Clears dedicated reading buffer 1 (source-measure unit (SMU) channel A).

Also see

[Reading buffers](#) (on page 6-1)
[smuX.nvbufferY](#) (on page 15-230)

bufferVar.clearcache()

This function clears the cache.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar.clearcache()
```

<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This function clears all readings from the specified cache.

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you:

- Initiate successive sweeps without reconfiguring the sweep measurements. Watch for this when running Lua code remotely on more than one node, because values in the reading buffer cache may change while the Lua code is running.
- Overwrite data in the reading buffer by setting the *bufferVar.fillmode* attribute to `smuX.FILL_WINDOW`.

To avoid this, you can include explicit calls to the `bufferVar.clearcache()` function to remove stale values from the reading buffer cache.

Example

```
smua.nvbuffer1.clearcache()
```

Clears the reading buffer cache for dedicated reading buffer 1.

Also see

[bufferVar.fillmode](#) (on page 15-25)

[Reading buffers](#) (on page 6-1)

[Removing stale values from the reading buffer cache](#) (on page 14-63)

[smuX.nvbufferY](#) (on page 15-230)

bufferVar.collectsourcevalues

This attribute sets whether or not source values are stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See Details	0 (disabled)

Usage

```
state = bufferVar.collectsourcevalues
bufferVar.collectsourcevalues = state
```

<code>state</code>	Source value collection status; set to one of the following: 0: Source value collection disabled (off) 1: Source value collection enabled (on)
<code>bufferVar</code>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)

Details

Assigning a value to this attribute enables or disables the storage of source values. Reading this attribute returns the state of source value collection. This value can only be changed with an empty buffer. Empty the buffer using the `bufferVar.clear()` function.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

When on, source values are stored with readings in the buffer. This requires four extra bytes of storage for each reading. Turning on additional collection items, such as source values (this attribute) and timestamps, decreases the capacity of a dedicated reading buffer, but does not change the capacity of a user-defined dynamically allocated buffer.

You cannot collect source values when `smuX.trigger.measure.action` is set to `smuX.ASYNC`, so `bufferVar.collectsourcevalues` must be set to 0 when the measurement action is set to be asynchronous.

Example

```
smua.nvbuffer1.collectsourcevalues = 1
```

Include source values with readings for dedicated reading buffer 1.

Also see

[bufferVar.clear\(\)](#) (on page 15-21)
[Reading buffers](#) (on page 6-1)
[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.trigger.measure.action](#) (on page 15-257)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

bufferVar.collecttimestamps

This attribute sets whether or not timestamp values are stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See Details	0 (disabled)

Usage

```
state = bufferVar.collecttimestamps
bufferVar.collecttimestamps = state
```

<i>state</i>	Timestamp value collection status; set to one of the following: 0: Timestamp value collection disabled (off) 1: Timestamp value collection enabled (on)
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute enables or disables the storage of timestamps. Reading this attribute returns the state of timestamp collection.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

When on, timestamp values are stored with readings in the buffer. This requires four extra bytes of storage for each reading. Turning on additional collection items, such as timestamps (this attribute) and source values, decreases the capacity of a dedicated reading buffer (for example, `smua.nvbuffer1`), but does not change the capacity of a user-defined dynamically allocated buffer.

The *state* variable can only be changed when the buffer is empty. Empty the buffer using the `bufferVar.clear()` function.

Example

<code>smua.nvbuffer1.collecttimestamps = 1</code>	Include timestamps with readings for dedicated reading buffer 1.

Also see

[bufferVar.clear\(\)](#) (on page 15-21)
[Reading buffers](#) (on page 6-1)
[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

bufferVar.fillcount

This attribute sets the reading buffer fill count.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See Details	0

Usage

```
fillCount = bufferVar.fillcount
bufferVar.fillcount = fillCount
```

<i>fillCount</i>	The reading buffer fill count
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)

Details

The reading buffer fill count sets the number of readings to store before restarting at index 1. If the value is zero (0), then the capacity of the buffer is used. Use this attribute to control when the SMU restarts filling the buffer at index 1, rather than having it restart when the buffer is full.

If the `bufferVar.fillcount` attribute is set to a value higher than the capacity of the buffer, after storing the element at the end of the buffer, the SMU overwrites the reading at index 1, the reading after that overwrites the reading at index 2, and so on.

This attribute is only used when the `bufferVar.fillmode` attribute is set to `smuX.FILL_WINDOW`.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
smua.nvbuffer1.fillcount = 50
```

Sets fill count of dedicated reading buffer 1 to 50.

Also see

[bufferVar.fillmode](#) (on page 15-25)

bufferVar.fillmode

This attribute sets the reading buffer fill mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See Details	0 (smuX.FILL_ONCE)

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

<i>fillMode</i>	The reading buffer fill mode; set to one of the following: 0 or smuX.FILL_ONCE: Do not overwrite old data 1 or smuX.FILL_WINDOW: New readings restart at index 1 after acquiring reading at index <i>bufferVar.fillcount</i>
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <i>smua.nvbuffer1</i>)

Details

When this attribute is set to *smuX.FILL_ONCE*, the reading buffer does not overwrite readings. If the buffer fills up, new readings are discarded.

When this attribute is set to *smuX.FILL_WINDOW*, new readings are added after existing data until the buffer holds *bufferVar.fillcount* elements. Continuing the sequence, the next reading overwrites the reading at index 1, the reading after that overwrites the reading at index 2, and so on.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
smua.nvbuffer1.fillmode = smua.FILL_ONCE
Sets fill mode of dedicated reading buffer 1 to fill once (do not overwrite old data).
```

Also see

[bufferVar.fillcount](#) (on page 15-24)

[Reading buffers](#) (on page 6-1)

bufferVar.measurefunctions

This attribute contains the measurement function that was used to acquire a reading stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See Details	Not applicable

Usage

```
measurefunction = bufferVar.measurefunctions[N]
```

<i>measurefunction</i>	The measurement function used (Current, Voltage, Ohms, or Watts) to acquire reading number <i>N</i> in the specified buffer
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

The `measurefunctions` buffer recall attribute is like an array (a Lua table) of strings indicating the function measured for the reading.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
measurefunction = smua.nvbuffer1.measurefunctions[5]
```

Store the measure function used to make reading number 5.

Example 2

```
printbuffer(1, 5, smua.nvbuffer1.measurefunctions)
```

Print the measurement function that was used to measure the first five readings saved in dedicated reading buffer 1.

Example output:

```
Current, Current, Current, Current, Current
```

Also see

[bufferVar.measuranges](#) (on page 15-26)

[bufferVar.n](#) (on page 15-28)

[bufferVar.readings](#) (on page 15-29)

[bufferVar.sourcefunctions](#) (on page 15-30)

[bufferVar.sourceoutputstates](#) (on page 15-31)

[bufferVar.sourceranges](#) (on page 15-31)

[bufferVar.sourcevalues](#) (on page 15-32)

[bufferVar.statutes](#) (on page 15-34)

[bufferVar.timestamps](#) (on page 15-36)

[Reading buffers](#) (on page 6-1)

bufferVar.measuranges

This attribute contains the measurement range values that were used for readings stored in a specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See Details	Not applicable

Usage

```
measurangerange = bufferVar.measurangeranges[N]
```

<i>measurangerange</i>	The measurement range used to acquire reading number <i>N</i> in the specified buffer
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

The `measurangeranges` buffer recall attribute is like an array (a Lua table) of full-scale range values for the measure range used when the measurement was made.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
measurangerange = smua.nvbuffer1.measurangeranges[1]
Store the measure range that was used to make reading number 1.
```

Example 2

```
printbuffer(1, 10, smua.nvbuffer1.measurangeranges)
Print the range values that were used for the first 10 readings saved in dedicated reading buffer 1.
Example output:
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07
```

Also see

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statues](#) (on page 15-34)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

bufferVar.n

This attribute contains the number of readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See Details	Not applicable

Usage

```
numberOfReadings = bufferVar.n
```

<i>numberOfReadings</i>	The number of readings stored in the buffer
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This read-only attribute contains the number of readings presently stored in the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
numberOfReadings = smua.nvbuffer1.n
print(numberOfReadings)
```

Reads the number of readings stored in dedicated reading buffer 1 (source-measure unit (SMU) channel A).

Output:

```
1.25000+02
```

The above output indicates that there are 125 readings stored in the buffer.

Also see

[bufferVar.measurefunctions](#) (on page 15-25)

[bufferVar.measureranges](#) (on page 15-26)

[bufferVar.readings](#) (on page 15-29)

[bufferVar.sourcefunctions](#) (on page 15-30)

[bufferVar.sourceoutputstates](#) (on page 15-31)

[bufferVar.sourceranges](#) (on page 15-31)

[bufferVar.sourcevalues](#) (on page 15-32)

[bufferVar.statuses](#) (on page 15-34)

[bufferVar.timestamps](#) (on page 15-36)

[Reading buffers](#) (on page 6-1)

[smuX.measure.overlappedY\(\)](#) (on page 15-223)

[smuX.measure.Y\(\)](#) (on page 15-228)

[smuX.nvbufferY](#) (on page 15-230)

[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

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	Clearing the buffer	See Details	Not applicable

Usage

```
reading = bufferVar.readings[N]
```

<i>reading</i>	The value of the reading in the specified reading buffer
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

The *bufferVar.readings* buffer recall attribute is like an array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly; that is, *rb[2]* and *rb.readings[2]* access the same value.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
print(smua.nvbuffer1.readings[1])
```

Output the first reading saved in source-measure unit (SMU) channel A, dedicated reading buffer 1.
Output:
8.81658e-08

Also see

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statues](#) (on page 15-34)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

bufferVar.sourcefunctions

This attribute contains the source function that was being used when the readings were stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See Details	Not applicable

Usage

```
sourcefunction = bufferVar.sourcefunctions[N]
```

<i>sourcefunction</i>	The source function used (Current or Voltage) to acquire reading number <i>N</i> in the specified buffer
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

The `bufferVar.sourcefunctions` buffer recall attribute is like an array (a Lua table) of strings indicating the source function at the time of the measurement.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
sourcefunction = smua.nvbuffer1.sourcefunctions[3]
print(sourcefunction)
```

Store the source function used to make reading number 3 and output the value.

Example 2

```
printbuffer(1, 10, smua.nvbuffer1.sourcefunctions)
```

Print the source function used for 10 readings stored in dedicated reading buffer 1.
Example output:
Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage

Also see

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statues](#) (on page 15-34)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

bufferVar.sourceoutputstates

This attribute indicates the state of the source output for readings that are stored in a specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See Details	Not applicable

Usage

```
state = bufferVar.sourceoutputstates[N]
```

<i>state</i>	The output state (Off or On) when reading <i>N</i> of the specified buffer was acquired
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

The `bufferVar.sourceoutputstates` buffer recall attribute is similar to an array (a Lua table) of strings. This array indicates the state of the source output (Off or On) at the time of the measurement.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
printbuffer(1, 1, smua.nvbuffer1.sourceoutputstates)
Print the source output for the first reading stored in dedicated reading buffer 1.
Example output:
On
```

Also see

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measuranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statues](#) (on page 15-34)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

bufferVar.sourceranges

This attribute contains the source range that was used for readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See Details	Not applicable

Usage

```
sourcerange = bufferVar.sourceranges[N]
```

<i>sourcerange</i>	The source range used to acquire reading number <i>N</i> in the specified buffer
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

The `bufferVar.sourceranges` buffer recall attribute is like an array (a Lua table) of full-scale range values for the source range used when the measurement was made.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
sourcerange = smua.nvbuffer1.sourceranges[1]
```

Store the source range that was used for the first reading stored in dedicated reading buffer 1.

Example 2

```
printbuffer(1, 6, smua.nvbuffer1.sourceranges)
```

Print the source ranges that were used for the first 6 readings stored in source-measure unit (SMU) A, buffer 1.
Example output:

```
1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04
```

Also see

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statues](#) (on page 15-34)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

bufferVar.sourcevalues

When enabled by the `bufferVar.collectsourcevalues` attribute, 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	Clearing the buffer	See Details	Not applicable

Usage

```
sourcevalue = bufferVar.sourcevalues[N]
```

<i>sourcevalue</i>	The output value of the source when reading <i>N</i> of the specified buffer was acquired
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated buffer (user-defined) or a dedicated reading buffer (such as <code>smua.nvbuffer1</code>)
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

If the `bufferVar.collectsourcevalues` attribute is enabled before readings are made, the `bufferVar.sourcevalues` buffer recall attribute is like an array (a Lua table) of the sourced value in effect at the time of the reading.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
sourcevalue = smua.nvbuffer1.sourcevalues[1]
```

Get the sourced value of the first reading stored in dedicated reading buffer 1.

Example 2

```
printbuffer(1, 6, smua.nvbuffer1.sourcevalues)
```

Print the sourced value of the first 6 readings stored in source-measure unit (SMU) A, buffer 1.

Example output:

```
1.00000e-04, 1.00000e-04,
1.00000e-04, 1.00000e-04,
1.00000e-04, 1.00000e-04
```

Also see

[bufferVar.collectsourcevalues](#) (on page 15-22)
[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.statues](#) (on page 15-34)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

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	Clearing the buffer	See Details	Not applicable

Usage

```
statusInformation = bufferVar.statuses[N]
```

<i>statusInformation</i>	The status value when reading <i>N</i> of the specified buffer was acquired
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This read-only buffer recall attribute is like 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; see the following table for values.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Buffer status bits			
Bit	Name	Hex	Description
B1	Overtemp	0x02	Over temperature condition
B2	AutoRangeMeas	0x04	Measure range was autoranged
B3	AutoRangeSrc	0x08	Source range was autoranged
B4	4Wire	0x10	4-wire (remote) sense mode enabled
B5	Rel	0x20	Relative offset applied to reading
B6	Compliance	0x40	Source function was limited because the complementary function would be over the compliance limit
B7	Filtered	0x80	Reading was filtered

Example

```
reset()
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.autorangev = smua.AUTORANGE_ON
smua.source.levelv = 5
smua.source.limiti = 10e-3
smua.measure.rangei = 10e-3
smua.source.output = smua.OUTPUT_ON
print(smua.measure.i(smua.nvbuffer1))
smua.source.output = smua.OUTPUT_OFF

print(smua.nvbuffer1.statuses[1])
```

Reset the instrument.
Set the voltage source function to dc volts.
Set the range to auto.
Set the voltage source to 5 V.
Set current measure limit to 10 mA.
Set the current measure range to 10 mA.
Turn on the output.
Print and place the current reading in the reading buffer.
Turn off the output.
Output status value of the first measurement in the reading buffer.
Output example:
3.99470e-06
4.00000e+00

Also see

[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)

bufferVar.timestampresolution

This attribute contains the resolution of the timestamp.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See Details	1e-6 (1 µs)

Usage

```
resolution = bufferVar.timestampresolution
```

<i>resolution</i>	Timestamp resolution in seconds (minimum 1 µs; rounded to an even power of 2 µs)
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute sets the resolution for the timestamps. Reading this attribute returns the timestamp resolution value. This value can only be changed with an empty buffer. Empty the buffer using the `bufferVar.clear()` function.

The finest timestamp resolution is 0.000001 seconds (1 µs). At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. You can increase this value for very long tests.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
smua.nvbuffer1.timestampresolution = 0.000008
```

Sets the timestamp resolution of dedicated reading buffer 1 to 8 μ s.

Also see

[bufferVar.clear\(\)](#) (on page 15-21)
[bufferVar.collecttimestamps](#) (on page 15-23)
[bufferVar.timestamps](#) (on page 15-36)
[Reading buffers](#) (on page 6-1)
[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

bufferVar.timestamps

When enabled by the *bufferVar.collecttimestamps* attribute, 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	Clearing the buffer	See Details	Not applicable

Usage

```
timestamp = bufferVar.timestamps[N]
```

<i>timestamp</i>	The complete timestamp (including date, time, and fractional seconds) of reading number <i>N</i> in the specified reading buffer when the reading was acquired
<i>bufferVar</i>	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer
<i>N</i>	The reading number (1 to <i>bufferVar.n</i>)

Details

The *bufferVar.timestamps* information from a reading buffer is only available if the *bufferVar.collecttimestamps* attribute is set to 1 (default setting). If it is set to 0, you cannot access any time information from a reading buffer.

If enabled, this buffer recall attribute is like an array (a Lua table) that contains timestamps, in seconds, of when each reading occurred. These are relative to the *bufferVar.basetimestamp* for the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
timestamp = smua.nvbuffer1.timestamps[1]
```

Get the timestamp of the first reading stored in source-measure unit (SMU) A, buffer 1.

Also see

[bufferVar.clear\(\)](#) (on page 15-21)
[bufferVar.collecttimestamps](#) (on page 15-23)
[bufferVar.measurefunctions](#) (on page 15-25)
[bufferVar.measureranges](#) (on page 15-26)
[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[bufferVar.sourcefunctions](#) (on page 15-30)
[bufferVar.sourceoutputstates](#) (on page 15-31)
[bufferVar.sourceranges](#) (on page 15-31)
[bufferVar.sourcevalues](#) (on page 15-32)
[bufferVar.statutes](#) (on page 15-34)
[Reading buffers](#) (on page 6-1)

ConfigPulseIMeasureV()

This KIPulse factory script function configures a current pulse train with a voltage measurement at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```

f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag)
  
```

<i>f</i>	A Boolean flag; this flag is <code>true</code> when the pulse was successfully configured, <code>false</code> when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <code>false</code> , <i>msg</i> contains an error message; if it is <code>true</code> , <i>msg</i> contains a string that indicates successful configuration
<i>smu</i>	Instrument channel (for example, <code>smua</code> refers to SMU channel A)
<i>bias</i>	Bias level in amperes
<i>level</i>	Pulse level in amperes
<i>limit</i>	Voltage limit (for example, compliance) in volts
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles
<i>buffer</i>	Reading buffer where pulsed measurements are stored; if this is <code>nil</code> when the function is called, no measurements are made when the pulse train is initiated
<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>sync_in</i>	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
<i>sync_out</i>	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of <i>ton</i>

<code>sync_in_timeout</code>	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
<code>sync_in_abort</code>	Specifies whether or not to abort the pulse if an input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; <code>true</code> or <code>false</code>

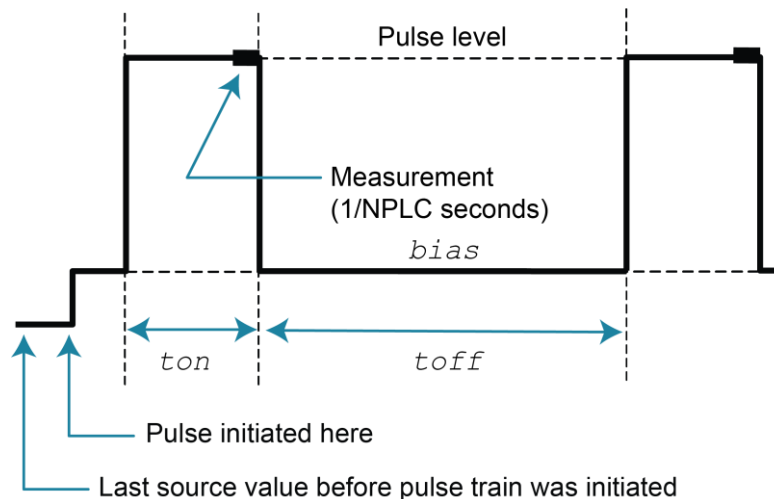
Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the `buffer` input parameter.

This function configures a current pulse train with a voltage measurement at each point. Measurements are made at the end of the `ton` time.

This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all the pulse dimensions can be achieved, and if they are, assigns the indicated `tag` or index to the pulse train. The `InitiatePulseTest(tag)` and `InitiatePulseTestDual(tag1, tag2)` functions are used to initiate a pulse train assigned to a valid `tag`.

Figure 136: ConfigPulseMeasureV()



Example

```
ConfigPulseMeasureV(smua, 0, 5, 10, 0.001, 0.080, 1, smua.nvbuffer1, 1)
```

Set up a pulse train that uses channel A. The pulse amplitude is 5 A and returns to 0 A after 1 ms. The pulse remains at 0 A for 80 ms and the voltage limit is 10 V during the pulse. The pulse train consists of only 1 pulse, and this pulse is assigned a `tag` index of 1.

Also see

[InitiatePulseTest\(\)](#) (on page 15-113)
[InitiatePulseTestDual\(\)](#) (on page 15-115)
[KIPulse factory script](#) (on page 13-20)

ConfigPulseIMeasureVSweepLin()

This KIPulse factory script function configures a linear pulsed current sweep with a voltage measurement at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag)
```

<i>f</i>	A Boolean flag; this flag is <i>true</i> if the pulse was successfully configured, <i>false</i> when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <i>false</i> , <i>msg</i> contains an error message; if it is <i>true</i> , <i>msg</i> contains a string indicating successful configuration
<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>bias</i>	Bias level in amperes
<i>start</i>	Pulse sweep start level in amperes
<i>stop</i>	Pulse sweep stop level in amperes
<i>limit</i>	Voltage limit (for example, compliance) in volts
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles
<i>buffer</i>	Reading buffer where pulsed measurements are stored; if this is <i>nil</i> when the function is called, no measurements are made when the pulse train is initiated
<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>sync_in</i>	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
<i>sync_out</i>	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of <i>ton</i>
<i>sync_in_timeout</i>	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
<i>sync_in_abort</i>	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; <i>true</i> or <i>false</i>

Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the *buffer* input parameter.

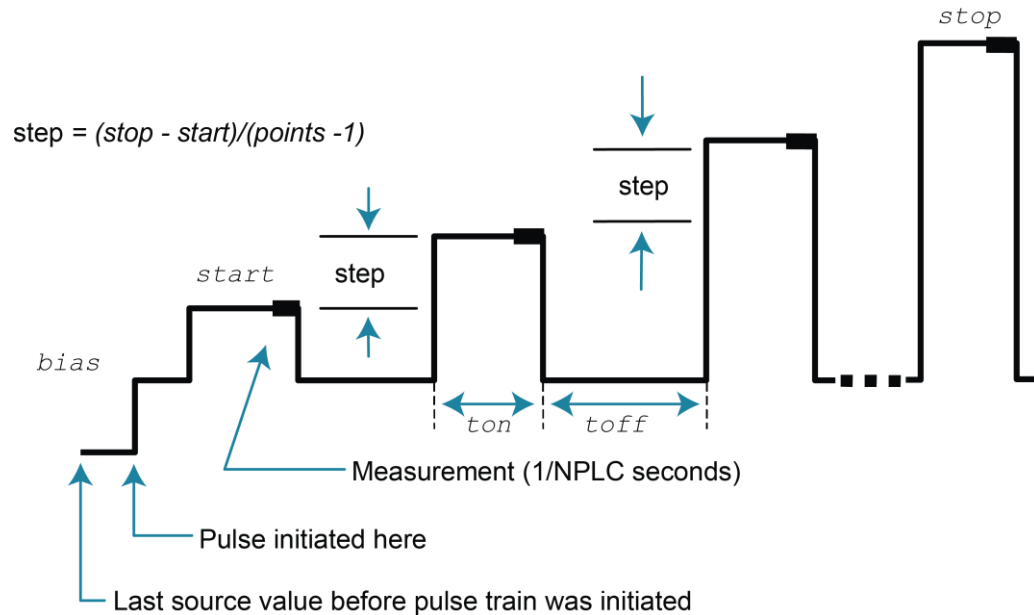
This function configures a linear pulsed current sweep with a voltage measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse is *start* amperes; the magnitude of the last pulse is *stop* amperes. The magnitude of each pulse in between is *step* amperes larger than the previous pulse, where:

$$step = (stop - start) / (points - 1)$$

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(tag)` and `InitiatePulseTestDual(tag1, tag2)` functions are used to initiate a pulse train assigned to a valid *tag*.

Figure 137: ConfigPulseMeasureVSweepLin()



Example

```
ConfigPulseMeasureVSweepLin(smua, 0, 0.01, 0.05, 1, 1e-3, 0.1, 20, smua.nvbuffer2, 3)
```

Set up a pulsed sweep that uses channel A. The pulsed sweep starts at 10 mA, ends at 50 mA, and returns to a 0 mA bias level between pulses. Each pulsed step is on for 1 ms, and then at the bias level for 100 ms. The voltage limit is 1 V during the entire pulsed sweep. The pulse train is comprised of 20 pulsed steps and the pulse train is assigned a *tag* index of 3.

Also see

[InitiatePulseTest\(\)](#) (on page 15-113)

[InitiatePulseTestDual\(\)](#) (on page 15-115)

[KIPulse factory script](#) (on page 13-20)

ConfigPulseIMeasureVSweepLog()

This [KIPulse factory script](#) (on page 13-20) function configures a voltage pulse train with a current measurement at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag)
```

<i>f</i>	A Boolean flag; this flag is true when the pulse was successfully configured, false when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is false , <i>msg</i> contains an error message; if it is true , <i>msg</i> contains a string indicating successful configuration
<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>bias</i>	Bias level in amperes
<i>start</i>	Pulse sweep start level in amperes
<i>stop</i>	Pulse sweep stop level in amperes
<i>limit</i>	Voltage limit (for example, compliance) in volts
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles
<i>buffer</i>	Reading buffer where pulsed measurements are stored; if this is nil when the function is called, no measurements are made when the pulse train is initiated
<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>sync_in</i>	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
<i>sync_out</i>	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of <i>ton</i>
<i>sync_in_timeout</i>	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
<i>sync_in_abort</i>	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; true or false

Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a logarithmic pulsed current sweep with a voltage measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse is *start* amperes; the magnitude of the last pulse is *stop* amperes. The magnitude of each pulse in between is LogStep_n amperes larger than the previous pulse, where:

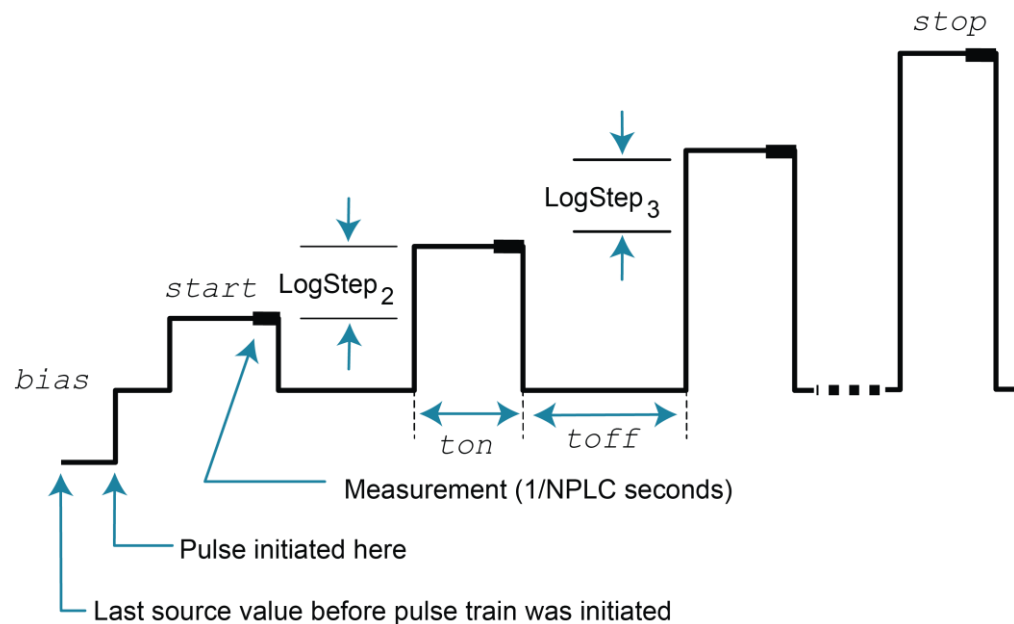
$$\text{LogStepSize} = (\log_{10}(\text{stop}) - \log_{10}(\text{start})) / (\text{points} - 1)$$

$$\text{LogStep}_n = (n - 1) * (\text{LogStepSize}), \text{ where } n = [2, \text{points}]$$

$$\text{SourceStepLevel}_n = \text{antilog}(\text{LogStep}_n) * \text{start}$$

This function does not cause the specified *smu* to output a pulse train. It simply checks to see if all of the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. To initiate a pulse train assigned to a valid *tag*, use `InitiatePulseTest(tag)` and `InitiatePulseTestDual(tag1, tag2)`.

Figure 138: ConfigPulseMeasureVSweepLog()



Example

```
ConfigPulseMeasureVSweepLog(smua, 0, 1e-3, 0.01, 1, 1e-3, 10e-3, 10,
smua.nvbuffer1, 5)
```

Set up a pulsed logarithmic sweep that uses channel A. The pulsed sweep starts at 1 mA, ends at 10 mA, and returns to a 0 A bias level between pulses. Each pulsed step is on for 1 ms, and then at the bias level for 10 ms. The voltage limit is 1 V during the entire pulsed sweep. The pulse train is comprised of 10 pulsed steps, and the pulse train is assigned a *tag* index of 5.

Also see

[InitiatePulseTest\(\)](#) (on page 15-113)

[InitiatePulseTestDual\(\)](#) (on page 15-115)

[KIPulse factory script](#) (on page 13-20)

ConfigPulseVMeasureI()

This KIPulse factory script function configures a voltage pulse train with a current measurement at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer, tag)
```

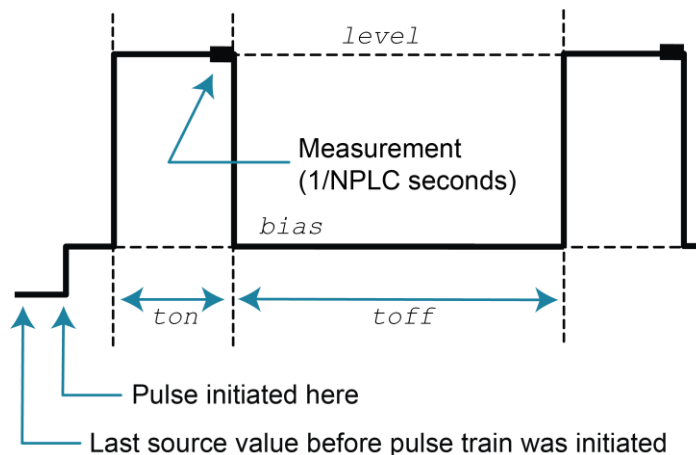
<i>f</i>	A Boolean flag; this flag is <code>true</code> when the pulse was successfully configured, <code>false</code> when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <code>false</code> , <i>msg</i> contains an error message; if it is <code>true</code> , <i>msg</i> contains a string indicating successful configuration
<i>smu</i>	Instrument channel (for example, <code>smua</code> refers to SMU channel A)
<i>bias</i>	Bias level in volts
<i>level</i>	Pulse level in volts
<i>limit</i>	Current limit (for example, compliance) in amperes
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles
<i>buffer</i>	Reading buffer where pulsed measurements are stored; if this is <code>nil</code> when the function is called, no measurements are made when the pulse train is initiated
<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>sync_in</i>	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
<i>sync_out</i>	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of <i>ton</i>
<i>sync_in_timeout</i>	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
<i>sync_in_abort</i>	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; <code>true</code> or <code>false</code>

Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a voltage pulse train with a current measurement at each point. Measurements are made at the end of the *ton* time.

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. To initiate a pulse train assigned to a valid *tag*, use `InitiatePulseTest(tag)` and `InitiatePulseTestDual(tag1, tag2)`.

Figure 139: ConfigPulseVMeasureI()**Example 1**

```
ConfigPulseVMeasureI(smua, 0, 20, 1, 0.001, 0.080, 10, smua.nvbuffer1, 2)
```

Set up a pulse train that uses channel A. The pulse amplitude is 20 V and returns to 0 V after 1 ms. The pulse remains at 0 V for 80 ms, and the current limit is 1 A during the pulse. The pulse train consists of 10 pulses, and the pulse train is assigned a *tag* index of 2.

Example 2

```
local timelist = { 1, 2, 3, 4, 5 }

f, msg = ConfigPulseVMeasureI(smua, 0, 1,
    100e-3, 1, timelist, 5, nil, 1)
```

Variable off time between pulses in a pulse train.

Configure a pulse with 1 second on-time and variable off-time, no measurement.

Example 3

```
rbi = smua.makebuffer(10)
rbv = smua.makebuffer(10)
rbi.appendmode = 1
rbv.appendmode = 1
rbs = { i = rbi, v = rbv }

f, msg = ConfigPulseVMeasureI(smua, 0, 10, 1e-3, 1e-3, 1e-3, 2, rbs, 1)
```

Simultaneous IV measurement during pulse.

Also see

[InitiatePulseTest\(\)](#) (on page 15-113)

[InitiatePulseTestDual\(\)](#) (on page 15-115)

[KIPulse factory script](#) (on page 13-20)

ConfigPulseVMeasureISweepLin()

This KIPulse factory script function configures a voltage pulse train with a current measurement at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag)
```

<i>f</i>	A Boolean flag; this flag is <code>true</code> when the pulse was successfully configured, <code>false</code> when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <code>false</code> , <i>msg</i> contains an error message; if it is <code>true</code> , <i>msg</i> contains a string indicating successful configuration
<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>bias</i>	Bias level in volts
<i>start</i>	Pulse sweep start level in volts
<i>stop</i>	Pulse sweep stop level in volts
<i>limit</i>	Current limit (for example, compliance) in amperes
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles
<i>buffer</i>	Reading buffer where pulsed measurements are stored; if this is <code>nil</code> when the function is called, no measurements are made when the pulse train is initiated
<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>sync_in</i>	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
<i>sync_out</i>	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of <i>ton</i>
<i>sync_in_timeout</i>	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
<i>sync_in_abort</i>	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; <code>true</code> or <code>false</code>

Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a linear pulsed voltage sweep with a current measurement at each point. Measurements are made at the end of the *ton* time.

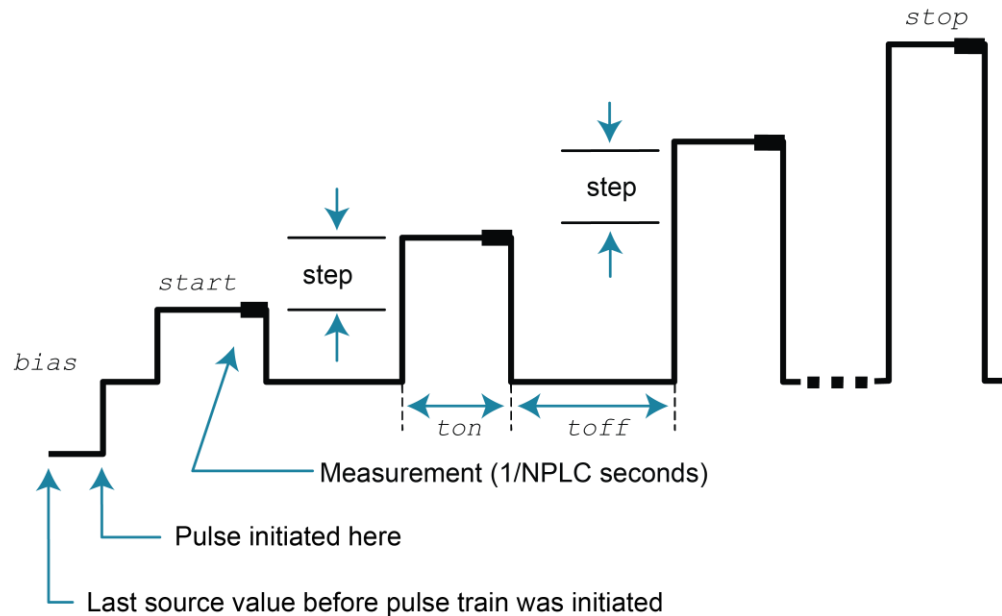
The magnitude of the first pulse is *start* volts; the magnitude of the last pulse is *stop* volts. The magnitude of each pulse in between is *step* volts larger than the previous pulse, where:

$$step = (stop - start) / (points - 1)$$

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train.

The `InitiatePulseTest(tag)` and `InitiatePulseTestDual(tag1, tag2)` functions are used to initiate a pulse train assigned to a valid *tag*.

Figure 140: ConfigPulseVMeasureISweepLin()



Example

```
ConfigPulseVMeasureISweepLin(smua, 0, 1, 10, 1, 10e-3, 20e-3, 16, smua.nvbuffer1, 4)
```

Set up a pulsed sweep that uses channel A. The pulsed sweep starts at 1 V, ends at 10 V, and returns to a 0 V bias level between pulses. Each pulsed step is on for 10 ms, and then at the bias level for 20 ms.

The current limit is 1 A during the entire pulsed sweep. The pulse train is comprised of 16 pulsed steps, and the pulse train is assigned a *tag* index of 4.

Also see

[InitiatePulseTest\(\)](#) (on page 15-113)

[InitiatePulseTestDual\(\)](#) (on page 15-115)

[KIPulse factory script](#) (on page 13-20)

ConfigPulseVMeasureISweepLog()

This KIPulse factory script function configures a voltage pulse train with a current measurement at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag)
```

<i>f</i>	A Boolean flag; this flag is <code>true</code> when the pulse was successfully configured, <code>false</code> when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <code>false</code> , <i>msg</i> contains an error message; if it is <code>true</code> , <i>msg</i> contains a string indicating successful configuration
<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>bias</i>	Bias level in volts
<i>start</i>	Pulse sweep start level in volts
<i>stop</i>	Pulse sweep stop level in volts
<i>limit</i>	Current limit (for example, compliance) in amperes
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles
<i>buffer</i>	Reading buffer where pulsed measurements are stored; if this is <code>nil</code> when the function is called, no measurements are made when the pulse train is initiated
<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>sync_in</i>	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
<i>sync_out</i>	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of <i>ton</i>
<i>sync_in_timeout</i>	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
<i>sync_in_abort</i>	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; <code>true</code> or <code>false</code>

Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a logarithmic pulsed voltage sweep with a current measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse is *start* volts; the magnitude of the last pulse is *stop* volts. The magnitude of each pulse in between is LogStep_n volts larger than the previous pulse, where:

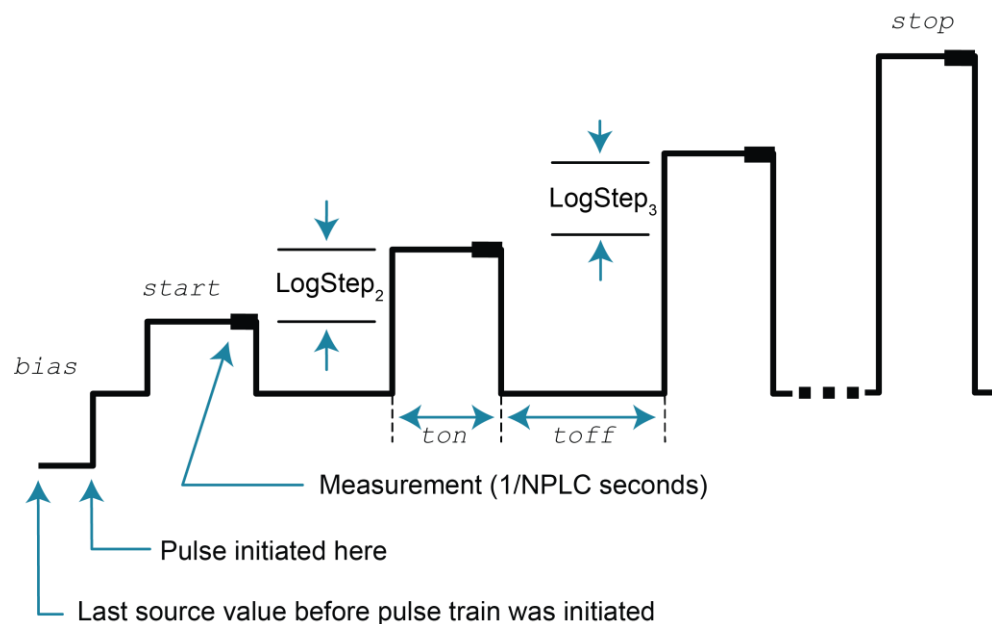
$$\text{LogStepSize} = (\log_{10}(\text{stop}) - \log_{10}(\text{start})) / (\text{points} - 1)$$

$$\text{LogStep}_n = (n - 1) * (\text{LogStepSize}), \text{ where } n = [2, \text{points}]$$

$$\text{SourceStepLevel}_n = \text{antilog}(\text{LogStep}_n) * \text{start}$$

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(tag)` and `InitiatePulseTestDual(tag1, tag2)` functions are used to initiate a pulse train assigned to a valid *tag*.

Figure 141: ConfigPulseVMeasureISweepLog()



Example

```
ConfigPulseVMeasureISweepLog(smua, 0, 1, 10, 1, 10e-3, 20e-3, 10, smua.nvbuffer1, 6)
```

Set up a pulsed logarithmic sweep that uses SMU channel A. The pulsed sweep starts at 1 V, ends at 10 V, and returns to a 0 V bias level between pulses. Each pulsed step is on for 10 ms, and then at the bias level for 20 ms.

The current limit is 1 A during the entire pulsed sweep. The pulse train is comprised of 10 pulsed steps, and the pulse train is assigned a *tag* index of 6.

Also see

[InitiatePulseTest\(\)](#) (on page 15-113)

[InitiatePulseTestDual\(\)](#) (on page 15-115)

[KIPulse factory script](#) (on page 13-20)

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)
```

<i>result</i>	The resulting value of <code>true</code> or <code>false</code> based on the success of the function
<i>value</i>	The data item to add; <i>value</i> can be of any type
<i>timeout</i>	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 15-50)

[dataqueue.clear\(\)](#) (on page 15-50)

[dataqueue.count](#) (on page 15-51)

[dataqueue.next\(\)](#) (on page 15-52)

[Using the data queue for real-time communication](#) (on page 14-63)

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
```

<code>count</code>	The variable that is assigned the value of <code>dataqueue.CAPACITY</code>

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 15-49)

[dataqueue.clear\(\)](#) (on page 15-50)

[dataqueue.count](#) (on page 15-51)

[dataqueue.next\(\)](#) (on page 15-52)

[Using the data queue for real-time communication](#) (on page 14-63)

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 15-49)

[dataqueue.CAPACITY](#) (on page 15-50)

[dataqueue.count](#) (on page 15-51)

[dataqueue.next\(\)](#) (on page 15-52)

[Using the data queue for real-time communication](#) (on page 14-63)

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 15-49)

[dataqueue.CAPACITY](#) (on page 15-50)

[dataqueue.clear\(\)](#) (on page 15-50)

[dataqueue.next\(\)](#) (on page 15-52)

[Using the data queue for real-time communication](#) (on page 14-63)

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)
```

<i>value</i>	The next entry in the data queue
<i>timeout</i>	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 15-49)
[dataqueue.CAPACITY](#) (on page 15-50)
[dataqueue.clear\(\)](#) (on page 15-50)
[dataqueue.count](#) (on page 15-51)
[format.asciiprecision](#) (on page 15-100)
[Using the data queue for real-time communication](#) (on page 14-63)

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)`

<i>seconds</i>	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 μ s to 10 μ s (typical) more than the requested delay.

Example

```
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.

Also see

None

digio.readbit()

This function reads one digital I/O line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readbit(N)
```

<i>data</i>	The state of the I/O line
<i>N</i>	Digital I/O line number to be read (1 to 14)

Details

A returned value of zero (0) indicates that the line is low. A returned value of one (1) indicates that the line is high.

Example

```
print(digio.readbit(4))
```

Assume line 4 is set high, and it is then read.

Output:

```
1.000000e+00
```

Also see

[digio.readport\(\)](#) (on page 15-54)

[digio.writebit\(\)](#) (on page 15-63)

[digio.writeport\(\)](#) (on page 15-63)

[Digital I/O port](#) (on page 7-37)

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()
```

<i>data</i>	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 I/O port. The least significant bit (bit B1) of the binary number corresponds to line 1; bit B14 corresponds to line 14.

For example, a returned value of 170 has a binary equivalent of 000000010101010, which indicates that lines 2, 4, 6, and 8 are high (1), and the other 10 lines are low (0).

Example

```
data = digio.readport()
print(data)
```

Assume lines 2, 4, 6, and 8 are set high when the I/O port is read.

Output:

```
1.700000e+02
```

This is binary 10101010.

Also see

[digio.readbit\(\)](#) (on page 15-54)

[digio.writebit\(\)](#) (on page 15-63)

[digio.writeport\(\)](#) (on page 15-63)

[Digital I/O port](#) (on page 7-37)

digio.trigger[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

```
digio.trigger[N].assert()
```

N

Digital I/O trigger line (1 to 14)

Details

The pulse width that is set determines how long the instrument asserts the trigger.

Example

```
digio.trigger[2].assert()
```

Asserts a trigger on digital I/O line 2.

Also see

[digio.trigger\[N\].pulsewidth](#) (on page 15-59)

digio.trigger[N].clear()

This function clears the trigger event on a digital I/O line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].clear()
```

<i>N</i>	Digital I/O trigger line (1 to 14)

Details

The event detector of a trigger enters the detected state when an event is detected. It is cleared when `digio.trigger[N].wait()` or `digio.trigger[N].clear()` is called.

`digio.trigger[N].clear()` clears the event detector of the specified trigger line, discards the history of the trigger line, and clears the `digio.trigger[N].overrun` attribute.

Example

<code>digio.trigger[2].clear()</code>	Clears the trigger event detector on I/O line 2.

Also see

[digio.trigger\[N\].overrun](#) (on page 15-58)

[digio.trigger\[N\].wait\(\)](#) (on page 15-62)

digio.trigger[N].EVENT_ID

This constant identifies the trigger event generated by the digital I/O line *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = digio.trigger[N].EVENT_ID
```

<i>eventID</i>	The trigger event number
<i>N</i>	Digital I/O trigger line (1 to 14)

Details

To have another trigger object respond to trigger events generated by the trigger line, set the stimulus attribute of the other object to the value of this constant.

Example

<code>digio.trigger[5].stimulus = digio.trigger[3].EVENT_ID</code>
Uses a trigger event on digital I/O trigger line 3 to be the stimulus for digital I/O line 5.

Also see

None

digio.trigger[N].mode

This attribute sets the mode in 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	Instrument reset Digital I/O trigger <i>N</i> reset Recall setup	Not saved	0 (digio.TRIG_BYPASS)

Usage

```
triggerMode = digio.trigger[N].mode
digio.trigger[N].mode = triggerMode
```

<i>triggerMode</i>	The trigger mode; see Details for values
<i>N</i>	Digital I/O trigger line (1 to 14)

Details

Set *triggerMode* to one of the following values:

Trigger mode values

<i>triggerMode</i>	Description
digio.TRIG_BYPASS or 0	Allows direct control of the line.
digio.TRIG_FALLING or 1	Detects falling-edge triggers as input; asserts a TTL-low pulse for output.
digio.TRIG_RISING or 2	If the programmed state of the line is high, the digio.TRIG_RISING mode behavior is similar to digio.TRIG_RISINGA. If the programmed state of the line is low, the digio.TRIG_RISING mode behavior is similar to digio.TRIG_RISINGM. Only use this setting if necessary for compatibility with other Keithley Instruments products.
digio.TRIG_EITHER or 3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
digio.TRIG_SYNCHRONOUSA or 4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line.
digio.TRIG_SYNCHRONOUS or 5	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
digio.TRIG_SYNCHRONOUSM or 6	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
digio.TRIG_RISINGA or 7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
digio.TRIG_RISINGM or 8	Asserts a TTL-high pulse for output. Input edge detection is not possible in this mode.

When programmed to any mode except digio.TRIG_BYPASS, the output state of the I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Use of either digio.TRIG_SYNCHRONOUSA or digio.TRIG_SYNCHRONOUSM is preferred over digio.TRIG_SYNCHRONOUS, because digio.TRIG_SYNCHRONOUS is provided for compatibility with the digital I/O and TSP-Link triggering on older firmware.

To control the line state, set the mode to `digio.TRIG_BYPASS` and use the `digio.writebit()` and `digio.writeport()` commands.

Example

```
digio.trigger[4].mode = 2
Sets the trigger mode for I/O line 4 to digio.TRIG_RISING.
```

Also see

[digio.trigger\[N\].clear\(\)](#) (on page 15-56)

[digio.trigger\[N\].reset\(\)](#) (on page 15-60)

[digio.writebit\(\)](#) (on page 15-63)

[digio.writeport\(\)](#) (on page 15-63)

[Sweep operation](#) (on page 5-34)

digio.trigger[N].overrun

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Digital I/O trigger <i>N</i> clear Digital I/O trigger <i>N</i> reset Recall setup	Not saved	Not applicable

Usage

```
overrun = digio.trigger[N].overrun
```

<i>overrun</i>	Trigger overrun state (<code>true</code> or <code>false</code>)
<i>N</i>	Digital I/O trigger line (1 to 14)

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 = digio.trigger[1].overrun
print(overrun)
If there is no trigger overrun, the following text is output:
false
```

Also see

[digio.trigger\[N\].clear\(\)](#) (on page 15-56)

[digio.trigger\[N\].reset\(\)](#) (on page 15-60)

digio.trigger[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	Instrument reset Digital I/O trigger <i>N</i> reset Recall setup	Not saved	10e-6 (10 µs)

Usage

```
width = digio.trigger[N].pulsewidth
digio.trigger[N].pulsewidth = width
```

<i>width</i>	The pulse width (seconds)
<i>N</i>	Digital I/O trigger line (1 to 14)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `digio.trigger[N].release()`.

Example

```
digio.trigger[4].pulsewidth = 20e-6
Sets the pulse width for trigger line 4 to 20 µs.
```

Also see

[digio.trigger\[N\].assert\(\)](#) (on page 15-55)
[digio.trigger\[N\].release\(\)](#) (on page 15-59)
[digio.trigger\[N\].reset\(\)](#) (on page 15-60)

digio.trigger[N].release()

This function releases an indefinite length or latched trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].release()
```

<i>N</i>	Digital I/O trigger line (1 to 14)

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.trigger[4].release()
```

Releases digital I/O trigger line 4.

Also see

[digio.trigger\[N\].assert\(\)](#) (on page 15-55)

[digio.trigger\[N\].pulsewidth](#) (on page 15-59)

digio.trigger[N].reset()

This function resets trigger values to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].reset()
```

<i>N</i>	Digital I/O trigger line (1 to 14)

Details

This function resets the following attributes to factory default settings:

- `digio.trigger[N].mode`
- `digio.trigger[N].pulsewidth`
- `digio.trigger[N].stimulus`

It also clears `digio.trigger[N].overrun`.

Example

```
digio.trigger[3].mode = 2
digio.trigger[3].pulsewidth = 50e-6
digio.trigger[3].stimulus = digio.trigger[5].EVENT_ID
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth,
      digio.trigger[3].stimulus)
digio.trigger[3].reset()
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth,
      digio.trigger[3].stimulus)
```

Set the digital I/O trigger line 3 for a falling edge with a pulsewidth of 50 μ s.

Use digital I/O line 5 to trigger the event on line 3.

Reset the line back to factory default values.

Output before reset:

```
2.00000e+00      5.00000e-05      5.00000e+00
```

Output after reset:

```
0.00000e+00      1.00000e-05      0.00000e+00
```

Also see

[digio.trigger\[N\].mode](#) (on page 15-57)

[digio.trigger\[N\].overrun](#) (on page 15-58)

[digio.trigger\[N\].pulsewidth](#) (on page 15-59)

[digio.trigger\[N\].stimulus](#) (on page 15-61)

digio.trigger[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	Instrument reset Digital I/O trigger N reset Recall setup	Not saved	0

Usage

```
triggerStimulus = digio.trigger[N].stimulus
digio.trigger[N].stimulus = triggerStimulus
```

<i>triggerStimulus</i>	The event identifier for the triggering event
<i>N</i>	Digital I/O trigger line (1 to 14)

Details

Set this attribute to zero (0) to disable the automatic trigger output.

Do not use the stimulus attribute for generating output triggers under script control. Use `digio.trigger[N].assert()` instead.

The trigger stimulus for a digital I/O line may be set to one of the existing trigger event IDs, described in the following table.

Trigger event IDs*	
Event ID	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measurement action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed

* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).

Example 1

```
digio.trigger[3].stimulus = 0
```

Clear the trigger stimulus of digital I/O line 3.

Example 2

```
digio.trigger[3].stimulus =
    smua.trigger.SOURCE_COMPLETE_EVENT_ID
```

Set the trigger stimulus of digital I/O line 3 to be the source complete event.

Also see

[digio.trigger\[N\].assert\(\)](#) (on page 15-55)

[digio.trigger\[N\].clear\(\)](#) (on page 15-56)

[digio.trigger\[N\].reset\(\)](#) (on page 15-60)

digio.trigger[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = digio.trigger[N].wait(timeout)
```

<i>triggered</i>	The value is <code>true</code> if a trigger is detected, or <code>false</code> if no triggers are detected during the timeout period
<i>N</i>	Digital I/O trigger line (1 to 14)
<i>timeout</i>	Timeout in seconds

Details

This function pauses trigger operation up to the seconds set by *timeout* for an input trigger. If one or more trigger events are detected since the last time `digio.trigger[N].wait()` or `digio.trigger[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 ready to detect the next trigger. This is true regardless of the number of events detected.

Example

```
triggered = digio.trigger[4].wait(3)
print(triggered)
```

Waits up to three seconds 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.trigger\[N\].clear\(\)](#) (on page 15-56)

digio.writebit()

This function sets a digital I/O line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writebit(N, data)
```

<i>N</i>	Digital I/O trigger line (1 to 14)
<i>data</i>	The value to write to the bit: 0 (low) - Non-zero (high)

Details

If the output line is write-protected using the `digio.writeprotect` attribute, the command is ignored.

The `reset()` function does not affect the present state of the digital I/O lines.

Use the `digio.writebit()` and `digio.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `digio.TRIG_BYPASS`.

The data must be zero (0) to clear the bit. Any value other than zero (0) sets the bit.

Example

```
digio.writebit(4, 0)    Sets digital I/O line 4 low (0).
```

Also see

[digio.readbit\(\)](#) (on page 15-54)
[digio.readport\(\)](#) (on page 15-54)
[digio.trigger\[N\].mode](#) (on page 15-57)
[digio.writeport\(\)](#) (on page 15-63)
[digio.writeprotect](#) (on page 15-64)

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)
```

<i>data</i>	Value to write to the port (0 to 16383)

Details

The binary representation of *data* indicates the output pattern to be written to the I/O port. For example, a *data* value of 170 has a binary equivalent of 00000010101010. Lines 2, 4, 6, and 8 are set high (1), and the other 10 lines are set low (0).

Write-protected lines are not changed.

The `reset()` function does not affect the present states of the digital I/O lines.

Use the `digio.writebit()` and `digio.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `digio.TRIG_BYPASS`.

Example

```
digio.writeport(255)
```

Sets digital I/O Lines 1 through 8 high (binary 00000011111111).

Also see

[digio.readbit\(\)](#) (on page 15-54)

[digio.readport\(\)](#) (on page 15-54)

[digio.writebit\(\)](#) (on page 15-63)

[digio.writeprotect](#) (on page 15-64)

digio.writeprotect

This attribute contains the write-protect mask that protects bits from changes from the `digio.writebit()` and `digio.writeport()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0

Usage

```
mask = digio.writeprotect
digio.writeprotect = mask
```

```
mask
```

Sets the value that specifies the bit pattern for write-protect

Details

Bits that are set to one cause the corresponding line to be write-protected.

The binary equivalent of *mask* indicates the mask to be set for the I/O port. For example, a mask value of 7 has a binary equivalent of 00000000000111. This mask write-protects lines 1, 2, and 3.

Example

```
digio.writeprotect = 15
```

Write-protects lines 1, 2, 3, and 4.

Also see

[digio.writebit\(\)](#) (on page 15-63)

[digio.writeport\(\)](#) (on page 15-63)

display.clear()

This function clears all lines of the virtual front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.clear()
```

Details

This function switches to the user screen and then clears the virtual front-panel display.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the virtualdisplay. For performance considerations, they update the display as soon as processing time becomes available.

Also see

[display.setcursor\(\)](#) (on page 15-79)

[display.settext\(\)](#) (on page 15-80)

display.getannunciators()

This function reads the annunciators (indicators) that are presently turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
annunciators = display.getannunciators()
```

<code>annunciators</code>	The bitmasked value that shows which indicators are turned on

Details

This function returns a bitmasked value showing which indicators are turned on. The 16-bit binary equivalent of the returned value is the bitmask. The return value is a sum of set annunciators, based on the weighted value, as shown in the following table.

Annunciator (indicator) bitmasked values and equivalent constants			
Indicator	Bit	Weighted value	Equivalent constant
FILT	1	1	display.ANNUNCIATOR_FILTER
MATH	2	2	display.ANNUNCIATOR_MATH
4W	3	4	display.ANNUNCIATOR_4_WIRE
AUTO	4	8	display.ANNUNCIATOR_AUTO
ARM	5	16	display.ANNUNCIATOR_ARM
TRIG	6	32	display.ANNUNCIATOR_TRIGGER
*(asterisk)	7	64	display.ANNUNCIATOR_STAR
SMPL	8	128	display.ANNUNCIATOR_SAMPLE
EDIT	9	256	display.ANNUNCIATOR_EDIT
ERR	10	512	display.ANNUNCIATOR_ERROR
REM	11	1024	display.ANNUNCIATOR_REMOTE
TALK	12	2048	display.ANNUNCIATOR_TALK
LSTN	13	4096	display.ANNUNCIATOR_LISTEN
SRQ	14	8192	display.ANNUNCIATOR_SRQ
REAR	15	16384	display.ANNUNCIATOR_REAR
REL	16	32768	display.ANNUNCIATOR_REL

Example 1

```
testAnnunciators = display.getannunciators()
print(testAnnunciators)

rem = bit.bitand(testAnnunciators, 1024)
if rem > 0 then
    print("REM is on")
else
    print("REM is off")
end
```

REM indicator is turned on.

Output:

1.28000e+03

REM is on

Example 2

```
print(display.ANNUNCIATOR_EDIT)
print(display.ANNUNCIATOR_TRIGGER)
print(display.ANNUNCIATOR_AUTO)
```

Output:

2.56000e+02

3.20000e+01

8.00000e+00

Also see

[bit.bitand\(\)](#) (on page 15-8)

display.getcursor()

This function reads the present position of the cursor on the virtual front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
row, column, style = display.getcursor()
```

<i>row</i>	The row where the cursor is: 1 (top row); 2 (bottom row)
<i>column</i>	The column where the cursor is: ■ If the cursor is in the top row: 1 to 20 ■ If the cursor is in the bottom row: 1 to 32
<i>style</i>	Visibility of the cursor: ■ Invisible: 0 ■ Blinking: 1

Details

This function switches the virtual front-panel display to the user screen (the text set by `display.settext()`), and then returns values to indicate the row that contains the cursor and the column position and cursor style.

Columns are numbered from left to right on the display.

Example 1

```
testRow, testColumn = display.getcursor()
print(testRow, testColumn)
```

This example reads the cursor position into local variables and prints them.

Example output:

```
1.000000e+00    1.000000e+00
```

Example 2

```
print(display.getcursor())
```

This example prints the cursor position directly. In this example, the cursor is in row 1 at column 3, with an invisible cursor:

```
1.000000e+00    3.000000e+00    0.000000e+00
```

Also see

[display.gettext\(\)](#) (on page 15-69)

[display.screen](#) (on page 15-78)

[display.setcursor\(\)](#) (on page 15-79)

[display.settext\(\)](#) (on page 15-80)

display.getlastkey()

This function retrieves the key code for the last pressed key.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
keyCode = display.getlastkey()
```

<i>keyCode</i>	A returned value that represents the last virtual front-panel key pressed; see Details for more information

Details

A history of the key code for the last pressed virtual front-panel key is maintained by the instrument. When the instrument is turned on, or when it is transitioning from local to remote operation, the key code is set to 0 (`display.KEY_NONE`).

Pressing the EXIT (LOCAL) key normally aborts a script. To use this function with the EXIT (LOCAL) key, you must set `display.locallockout` to `display.LOCK`.

The table below lists the *keyCode* value for each virtual front-panel action.

Key codes

Value	Key list	Value	Key list
0	<code>display.KEY_NONE</code>	82	<code>display.KEY_ENTER</code>
65	<code>display.KEY_RANGEUP</code>	83	<code>display.KEY_MEASB</code>
67	<code>display.KEY_RELB</code>	84	<code>display.DIGITSB</code>
68	<code>display.KEY_MENU</code>	85	<code>display.KEY_RECALL</code>
69	<code>display.KEY_MODEA</code>	86	<code>display.KEY_MEASA</code>
70	<code>display.KEY_RELA</code>	87	<code>display.KEY_DIGITSA</code>
71	<code>display.KEY_RUN</code>	90	<code>display.KEY_LIMITB</code>
72	<code>display.KEY_DISPLAY</code>	91	<code>display.KEY_SPEEDB</code>
73	<code>display.KEY_AUTO</code>	92	<code>display.KEY_TRIG</code>
74	<code>display.KEY_FILTERB</code>	93	<code>display.KEY_LIMITA</code>
75	<code>display.KEY_EXIT</code>	94	<code>display.KEY_SPEEDA</code>
76	<code>display.KEY_SRCB</code>	95	<code>display.KEY_LOAD</code>
77	<code>display.KEY_FILTERA</code>	97	<code>display.WHEEL_ENTER</code>
78	<code>display.KEY_STORE</code>	103	<code>display.KEY_RIGHT</code>
79	<code>display.KEY_SRCA</code>	104	<code>display.KEY_LEFT</code>
80	<code>display.KEY_CONFIG</code>	107	<code>display.WHEEL_LEFT</code>
81	<code>display.KEY_RANGEDOWN</code>	114	<code>display.WHEEL_RIGHT</code>

NOTE

When using this function, use built-in constants such as `display.KEY_RIGHT` (rather than the numeric value of 103). This allows for better forward compatibility with firmware revisions.

You cannot use this function to track the OUTPUT ON/OFF controls for SMU A or SMU B.

Example

```
key = display.getlastkey()
print(key)
```

On the virtual front panel, press the **MENU** key and then send the code shown here. This retrieves the key code for the last pressed key.

Output:

```
6.80000e+01
```

Also see

[display.locallockout](#) (on page 15-74)

[display.sendkey\(\)](#) (on page 15-78)

display.gettext()

This function reads the text displayed on the virtual front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, columnStart)
text = display.gettext(embellished, row, columnStart, columnEnd)
```

<i>text</i>	The returned value, which contains the text that is presently displayed
<i>embellished</i>	Indicates type of returned text: <i>false</i> (simple text); <i>true</i> (text with embedded character codes)
<i>row</i>	Selects the row from which to read the text: 1 (row 1); 2 (row 2). If <i>row</i> is not included, both rows of text are read
<i>columnStart</i>	Selects the first column from which to read text; for row 1, the valid column numbers are 1 to 20; for row 2, the valid column numbers are 1 to 32; if nothing is selected, 1 is used
<i>columnEnd</i>	Selects the last column from which to read text; for row 1, the valid column numbers are 1 to 20; for row 2, the valid column numbers are 1 to 32; the default is 20 for row 1, and 32 for row 2

Details

Using the command without any parameters returns both lines of the virtual front-panel display.

The `$N` character code is included in the returned value to show where the top line ends and the bottom line begins. This is not affected by the value of *embellished*.

When *embellished* is set to *true*, all other character codes are returned along with the message. When *embellished* is set to *false*, only the message and the `$N` character code is returned. For information on the embedded character codes, see [display.settext\(\)](#) (on page 15-80).

The display is not switched to the user screen (the screen set using `display.settext()`). Text is read from the active screen.

Example 1

```
display.clear()
display.setcursor(1, 1)
display.settext("ABCDEFGH IJ$DKLMNOPQRST")
display.setcursor(2, 1)
display.settext("abcdefghijk l m$Bnopqrstuvwxyz$F123456")
print(display.gettext())
print(display.gettext(true))
print(display.gettext(false, 2))
print(display.gettext(true, 2, 9))
print(display.gettext(false, 2, 9, 10))
```

This example shows how to retrieve the display text in multiple ways. The output is:

```
ABCDEFGH IJ KLMNOPQRST$Nabcdefghijk l m n o p q r s t u v w x y z 1 2 3 4 5 6
$RABCDEFGH IJ$DKLMNOPQRST$N$Rabcdefghijk l m$Bnopqrstuvwxyz$F123456
abcdefghijk l m n o p q r s t u v w x y z 1 2 3 4 5 6
$Rijk l m$Bnopqrstuvwxyz$F123456
ij
```

Example 2

```
display.clear()
display.settext("User Screen")
text = display.gettext()
print(text)
```

This outputs all text in both lines of the display:

```
User Screen      $N
```

This indicates that the message "User Screen" is on the top line. The bottom line is blank.

Also see

[display.clear\(\)](#) (on page 15-65)

[display.getcursor\(\)](#) (on page 15-67)

[display.setcursor\(\)](#) (on page 15-79)

[display.settext\(\)](#) (on page 15-80)

display.inputvalue()

This function displays a formatted input field on the virtual front-panel display that the operator can edit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.inputvalue("format")
display.inputvalue("format", default)
display.inputvalue("format", default, minimum)
display.inputvalue("format", default, minimum, maximum)
```

<i>format</i>	A string that defines how the input field is formatted; see Details for more information
<i>default</i>	The default value for the input value
<i>minimum</i>	The minimum input value
<i>maximum</i>	The maximum input value

Details

The *format* parameter uses zeros (0), the decimal point, polarity sign, and exponents to define how the input field is formatted. The *format* parameter can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially	0.00000e+0
+	Allows operators to enter positive or negative values; if the "+" sign is not included, the operator cannot enter a negative value	+0.00
0	Defines the digit positions for the value; up to six zeros (0)	+00.0000e+00
.	Include to have a decimal point appear in the value	+0.00

The *default* parameter is the value shown when the value is first displayed.

You can use the *minimum* and *maximum* parameters to limit the values that can be entered. When + is not selected for *format*, the minimum limit must be more than or equal to zero (0). When limits are used, you cannot enter values above or below these limits.

The input value is limited to $\pm 1e37$.

Before calling `display.inputvalue()`, you should send a message prompt to the operator using `display.prompt()`. Make sure to position the cursor where the edit field should appear.

After this command is sent, script execution pauses until you enter a value and press the **ENTER** key.

For positive and negative entry (plus sign (+) used for the value field and the exponent field), polarity of a nonzero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity is also toggled when using the navigation wheel to decrease or increase the value or exponent past zero. A zero (0) value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After executing this command and pressing the EXIT (LOCAL) key, the function returns `nil`.

Example

```
display.clear()
display.settext("Enter value between$N -0.10 and 2.00: ")
value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)
print("Value entered = ", value)
```

Displays an editable field (+0.50) for operator input. The valid input range is -0.10 to +2.00, with a default of 0.50.

Output:

```
Value entered = 1.35000e+00
```

Also see

[display.prompt\(\)](#) (on page 15-76)

[display.setcursor\(\)](#) (on page 15-79)

[display.settext\(\)](#) (on page 15-80)

display.loadmenu.add()

This function adds an entry to the USER menu, which can be accessed by pressing the **LOAD** key on the virtual front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.loadmenu.add("displayName", "code")
display.loadmenu.add("displayName", "code", memory)
```

<i>displayName</i>	The name that is added to the USER menu
<i>code</i>	The code that is run from the USER menu
<i>memory</i>	Determines if code is saved to nonvolatile memory: 0 or <code>display.DONT_SAVE</code>: Does not save the code to nonvolatile memory 1 or <code>display.SAVE</code>: Saves the code to nonvolatile memory (default)

Details

After adding code to the load menu, you can run it from the virtual front panel by pressing the **LOAD** key, then selecting **USER** to select from the available code to load. Pressing the **RUN** key then runs the script.

You can add items in any order. They are always displayed in alphabetical order when the menu is selected.

Any Lua code can be included in the *code* parameter. If *memory* is set to `display.SAVE`, the entry (name and code) is saved in nonvolatile memory. Scripts, functions, and variables used in the code are not saved by `display.SAVE`. Functions and variables need to be saved with the code. If the code is not saved in nonvolatile memory, it is lost when the 2606B is turned off. See **Example 2** below.

If you do not make a selection for *memory*, the code is automatically saved to nonvolatile memory.

NOTE

You can create a script that defines several functions, and then use the `display.loadmenu.add()` command to add items that call those individual functions. This allows the operator to run tests from the virtual front panel.

Example 1

```
display.loadmenu.add("Test9", "Test9()")
```

Assume a user script named `Test9` is loaded into the runtime environment. Adds the menu entry to the USER menu to run the script after loading.

Example 2

```
display.loadmenu.add("Test", "DUT1() beeper.beep(2, 500)", display.SAVE)
```

Assume a script with a function named “DUT1” is loaded into the instrument, and the script has not been saved in nonvolatile memory.

Now assume you want to add a test named “Test” to the USER menu. You want the test to run the function named DUT1 and sound the beeper. This example adds Test to the menu, defines the code, and then saves the displayName and code in nonvolatile memory.

When Test is run from the virtual front panel USER menu, the function named DUT1 executes and the beeper beeps for two seconds.

Now assume you turn off instrument power. Because the script was not saved in nonvolatile memory, the function named DUT1 is lost when you turn the instrument on. When Test is run again from the virtual front panel, an error is generated because DUT1 no longer exists in the instrument as a function.

Example 3

```
display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)", display.SAVE)
```

Adds an entry called Part1 to the virtual front-panel USER load menu for the code testpart([[Part1]], 5.0) and saves it in nonvolatile memory.

Also see

[display.loadmenu.delete\(\)](#) (on page 15-74)

display.loadmenu.catalog()

This function creates an iterator for the user menu items accessed using the LOAD key on the virtual front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for displayName in display.loadmenu.catalog() do body end
for displayName, code in display.loadmenu.catalog() do body end
```

<i>displayName</i>	The name displayed in the menu
<i>code</i>	The code associated with the <i>displayName</i>
<i>body</i>	The body of the code to process the entries in the loop

Details

Each time through the loop, *displayName* and *code* take on the values in the USER menu.

The instrument goes through the list in random order.

Example

```
for displayName, code in display.loadmenu.catalog() do
  print(displayName, code)
end
```

Output:

```
Test DUT1() beeper.beep(2, 500)
Part1 testpart([[Part1]], 5.0)
Test9 Test9()
```

Also see

[display.loadmenu.add\(\)](#) (on page 15-72)

[display.loadmenu.delete\(\)](#) (on page 15-74)

display.loadmenu.delete()

This function removes an entry from the USER menu, which can be accessed using the **LOAD** key on the virtual front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.loadmenu.delete("displayName")
```

<i>displayName</i>	The name to be deleted from the USER menu

Details

If you delete an entry from the USER menu, you can no longer run it by pressing the **LOAD** key.

Example

```
display.loadmenu.delete("Test9")
for displayName, code in display.loadmenu.catalog() do
    print(displayName, code)
end
```

Deletes the entry named Test9.

Output:

```
Test    DUT1() beeper.beep(2, 500)
Part1   testpart([[Part1]], 5.0)
```

Also see

[display.loadmenu.add\(\)](#) (on page 15-72)

[display.loadmenu.catalog\(\)](#) (on page 15-73)

display.locallockout

This attribute describes whether or not the EXIT (LOCAL) key on the instrument virtual front panel is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (display.UNLOCK)

Usage

```
lockout = display.locallockout
display.locallockout = lockout
```

<i>lockout</i>	0 or display.UNLOCK: Unlocks EXIT (LOCAL) key 1 or display.LOCK: Locks out EXIT (LOCAL) key

Details

Set `display.locallockout` to `display.LOCK` to prevent the user from interrupting remote operation by pressing the EXIT (LOCAL) key.

Set this attribute to `display.UNLOCK` to allow the EXIT (LOCAL) key to interrupt script or remote operation.

Example

```
display.locallockout = display.LOCK
```

Disables the virtual front-panel EXIT (LOCAL) key.

Also see

None

display.menu()

This function presents a menu on the virtual front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
selection = display.menu("name", "items")
```

<i>selection</i>	Name of the variable that holds the selected menu item
<i>name</i>	Menu name to display on the top line
<i>items</i>	Menu items to display on the bottom line

Details

The menu consists of the menu name string on the top line, and a selectable list of items on the bottom line. The menu items must be a single string with each item separated by whitespace. The name for the top line is limited to 20 characters.

After sending this command, script execution pauses for the operator to select a menu item. An item is selected by rotating the navigation wheel to place the blinking cursor on the item, and then pressing the navigation wheel (or the ENTER key). When an item is selected, the text of that selection is returned.

Pressing the EXIT (LOCAL) key does not abort the script while the menu is displayed, but it does return `nil`. The script can be aborted by calling the exit function when `nil` is returned.

Example

```
selection = display.menu("Menu", "Test1 Test2 Test3")
print(selection)
```

Displays a menu with three menu items. If the second menu item is selected, selection is given the value Test2.

Output:
Test2

Also see

None

display.numpad

This attribute controls whether the virtual front panel keys act as a numeric keypad during value entry.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	1 (display.ENABLE)

Usage

```
numericKeypad = display.numpad
display.numpad = numericKeypad
```

<i>numericKeypad</i>	Enable the numeric keypad feature (1 or <code>display.ENABLE</code>) Disable the numeric keypad feature (0 or <code>display.DISABLE</code>)

Details

The numeric keypad feature is only available when editing a numeric value at the same time that the EDIT indicator is lit.

Example

<code>display.numpad = display.ENABLE</code>	Turn on the numeric keypad feature.

Also see

[Setting a value](#) (on page 3-32)

display.prompt()

This function prompts the user to enter a parameter from the virtual front panel of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.prompt("format", "units", "help")
display.prompt("format", "units", "help", default)
display.prompt("format", "units", "help", default, minimum)
display.prompt("format", "units", "help", default, minimum, maximum)
```

<i>format</i>	A string that defines how the input field is formatted; see Details for more information
<i>units</i>	Set the units text string for the top line (eight characters maximum); this indicates the units (for example, "V" or "A") for the value
<i>help</i>	Text string to display on the bottom line (32 characters maximum)
<i>default</i>	The value that is shown when the value is first displayed
<i>minimum</i>	The minimum input value that can be entered
<i>maximum</i>	The maximum input value that can be entered (must be more than minimum)

Details

This function creates an editable input field at the present cursor position, and an input prompt message on the bottom line. Example of a displayed input field and prompt:

```
0.00V
```

```
Input 0 to +2V
```

The *format* parameter uses zeros (0), the decimal point, polarity sign, and exponents to define how the input field is formatted.

The *format* parameter can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially. Include a plus sign (+) for positive/negative exponent entry. Do not include the plus sign (+) to prevent negative value entry. 0 defines the digit positions for the exponent.	0.00000E+0
+	Allows operators to enter positive or negative values. If the plus sign (+) is not included, the operator cannot enter a negative value.	+0.00
0	Defines the digit positions for the value. You can use up to six zeros (0).	+00.00000E+00
.	The decimal point where needed for the value.	+0.00

You can use the *minimum* and *maximum* parameters to limit the values that can be entered. When a plus sign (+) is not selected for *format*, the minimum limit must be greater than or equal to zero (0). When limits are used, the operator cannot enter values above or below these limits.

The input value is limited to $\pm 1e37$.

After sending this command, script execution pauses for the operator to enter a value and press **ENTER**.

For positive and negative entry (plus sign (+) used for the value field and the exponent field), polarity of a nonzero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity also toggles when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After executing this command and pressing the **EXIT (LOCAL)** key, the value returns *nil*.

Example

```
value = display.prompt("0.00", "V", "Input 0 to +2V", 0.5, 0, 2)
print(value)
```

The above command prompts the operator to enter a voltage value. The valid input range is 0 to +2.00, with a default of 0.50:

```
0.50V
```

```
Input 0 to +2V
```

If the operator enters 0.70, the output is:

```
7.00000e-01
```

Also see

[display.inputvalue\(\)](#) (on page 15-70)

display.screen

This attribute contains the selected display screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	2 (display.SMUA_SMUB)

Usage

```
displayID = display.screen  
display.screen = displayID
```

displayID

One of the following values:

- 0 or `display.SMUA`: Displays source-measure and compliance for SMU A
- 1 or `display.SMUB`: Displays source-measure and compliance for SMU B
- 2 or `display.SMUA_SMUB`: Displays source-measure for SMU A and SMU B
- 3 or `display.USER`: Displays the user screen

Details

Setting this attribute selects the display screen for the virtual front panel. This performs the same action as selecting the DISPLAY key on the virtual front panel. The text for the display screen is set by `display.settext()`.

Read this attribute to determine which of the available display screens was last selected.

Example

```
display.screen = display.SMUA
```

Selects the source-measure and compliance limit display for SMU A.

Also see

[display.settext\(\)](#) (on page 15-80)

display.sendkey()

This function sends a code that simulates the action of a virtual front-panel control.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.sendkey(keyCode)
```

keyCode

A parameter that specifies the key press to simulate; see **Details** for more information

Details

This command simulates pressing a virtual front-panel key or navigation wheel, or turning the navigation wheel one click to the left or right.

Key codes			
Value	Key list	Value	Key list
65	display.KEY_RANGEUP	83	display.KEY_MEASB
66	display.KEY_MODEB	84	display.KEY_DIGITSB
67	display.KEY_RELB	85	display.KEY_RECALL
68	display.KEY_MENU	86	display.KEY_MEASA
69	display.KEY_MODEA	87	display.KEY_DIGITSA
70	display.KEY_RELA	88	display.KEY_OUTPUTA
71	display.KEY_RUN	90	display.KEY_LIMITB
72	display.KEY_DISPLAY	91	display.KEY_SPEEDB
73	display.KEY_AUTO	92	display.KEY_TRIG
74	display.KEY_FILTERB	93	display.KEY_LIMITA
75	display.KEY_EXIT	94	display.KEY_SPEEDA
76	display.KEY_SRCB	95	display.KEY_LOAD
77	display.KEY_FILTERA	96	display.KEY_OUTPUTB
78	display.KEY_STORE	97	display.WHEEL_ENTER
79	display.KEY_SRCB	103	display.KEY_RIGHT
80	display.KEY_CONFIG	104	display.KEY_LEFT
81	display.KEY_RANGEDOWN	107	display.WHEEL_LEFT
82	display.KEY_ENTER	114	display.WHEEL_RIGHT

NOTE

When using this function, send built-in constants, such as `display.KEY_RIGHT`, rather than the numeric value, such as 103. This allows for better forward compatibility with firmware revisions.

Example

```
display.sendkey(display.KEY_RUN)
```

Simulates pressing the RUN key.

Also see

[Front panel](#) (on page 3-1)

display.setcursor()

This function sets the position of the cursor.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.setcursor(row, column)
display.setcursor(row, column, style)
```

<i>row</i>	The row number for the cursor (1 or 2)
<i>column</i>	The active column position to set; row 1 has columns 1 to 20, row 2 has columns 1 to 32
<i>style</i>	Set the cursor to invisible (0, default) or blinking (1)

Details

Sending this command selects the user screen and then moves the cursor to the given location.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the virtualdisplay. For performance considerations, they update the display as soon as processing time becomes available.

An out-of-range parameter for `row` sets the cursor to row 2. An out-of-range parameter for `column` sets the cursor to column 20 for row 1, or 32 for row 2.

An out-of-range parameter for `style` sets it to 0 (invisible).

A blinking cursor is only visible when it is positioned over displayed text. It cannot be seen when positioned over a space character.

Example

```
display.clear()
display.setcursor(1, 8)
display.settext("Hello")
display.setcursor(2, 14)
display.settext("World")
```

This example displays a message on the virtual front panel, approximately center. Note that the top line of text is larger than the bottom line of text.

The virtual front panel of the instrument displays `Hello` on the top line and `World` on the second line.

Also see

[display.clear\(\)](#) (on page 15-65)

[display.getcursor\(\)](#) (on page 15-67)

[display.gettext\(\)](#) (on page 15-69)

[display.screen](#) (on page 15-78)

[display.settext\(\)](#) (on page 15-80)

display.settext()

This function displays text on the virtual front-panel user screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.settext("text")
```

<code>text</code>	Text message to be displayed, with optional character codes

Details

This function selects the user display screen and displays the given text.

After the instrument is turned on, the first time you use a display command to write to the display, the message "User Screen" is cleared. After the first write, you need to use `display.clear()` to clear the message.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the virtualdisplay. For performance considerations, they update the display as soon as processing time becomes available.

The text starts at the present cursor position. After the text is displayed, the cursor is after the last character in the display message.

Top line text does not wrap to the bottom line of the display automatically. Any text that does not fit on the current line is truncated. If the text is truncated, the cursor remains at the end of the line.

The text remains on the display until replaced or cleared.

The character codes described in the following table can also be included in the text string.

Display character codes

Character Code	Description
\$N	Newline, starts text on the next line; if the cursor is already on line 2, text is ignored after the \$N is received
\$R	Sets text to normal intensity, nonblinking
\$B	Sets text to blink
\$D	Sets text to dim intensity
\$F	Sets the text to background blink
\$\$	Escape sequence to display a single dollar symbol (\$)

Example

```
display.clear()
display.settext("Normal $B$Blinking$N")
display.settext("$DDim $F$FBackgroundBlink$R $$$$ 2 dollars")
```

This example sets the display to:

```
Normal Blinking
Dim BackgroundBlink $$ 2 dollars
with the named effect on each word.
```

Also see

[display.clear\(\)](#) (on page 15-65)
[display.getcursor\(\)](#) (on page 15-67)
[display.gettext\(\)](#) (on page 15-69)
[display.screen](#) (on page 15-78)
[display.setcursor\(\)](#) (on page 15-79)

display.smuX.digits

This attribute sets the virtual front-panel display resolution of the selected measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	5 (display.DIGITS_5_5)

Usage

```
digits = display.smuX.digits  
display.smuX.digits = digits
```

<i>digits</i>	Set digits to one of the following values: ■ Select 4½ digit resolution (4 or <code>display.DIGITS_4_5</code>) ■ Select 5½ digit resolution (5 or <code>display.DIGITS_5_5</code>) ■ Select 6½ digit resolution (6 or <code>display.DIGITS_6_5</code>)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>display.smua.digits</code> applies to SMU channel A)

Details

SMU A and SMU B can be set for different measurement display resolutions.

Example

```
display.smua.digits = display.DIGITS_5_5
```

 Select 5½ digit resolution for SMU A.

Also see

None

display.smuX.limit.func

If you are using a display mode that shows a single channel, this attribute specifies the type of limit value setting displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0 (display.LIMIT_IV)

Usage

```
func = display.smuX.limit.func  
display.smuX.limit.func = func
```

<i>func</i>	One of the following values: ■ 0 or <code>display.LIMIT_IV</code>: Displays the primary limit setting ■ 1 or <code>display.LIMIT_P</code>: Displays the power limit setting
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>display.smua.limit.func</code> applies to SMU channel A)

Details

Selects the displayed limit function: primary (IV) or power (P).

SMU A and SMU B can be set for different display functions.

Example

```
display.smuA.limit.func = display.LIMIT_P
```

Specify the power limit value is displayed for SMU Channel A.

Also see

[display.smuX.measure.func](#) (on page 15-83)

[Display mode](#) (on page 3-34)

display.smuX.measure.func

This attribute specifies the type of measurement that is being displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	1 (display.MEASURE_DCVOLTS)

Usage

```
func = display.smuX.measure.func
display.smuX.measure.func = func
```

<i>func</i>	The type of measurement: ▪ 0 or display.MEASURE_DCAMPS: Current measurement function ▪ 1 or display.MEASURE_DCVOLTS: Voltage measurement function ▪ 2 or display.MEASURE_OHMS: Resistance measurement function ▪ 3 or display.MEASURE_WATTS: Power measurement function
<i>X</i>	Source-measure unit (SMU) channel (for example, display.smuA.measure.func applies to SMU channel A)

Details

Selects the measurement function that is displayed on the virtual front panel: Amps, volts, ohms, or watts.

SMU A and SMU B can be set for different measurement functions.

Example

```
display.smuA.measure.func = display.MEASURE_DCAMPS
```

Selects the current measure function for SMU A.

Also see

[display.smuX.limit.func](#) (on page 15-82)

display.trigger.clear()

This function clears the virtual front-panel trigger event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.trigger.clear()
```

Details

The trigger event detector remembers if an event has been detected since the last `display.trigger.wait()` call. This function clears the trigger event detector and discards the previous history of TRIG key presses.

This attribute also clears the `display.trigger.overrun` attribute.

Also see

[display.trigger.overrun](#) (on page 15-84)

[display.trigger.wait\(\)](#) (on page 15-85)

display.trigger.EVENT_ID

This constant is the event ID of the event generated when the virtual front-panel TRIG key is pressed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = display.trigger.EVENT_ID
```

<code>eventID</code>	The trigger event number

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to virtual front-panel trigger key events.

Also see

None

display.trigger.overrun

This attribute contains the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Display trigger clear Instrument reset Recall setup	Not saved	false

Usage

```
overrun = display.trigger.overrun
```

<code>overrun</code>	The trigger overrun state (true or false)

Details

Indicates if a trigger event was ignored because the event detector was already in the detected state when the TRIG button was pressed.

Indicates the overrun state of the event detector built into the display.

This attribute does not indicate whether an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = display.trigger.overrun
```

Sets the variable `overrun` equal to the present state of the event detector built into the display.

Also see

[display.trigger.clear\(\)](#) (on page 15-84)

display.trigger.wait()

This function waits for the TRIG key on the virtual front panel to be pressed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = display.trigger.wait(timeout)
```

<code>triggered</code>	true: Trigger was detected false: The operation timed out
<code>timeout</code>	Timeout in seconds

Details

If the trigger key was previously pressed and one or more trigger events were 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.

Use the `display.trigger.clear()` call to clear the trigger event detector.

Example

```
triggered = display.trigger.wait(5)
print(triggered)
```

Waits up to five seconds for the TRIG key to be pressed. If TRIG is pressed within five seconds, the output is `true`. If not, the output is `false`.

Also see

[display.trigger.clear\(\)](#) (on page 15-84)

display.waitkey()

This function captures the key code value for the next virtual front-panel action.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
keyCode = display.waitkey()
```

keyCode	See Details for more information

Details

After you send this function, script execution pauses until a virtual front-panel action (for example, pressing a key or the navigation wheel, or turning the navigation wheel). After the action, the value of the key (or action) is returned.

If the EXIT (LOCAL) key is pressed while this function is waiting for a virtual front-panel action, the script is not aborted.

A typical use for this function is to prompt the user to press the EXIT (LOCAL) key to abort the script or press any other key to continue. For example, if the *keyCode* value 75 is returned (the EXIT (LOCAL) key was pressed), you can call the `exit()` function to abort the script.

The table below lists the *keyCode* values for each front panel action.

Key codes			
Value	Key (or action)	Value	Key (or action)
65	display.KEY_RANGEUP	83	display.KEY_MEASB
66	display.KEY_MODEB	84	display.KEY_DIGITSB
67	display.KEY_RELB	85	display.KEY_RECALL
68	display.KEY_MENU	86	display.KEY_MEASA
69	display.KEY_MODEA	87	display.KEY_DIGITSA
70	display.KEY_RELA	88	display.KEY_OUTPUTA
71	display.KEY_RUN	90	display.KEY_LIMITB
72	display.KEY_DISPLAY	91	display.KEY_SPEEDB
73	display.KEY_AUTO	92	display.KEY_TRIG
74	display.KEY_FILTERB	93	display.KEY_LIMITA
75	display.KEY_EXIT	94	display.KEY_SPEEDA
76	display.KEY_SRCB	95	display.KEY_LOAD
77	display.KEY_FILTERA	96	display.KEY_OUTPUTB
78	display.KEY_STORE	97	display.WHEEL_ENTER
79	display.KEY_SRCB	103	display.KEY_RIGHT
80	display.KEY_CONFIG	104	display.KEY_LEFT
81	display.KEY_RANGEDOWN	107	display.WHEEL_LEFT
82	display.KEY_ENTER	114	display.WHEEL_RIGHT

NOTE

When using this function, use built-in constants such as `display.KEY_RIGHT` (rather than the numeric value of 103). This allows for better forward compatibility with firmware revisions.

Example

```
key = display.waitkey()
print(key)
```

Pause script execution until the operator presses a key or the navigation wheel, or rotates the navigation wheel.

If the output is:

```
8.60000e+01
```

It indicates that the MEAS(A) key was pressed.

Also see

[Capturing key-press codes](#) (on page 3-46)

[display.getlastkey\(\)](#) (on page 15-68)

[display.sendkey\(\)](#) (on page 15-78)

[display.settext\(\)](#) (on page 15-80)

errorqueue.clear()

This function clears all entries out of the error queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
errorqueue.clear()
```

Details

See [Error queue](#) (on page 10-3) for additional information about the error queue.

Also see

[errorqueue.count](#) (on page 15-88)

[errorqueue.next\(\)](#) (on page 15-88)

errorqueue.count

This attribute gets the number of entries in the error queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle Clearing error queue Reading error messages	Not applicable	Not applicable

Usage

```
count = errorqueue.count
```

count	The number of entries in the error queue

Example

```
count = errorqueue.count  
print(count)
```

Returns the number of entries in the error queue.
The output below indicates that there are four entries in the error queue:
4.00000e+00

Also see

[Error queue](#) (on page 10-3)

[errorqueue.clear\(\)](#) (on page 15-87)

[errorqueue.next\(\)](#) (on page 15-88)

errorqueue.next()

This function reads the oldest entry from the error queue and removes it from the queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
errorCode, message, severity, errorNode = errorqueue.next()
```

errorCode	The error code number for the entry
message	The message that describes the error code
severity	The severity level (0, 10, 20, 30, or 40); see Details for more information
errorNode	The node number where the error originated

Details

Entries are stored in a first-in, first-out (FIFO) queue. This function reads the oldest entry and removes it from the queue.

Error codes and messages are listed in the [Error summary list](#) (on page 12-2).

If there are no entries in the queue, code 0, `Queue is Empty`, is returned.

Returned severity levels are described in the following table.

Number	Error level	Description
0	NO_SEVERITY	The message is information only. This level is used when the error queue is empty; the message does not represent an error.
10	INFORMATIONAL	The message is information only. This level is used to indicate status changes; the message does not represent an error.
20	RECOVERABLE	The error was caused by improper use of the instrument or by conditions that can be corrected. This message indicates that an error occurred. The instrument is still operating normally.
30	SERIOUS	There is a condition that prevents the instrument from functioning properly. The message indicates that the instrument is presently operating in an error condition. If the condition is corrected, the instrument returns to normal operation.
40	FATAL	There is a condition that cannot be corrected that prevents the instrument from functioning properly. Disconnect the DUT and turn the power off and then on again. If the error is a hardware fault that persists after cycling the power, the instrument must be repaired.

In an expanded system, each TSP-Link enabled instrument is assigned a node number. The variable *errorNode* stores the node number where the error originated.

Example

```
errorcode, message = errorqueue.next()
print(errorcode, message)
```

Reads the oldest entry in the error queue. The output below indicates that the queue is empty.

Output:

```
0.00000e+00 Queue Is Empty
```

Also see

[Error queue](#) (on page 10-3)

[errorqueue.clear\(\)](#) (on page 15-87)

[errorqueue.count](#) (on page 15-88)

[Error summary list](#) (on page 12-2)

eventlog.all()

This function returns all entries from the event log as a single string and removes them from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
logString = eventlog.all()
```

<i>logString</i>	A listing of all event log entries

Details

This function returns all events in the event log. Logged items are shown from oldest to newest. The response is a string that has the messages delimited with a new line character.

This function also clears the event log.

If there are no entries in the event log, this function returns the value *nil*.

Example

```
print(eventlog.all())
```

Get and print all entries from the event log and remove the entries from the log.

Output:

```
17:26:35.690 10 Oct 2019, LAN0, 192.168.1.102, LXI, 0, 1570728395,
1192037155.733269000, 0, 0x0
17:26:39.009 10 Oct 2019, LAN5, 192.168.1.102, LXI, 0, 1570728399,
1192037159.052777000, 0, 0x0
```

Also see

[eventlog.clear\(\)](#) (on page 15-90)

[eventlog.count](#) (on page 15-90)

[eventlog.enable](#) (on page 15-91)

[eventlog.next\(\)](#) (on page 15-92)

[eventlog.overwritemethod](#) (on page 15-93)

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 messages from the event log.

Also see

[eventlog.all\(\)](#) (on page 15-89)

[eventlog.count](#) (on page 15-90)

[eventlog.enable](#) (on page 15-91)

[eventlog.next\(\)](#) (on page 15-92)

[eventlog.overwritemethod](#) (on page 15-93)

eventlog.count

This attribute returns the number of unread events in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Clearing event log Reading event log	Not applicable	Not applicable

Usage

```
N = eventlog.count
```

N

The number of events in the event log

Example

```
print(eventlog.count)
```

Displays the present number of events in the instrument event log.

Output looks similar to:

```
3.00000e+00
```

Also see

[eventlog.all\(\)](#) (on page 15-89)

[eventlog.clear\(\)](#) (on page 15-90)

[eventlog.enable](#) (on page 15-91)

[eventlog.next\(\)](#) (on page 15-92)

[eventlog.overwritemethod](#) (on page 15-93)

eventlog.enable

This attribute enables or disables the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Not saved	1 (eventlog.ENABLE)

Usage

```
status = eventlog.enable
eventlog.enable = status
```

status

The enable status of the event log:

1 or eventlog.ENABLE: Event log enable

0 or eventlog.DISABLE: Event log disable

Details

When the event log is disabled (eventlog.DISABLE or 0), no new events are added to the event log. You can, however, read and remove existing events.

When the event log is enabled, new events are logged.

Example

```
print(eventlog.enable)
eventlog.enable = eventlog.DISABLE
print(eventlog.enable)
```

Displays the present status of the 2606B event log.

Output:

```
1.00000e+00
```

```
0.00000e+00
```

Also see

[eventlog.all\(\)](#) (on page 15-89)

[eventlog.clear\(\)](#) (on page 15-90)

[eventlog.count](#) (on page 15-90)

[eventlog.next\(\)](#) (on page 15-92)

[eventlog.overwritemethod](#) (on page 15-93)

eventlog.next()

This function returns the oldest unread event message from the event log and removes it from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
logString = eventlog.next()
```

logString	The next log entry

Details

Returns the next entry from the event log and removes it from the log.

If there are no entries in the event log, returns the value `nil`.

Example 1

```
print(eventlog.next())
```

Get the oldest message in the event log and remove that entry from the log.

Output:

```
17:28:22.085 10 Oct 2019, LAN2, 192.168.1.102, LXI, 0, 1570728502, <no time>, 0, 0x0
```

Example 2

```
print(eventlog.next())
```

If you send this command when there is nothing in the event log, you get the following output:

```
nil
```

Also see

[eventlog.all\(\)](#) (on page 15-89)

[eventlog.clear\(\)](#) (on page 15-90)

[eventlog.count](#) (on page 15-90)

[eventlog.enable](#) (on page 15-91)

[eventlog.overwritemethod](#) (on page 15-93)

eventlog.overwritemethod

This attribute controls how the event log processes events if the event log is full.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Not saved	1 (eventlog.DISCARD_OLDEST)

Usage

```
method = eventlog.overwritemethod
eventlog.overwritemethod = method
```

method

Set to one of the following values:

- 0 or eventlog.DISCARD_NEWEST: New entries are not logged
- 1 or eventlog.DISCARD_OLDEST: Old entries are deleted as new events are logged

Details

When this attribute is set to eventlog.DISCARD_NEWEST, new entries are not logged.

When this attribute is set to eventlog.DISCARD_OLDEST, the oldest entry is discarded when a new entry is added.

Example

```
eventlog.overwritemethod = 0
```

When the log is full, the event log ignores new entries.

Also see

[eventlog.all\(\)](#) (on page 15-89)
[eventlog.clear\(\)](#) (on page 15-90)
[eventlog.count](#) (on page 15-90)
[eventlog.enable](#) (on page 15-91)
[eventlog.next\(\)](#) (on page 15-92)

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 15-407)

fileVar:close()

This function closes the file that is represented by the *fileVar* variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar:close()
```

<i>fileVar</i>	The file descriptor variable to close

Details

This command is equivalent to `io.close(fileVar)`.

Note that files are automatically closed when the file descriptors are garbage collected.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

print("\n*** fileVar:close")
do
    myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
    myfile:write("Line 1")
    myfile:close()
end
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
myfile:close()
os.remove(fileName)
```

Opens file myfile.txt for writing. If no errors were found while opening, writes Removing file and closes the file.

Also see

[fileVar:flush\(\)](#) (on page 15-95)

[fileVar:read\(\)](#) (on page 15-96)

[fileVar:seek\(\)](#) (on page 15-97)

[fileVar:write\(\)](#) (on page 15-99)

[io.close\(\)](#) (on page 15-117)

[io.open\(\)](#) (on page 15-119)

fileVar:flush()

This function writes buffered data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar:flush()
```

<i>fileVar</i>	The file descriptor variable to flush

Details

The `fileVar:write()` or `io.write()` functions buffer data, which may not be written immediately to the USB flash drive. Use `fileVar:flush()` to flush this data. Using this function removes the need to close a file after writing to it, allowing the file to be left open to write more 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, and you want to keep the file open, flush the file after you write to it to prevent loss of data.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()
print("\n*** io.read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1\n")
myfile:flush()
myfile:close()
do
    fileHandle = io.input(fileName)
    value = io.read("*a")
    print(value)
end
fileHandle:close()

print(errorqueue.next())
Writes data to a USB flash drive.
```

Also see

[fileVar:write\(\)](#) (on page 15-99)

[io.open\(\)](#) (on page 15-119)

[io.write\(\)](#) (on page 15-123)

fileVar:read()

This function reads data from a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
data1 = fileVar:read()
data1 = fileVar:read(format1)
data1, data2 = fileVar:read("format1", "format2")
data1, ..., datan = fileVar:read("format1", ..., "formatn")
```

<i>data1</i>	First data read from the file
<i>data2</i>	Second data read from the file
<i>datan</i>	Last data read from the file
<i>fileVar</i>	The descriptor of the file to be read
<i>format1</i>	A string or number indicating the first type of data to be read
<i>format2</i>	A string or number indicating the second type of data to be read
<i>formatn</i>	A string or number indicating the last type of data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

"*n": Returns a number.

"*a": Returns the whole file, starting at the current position (returns an empty string if the current file position is at the end of the file).

"*l": Returns the next line, skipping the end of line; returns `nil` if the current file position is at the end of file.

n: Returns a string with up to *n* characters; returns an empty string if *n* is zero; returns `nil` if the current file position is at the end of file.

If no format parameters are provided, the function performs as if the function is passed the value "*l".

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Example

```

local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

print("fileVar:read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
    myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
    contents = myfile:read("*a")
    print(contents)
end
myfile:close()
os.remove(fileName)

```

Reads data from the input file.

Also see

[fileVar:write\(\)](#) (on page 15-99)

[io.input\(\)](#) (on page 15-118)

[io.open\(\)](#) (on page 15-119)

fileVar:seek()

This function sets and gets the present position of a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```

position, errorMsg = fileVar:seek()
position, errorMsg = fileVar:seek("whence")
position, errorMsg = fileVar:seek("whence", offset)

```

<i>position</i>	The new file position, measured in bytes from the beginning of the file
<i>errorMsg</i>	A string containing the error message
<i>fileVar</i>	The file descriptor variable
<i>whence</i>	A string indicating the base against which <i>offset</i> is applied; the default is "cur"
<i>offset</i>	The intended new position, measured in bytes from a base indicated by <i>whence</i> (default is 0)

Details

The *whence* parameters may be any of the following:

"set": Beginning of file

"cur": Current position

"end": End of file

If an error is encountered, it is logged to the error queue, and the command returns `nil` and the error string.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

print("\n*** fileVar:seek")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
    myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
    position = myfile:seek("end", -1)
    print(position)
end
myfile:close()
os.remove(fileName)
```

Get the present position of a file.

Also see

[io.open\(\)](#) (on page 15-119)

fileVar:write()

This function writes data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar:write(data)
fileVar:write(data1, data2)
fileVar:write(data1, ..., datan)
```

<i>fileVar</i>	The file descriptor variable
<i>data</i>	Write all data to the file
<i>data1</i>	The first data to write to the file
<i>data2</i>	The second data to write to the file
<i>datan</i>	The last data to write to the file
...	One or more entries (or values) separated by commas

Details

This function may buffer data until a flush (*fileVar:flush()* or *io.flush()*) or close (*fileVar:close()* or *io.close()*) operation is performed.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

print("\n*** fileVar:write")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
do
    myfile:write("Line 1")
end
myfile:close()
os.remove(fileName)

Write data to a file.
```

Also see

[fileVar:close\(\)](#) (on page 15-94)
[fileVar:flush\(\)](#) (on page 15-95)
[io.close\(\)](#) (on page 15-117)
[io.flush\(\)](#) (on page 15-117)
[io.open\(\)](#) (on page 15-119)

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	Instrument reset Recall setup	Not saved	6

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

<i>precision</i>	A number representing the number of digits to be printed for numbers printed with the <code>print()</code> , <code>printbuffer()</code> , and <code>printnumber()</code> functions; must be a number between 1 and 16

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

<pre>format.asciiprecision = 10 x = 2.54 printnumber(x) format.asciiprecision = 3 printnumber(x)</pre>	Output: <pre>2.540000000e+00 2.54e+00</pre>

Also see

[format.byteorder](#) (on page 15-100)
[format.data](#) (on page 15-102)
[print\(\)](#) (on page 15-169)
[printbuffer\(\)](#) (on page 15-170)
[printnumber\(\)](#) (on page 15-171)

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)	Yes	Instrument reset Recall setup	Not saved	1 (format.LITTLEENDIAN)

Usage

```
order = format.byteorder
format.byteorder = order
```

`order`

Byte order value as follows:

- Most significant byte first: 0, `format.NORMAL`, `format.NETWORK`, or `format.BIGENDIAN`
- Least significant byte first: 1, `format.SWAPPED` or `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.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64` data formats.

`format.NORMAL`, `format.BIGENDIAN`, and `format.NETWORK` select the same byte order. `format.SWAPPED` and `format.LITTLEENDIAN` select the same byte order. Selecting which to use is a matter of preference.

Select the `format.SWAPPED` or `format.LITTLEENDIAN` byte order when sending data to a computer with a Microsoft Windows operating system.

Example

```
x = 1.23
format.data = format.REAL32
format.byteorder = format.LITTLEENDIAN
printnumber(x)
format.byteorder = format.BIGENDIAN
printnumber(x)
```

The output depends on the terminal program you use, but it looks something like:

```
#0p??
#0??p
```

Also see

[format.asciiprecision](#) (on page 15-100)

[format.data](#) (on page 15-102)

[printbuffer\(\)](#) (on page 15-170)

[printnumber\(\)](#) (on page 15-171)

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	Instrument reset Recall setup	Not saved	1 (format.ASCII)

Usage

```
value = format.data
format.data = value
```

<code>value</code>	The format to use for data, set to one of the following values: ■ ASCII format: 1 or <code>format.ASCII</code> ■ Single-precision IEEE Std 754 binary format: 2, <code>format.SREAL</code>, or <code>format.REAL32</code> ■ Double-precision IEEE Std 754 binary format: 3, <code>format.REAL</code>, <code>format.REAL64</code>, or <code>format.DREAL</code>

Details

The precision of numeric values can be controlled with the `format.asciiprecision` attribute. The byte order of `format.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64` can be selected with the `format.byteorder` attribute.

REAL32 and SREAL select the same single precision format. REAL and REAL64 select the same double-precision format. They are alternative identifiers. Selecting which to use is a matter of preference.

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.

NOTE

Binary formats are not intended to be interpreted by humans.

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 15-100)

[format.byteorder](#) (on page 15-100)

[printbuffer\(\)](#) (on page 15-170)

[printnumber\(\)](#) (on page 15-171)

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")
```

<i>workingDirectory</i>	Returned value containing the working path
<i>path</i>	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 is:

`/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()
```

<i>path</i>	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 is:

`/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")
```

<i>status</i>	Whether or not the given path is a directory (true or false)
<i>path</i>	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 15-105)

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")
```

<code>status</code>	true if the given path is a file; otherwise, false
<code>path</code>	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 15-104)

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")
```

<i>path</i>	The returned path of the new directory
<i>newpath</i>	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 15-107)

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")
```

<i>files</i>	A table containing the names of all the file system entries in the specified directory
<i>path</i>	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")
```

<i>path</i>	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 15-106)

gettimezone()

This function retrieves the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timeZone = gettimezone()
```

<code>timeZone</code>	The local time zone of the instrument

Details

See `settimezone()` for additional details about the time zone format and a description of the fields.

`timeZone` can be in either of the following formats:

- If one parameter was used with `settimezone()`, the format used is:
GMThh:mm:ss
- If four parameters were used with `settimezone()`, the format used is:
GMThh:mm:ssGMThh:mm:ss,Mmm.w.dw/hh:mm:ss,Mmm.w.dw/hh:mm:ss

Example

```
timezone = gettimezone()
```

Reads the value of the local time zone.

Also see

[settimezone\(\)](#) (on page 15-189)

gm_isweep()

This KIParlib factory script function performs a linear current sweep and calculates the transconductance (G_m) at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
gm_array, vbuf, ibuf = gm_isweep(smu, start_i, stop_i, points)
```

<i>gm_array</i>	A Lua table containing the calculated G_m values at each point
<i>vbuf</i>	A reading buffer containing the measured voltage at each point
<i>ibuf</i>	A reading buffer containing the measured current at each point
<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>start_i</i>	Starting current level of the sweep
<i>stop_i</i>	Ending current level of the sweep
<i>points</i>	Number of measurements between <i>start_i</i> and <i>stop_i</i> (must be ≥ 2)

Details

Output data includes transconductance values, reading buffer with measured voltages, reading buffer with measured voltages and currents.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

The `gm_isweep()` function performs a linear current sweep, measuring voltage and current, and then calculating the transconductance (G_m) at each point using the central difference method. It can return an array of G_m values, a reading buffer with the measured voltages, and a reading buffer with the measured currents.

Example

<code>gm_array = gm_isweep(smua, 0, 0.01, 20)</code>	Source-measure unit (SMU) A returns G_m values only.
<code>gm_array, vbuf = gm_isweep(smua, 0, 0.01, 20)</code>	SMU A returns G_m and reading buffer with measured voltages.
<code>gm_array, vbuf, ibuf = gm_isweep(smua, 0, 0.01, 20)</code>	SMU A returns G_m and reading buffers with measured voltages and currents.

Also see

[gm_vsweep\(\)](#) (on page 15-110)

[KIParlib factory script](#) (on page 13-22)

gm_vsweep()

This KIParlib factory script function performs a linear voltage sweep and calculates the transconductance (G_m) at each point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
gm_array, ibuf, vbuf = gm_vsweep(smu, start_v, stop_v, points)
```

<i>gm_array</i>	A Lua table containing the calculated G_m values at each point
<i>ibuf</i>	A reading buffer containing the measured current at each point
<i>vbuf</i>	A reading buffer containing the measured voltage at each point
<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>start_v</i>	Starting voltage level of the sweep
<i>stop_v</i>	Ending voltage level of the sweep
<i>points</i>	Number of measurements between <i>start_v</i> and <i>stop_v</i> (must be ≥ 2)

Details

Output data includes transconductance values, reading buffer with measured currents, reading buffer with measured currents and voltages.

The `gm_vsweep()` function performs a linear voltage sweep, measuring voltage and current, and then calculating the transconductance (G_m) at each point using the central difference method. It can return an array of G_m values, a reading buffer with the measured currents, and a reading buffer with the measured voltages.

Example

<code>gm_array = gm_vsweep(smua, 0, 5, 20)</code>	SMU A returns G_m values only.
<code>gm_array, ibuf = gm_vsweep(smua, 0, 5, 20)</code>	SMU A returns G_m and reading buffer with measured currents.
<code>gm_array, ibuf, vbuf = gm_vsweep(smua, 0, 5, 20)</code>	SMU A returns G_m and reading buffers with measured currents and voltages.

Also see

[gm_isweep\(\)](#) (on page 15-109)

[KIParlib factory script](#) (on page 13-22)

i_leakage_measure()

This KHighC factory script function performs a current leakage measurement after stepping the output voltage.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
imeas = i_leakage_measure(smuX, levelv, limiti, sourcedelay, measurei, measuredelay)
```

<i>imeas</i>	The measured current
<i>X</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>levelv</i>	Voltage level to step to when this function is called
<i>limiti</i>	Current limit setting for the voltage step
<i>sourcedelay</i>	Delay to wait before lowering the current limit for measurement
<i>measurei</i>	Current limit (and measure range); the current limit is lower at this level and because high-capacitance mode is active, the measure range follows
<i>measuredelay</i>	Delay to wait after lowering the current limit before making the measurement

Details

This function causes the SMU to:

- Change its current limit to *limiti* with a voltage output of *levelv* for *sourcedelay* time, and then change its current limit to *measurei* (that also changes the measurement range to *measurei*) for *measuredelay* time
- When *measuredelay* time expires, a measurement is made and returned as *imeas*

When measuring leakage current:

- Charge the capacitor before calling this function (the output of the instrument is usually at a nonzero voltage before calling this function; when measuring leakage, this function does not charge the capacitor)
- Set *levelv* = 0

Example

```
smua.source.highc = smua.ENABLE
smua.source.levelv = 5
smua.source.output = smua.OUTPUT_ON
delay(1)
imeas = i_leakage_measure(smua, 0, 1, 300e-3, 10e-6, 0.1)
```

Enable high-capacitance mode. Charge the capacitor at 5 V for 1 second set by *delay(1)*.

The parameters passed on to the *i_leakage_measure()* function in this example are:

```
smu = smua
levelv = 0 V
limiti = 1 A
sourcedelay = 300 ms
measurei = 10 µA range
measuredelay = 100 ms
```

The levels and delays depend on the value and type of capacitor used.

Also see

[i_leakage_threshold\(\)](#) (on page 15-112)

[High-capacitance mode](#) (on page 5-52)

[KIHighC factory script](#) (on page 13-22)

i_leakage_threshold()

This KIHighC factory script function measures the current and compares it to a threshold. This continues until either the measured current drops below the threshold or the timeout expires.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f = i_leakage_threshold(smuX, levelv, limiti, sourcedelay, measurei, measuredelay,
                        threshold, timeout)
```

<i>f</i>	A Boolean flag; this flag is <code>true</code> when the current is below the threshold, <code>false</code> if threshold is not reached before timeout expires
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua</code> applies to SMU channel A)
<i>levelv</i>	Voltage level to step to when this function is called
<i>limiti</i>	Current limit setting for the voltage step
<i>sourcedelay</i>	Delay to wait before lowering the current limit for measurement
<i>measurei</i>	Current limit (and measure range); the current limit is lower at this level and because high-capacitance mode is active, the measure range follows
<i>measuredelay</i>	Delay before the first measurement after measure range is changed
<i>threshold</i>	The specified current that establishes the test limit
<i>timeout</i>	Amount of time (in seconds) to wait for the current to drop to <i>threshold</i> after all the delays have occurred

Details

This function causes the SMU to:

- Change its current limit to *limiti* with a voltage output of *levelv* for *sourcedelay* time, and then changes its current limit to *measurei* (that also changes the measurement range to *measurei*) for *measuredelay* time.
- When *measuredelay* time expires, measurements are made at a rate determined by the `smuX.measure.nplc` setting.

When testing the leakage current threshold:

- Charge the capacitor before calling this function. The output of the instrument is usually at a non-zero voltage before calling this function; when measuring leakage, this function does not charge the capacitor.
- If testing the leakage current threshold of the device, set *levelv* = 0.

Example

```
smua.source.highc = smua.ENABLE
smua.source.levelv = 5
smua.source.output = smua.OUTPUT_ON
delay(1)
pass = i_leakage_threshold(smua, 0, 1, 300e-3, 10e-6, 100e-3, 1e-6, 1)
```

Enable high-capacitance mode.
Charge the capacitor.

The parameters passed on to the `i_threshold_measure()` function in this example are:

```
smu = smua
levelv = 0 V
limiti = 1 A
sourcedelay = 300 ms
measurei = 10 µA range
measuredelay = 100 ms
threshold = 1 µA
timeout = 1 s
```

The levels and delays depend on the value and type of capacitor used.

Sets `pass = true` if the current is measured below 1 µA in less than 1 second.

Also see

[High-capacitance mode](#) (on page 5-52)

[i_leakage_measure\(\)](#) (on page 15-111)

[High-capacitance mode](#) (on page 5-52)

[KIHighC factory script](#) (on page 13-22)

InitiatePulseTest()

This KIPulse factory script function initiates the pulse configuration assigned to *tag*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = InitiatePulseTest(tag)
```

<i>f</i>	A Boolean flag; this flag is <code>true</code> when the pulse was successfully configured, <code>false</code> when errors are encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <code>false</code> , <i>msg</i> contains an error message; if it is <code>true</code> , <i>msg</i> contains a string that indicates successful configuration
<i>tag</i>	Numeric identifier of the pulse configuration to be initiated

Details

This function only initiates configured pulse trains assigned to a valid *tag*. Configure the pulse before initiating it using one of the `ConfigurePulse*` functions (refer to the **Also see** section).

Example

```
smua.reset()

smua.source.rangev = 5
smua.source.rangei = 1
smua.source.levelv = 0

smua.measure.rangev = 5
smua.measure.rangei = 1
smua.measure.nplc = 0.01
smua.measure.autozero = smua.AUTOZERO_ONCE

smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode = 1

smua.source.output = smua.OUTPUT_ON

f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5, 1, 0.002, 0.2, 10, smua.nvbuffer1, 1)

if f1 == true then
    f2, msg2 = InitiatePulseTest(1)
    print("Initiate message:", msg2)
else
    print("Config errors:", msg1)
end
```

Configure channel A to generate a pulse train. If no errors are encountered, initiate the pulse train. Channel A pulses voltage from a bias level of 0 V to a pulse level of 5 V. The pulse level is present for 2 ms and the bias level for 200 ms, with a 1 A limit setting. A total of 10 pulses is generated, and the measurement data is stored in `smua.nvbuffer1`. This pulse train is assigned to `tag = 1`.

Also see

[ConfigPulseIMeasureV\(\)](#) (on page 15-37)
[ConfigPulseIMeasureVSweepLin\(\)](#) (on page 15-39)
[ConfigPulseIMeasureVSweepLog\(\)](#) (on page 15-41)
[ConfigPulseVMeasureI\(\)](#) (on page 15-43)
[ConfigPulseVMeasureISweepLin\(\)](#) (on page 15-45)
[ConfigPulseVMeasureISweepLog\(\)](#) (on page 15-47)
[KIPulse factory script](#) (on page 13-20)

InitiatePulseTestDual()

This [KIPulse factory script](#) (on page 13-20) function initiates the pulse configuration assigned *tag1* and *tag2*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
f, msg = InitiatePulseTestDual(tag1, tag2)
```

<i>f</i>	A Boolean flag; this flag is <i>true</i> when the pulse was successfully configured, <i>false</i> when errors were encountered
<i>msg</i>	A string message; if the <i>f</i> flag is <i>false</i> , <i>msg</i> contains an error message; if it is <i>true</i> , <i>msg</i> contains a string indicating successful configuration
<i>tag1</i>	Numeric identifier of the first pulse configuration to be initiated
<i>tag2</i>	Numeric identifier of the second pulse configuration to be initiated

Details

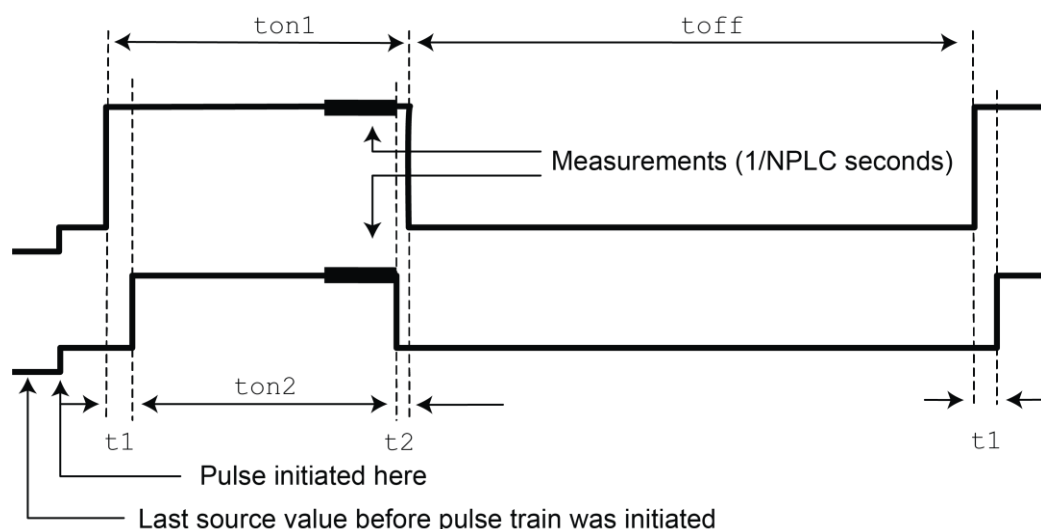
The pulse trains associated with the indicated tags are generated simultaneously. This is useful when testing devices such as voltage regulators, where the input signal and output load must be applied to the instrument at the same time.

When using this function, each *tag1* pulse encapsulates each *tag2* pulse in time. Specifically, the *tag1* pulse transitions from its bias level to its pulse level before the *tag2* pulse. Both the *tag1* and *tag2* pulses return to their respective bias levels at approximately the same time. Measurements for both pulse trains occur at the same time (see the waveform in the figure below).

To provide this encapsulation, the following rules are enforced:

- The *tag1* pulse on time, *ton1*, must be configured to be > 40 μ s longer than the *tag2* pulse on time.
- The *tag1* and *tag2* pulse off times, *toff*, must be the same.

Figure 142: InitiatePulseTestDual



Example

```

smua.reset()
smua.source.rangev = 5
smua.source.rangei = 1
smua.source.levelv = 0
smua.measure.rangev = 5
smua.measure.rangei = 1
smua.measure.nplc = 0.01
smua.measure.autozero = smua.AUTOZERO_ONCE

smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode = 1

smua.source.output = smua.OUTPUT_ON

smub.reset()
smub.source.func = smub.OUTPUT_DCAMPS
smub.source.rangei = 1
smub.source.rangev = 5
smub.source.leveli = 0
smub.measure.rangei = 1
smub.measure.rangev = 5
smub.measure.nplc = 0.01
smub.measure.autozero = smub.AUTOZERO_ONCE
smub.nvbuffer1.clear()
smub.nvbuffer1.appendmode = 1
smub.source.output = smub.OUTPUT_ON

f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5, 1, 0.002, 0.2, 10, smua.nvbuffer1, 1)
f2, msg2 = ConfigPulseIMeasureV(smub, 0,-1, 5, 0.001, 0.2, 10, smub.nvbuffer1, 2)
if (f1 == true) and (f2 == true) then
    f3, msg3 = InitiatePulseTestDual(1, 2)
    print("Initiate message:", msg3)
else
    print("Config errors:", msg1, msg2)
end

```

Set up channels A and B for pulse operation, configure pulse trains for each channel, and then initiate the pulse trains if no errors are encountered.

Channel A pulses voltage from a bias level of 0 V to pulse level of 5 V. The pulse level is present for 2 ms, and the bias level for 200 ms with a 1 A limit setting.

A total of 10 pulses is generated on channel A and the measurement data is stored in `smua.nvbuffer1`. This pulse train is assigned to `tag = 1`.

Channel B pulses current from a bias level of 0 A to pulse level of 1 A. The pulse level is present for 1 ms, and the bias level for 200 ms with a 5 V limit setting.

A total of 10 pulses is generated on channel B, and the measurement data is stored in `smub.nvbuffer1`. This pulse train is assigned to `tag = 2`.

Also see

[ConfigPulseIMeasureV\(\)](#) (on page 15-37)

[ConfigPulseIMeasureVSweepLin\(\)](#) (on page 15-39)

[ConfigPulseIMeasureVSweepLog\(\)](#) (on page 15-41)

[ConfigPulseVMeasureI\(\)](#) (on page 15-43)

[ConfigPulseVMeasureISweepLin\(\)](#) (on page 15-45)

[ConfigPulseVMeasureISweepLog\(\)](#) (on page 15-47)

io.close()

This function closes a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
io.close()
io.close(file)
```

<i>file</i>	The descriptor of the file to close

Details

If a file is not specified, the default output file closes.

Only `io.close()`, used without specifying a parameter, can be accessed from a remote node.

Example

```
testFile, testError = io.open("testfile.txt", "w")
if nil == testError then
    testFile:write("This is my test file")
    io.close(testFile)
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

[io.open\(\)](#) (on page 15-119)

io.flush()

This function saves buffered data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
io.flush()
```

Details

You must use the `io.flush()` or `io.close()` functions to write data to the file system.

NOTE

Data is not automatically written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB flash drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

This function only flushes the default output file.

Using this command removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written (and when you want to keep the file open and not close it), flush the file after writing to it.

Also see

[fileVar:flush\(\)](#) (on page 15-95)

[fileVar:write\(\)](#) (on page 15-99)

[io.write\(\)](#) (on page 15-123)

io.input()

This function assigns a previously opened file, or opens a new file, as the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
fileVar = io.input()  
fileVar = io.input("newfile")
```

<i>fileVar</i>	The descriptor of the input file or an error message (if the function fails)
<i>newfile</i>	A string representing the path of a file to open as the default input file, or the file descriptor of an open file to use as the default input file

Details

The *newfile* path may be absolute or relative to the current working directory.

When using this function from a remote TSP-Link® node, this command does not accept a file descriptor and does not return a value.

If the function fails, an error message is returned.

Also see

[io.open\(\)](#) (on page 15-119)

[io.output\(\)](#) (on page 15-119)

io.open()

This function opens a file for later reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar, errorMsg = io.open("path")
fileVar, errorMsg = io.open("path", "mode")
```

<i>fileVar</i>	The descriptor of the opened file
<i>errorMsg</i>	Indicates whether an error was encountered while processing the function
<i>path</i>	The path of the file to open
<i>mode</i>	A string representing the intended access mode ("r" = read, "w" = write, and "a" = append)

Details

The path to the file to open may be absolute or relative to the current working directory. If you successfully open the file, *errorMsg* is nil and *fileVar* has the descriptor used to access the file.

If an error is encountered, the command returns nil for *fileVar* and an error string.

Example

```
testFile, testError = io.open("testfile.txt", "w")
if testError == nil then
    testFile:write("This is my test file")
    io.close(testFile)
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

[io.close\(\)](#) (on page 15-117)

io.output()

This function assigns a previously opened file or opens a new file as the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
fileVar = io.output()
fileVar = io.output("newfile")
```

<i>fileVar</i>	The descriptor of the output file or an error message (if the function fails)
<i>newfile</i>	A file descriptor to assign (or the path of a file to open) as the default output file

Details

The path of the file to open may be absolute or relative to the current working directory.

When accessed from a remote node using the TSP-Link network, this command does not accept a file descriptor parameter and does not return a value.

If the function fails, an error message is returned.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

print("\n*** io.output")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
    fileHandle = io.output(fileName)
    print(fileHandle)
end
io.close(fileHandle)
print(fileHandle)
os.remove(fileName)
```

Assign the file to be the default output file.

Also see

[io.input\(\)](#) (on page 15-118)

[io.open\(\)](#) (on page 15-119)

io.read()

This function reads data from the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data1 = io.read()
data1 = io.read("format1")
data1, data2 = io.read("format1", "format2")
data1, ..., dataN = io.read("format1", ..., "formatN")
```

<i>data1</i>	The data read from the file
<i>data2</i>	The data read from the file
<i>dataN</i>	The data read from the file; the number of return values matches the number of format values given
<i>format1</i>	A string or number indicating the type of data to be read
<i>format2</i>	A string or number indicating the type of data to be read
<i>formatN</i>	A string or number indicating the type of data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

Format parameter	Description
"*n"	Returns a number
"*a"	Returns the whole file, starting at the present position; returns an empty string if it is at the end of file
"*l"	Default setting; returns the next line, skipping the end of line; returns <i>nil</i> if the present file position is at the end of file
<i>N</i>	Returns a string with up to <i>N</i> characters; returns an empty string if <i>N</i> is zero (0); returns <i>nil</i> if the present file position is at the end of file

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Example

```

local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

-- io.read
print("\n*** io.read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1\n")
myfile:flush()
myfile:close()
do
    fileHandle = io.input(fileName)
    value = io.read("*a")
    print(value)
end
fileHandle:close()

print(errorqueue.next())

```

Read data from the default input file.

Also see

None

io.type()

This function checks whether or not a given object is a file handle.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
type = io.type(obj)
```

<i>type</i>	Indicates whether the object is an open file handle
<i>obj</i>	Object to check

Details

Returns the string "file" if the object is an open file handle. If it is not an open file handle, `nil` is returned.

Example

```

local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

print("\n*** io.type")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
    fileHandle = io.output(fileName)
    state = io.type(fileHandle)
    print(state)
end
io.close(fileHandle)
local state = io.type(fileHandle)
print(state)
os.remove(fileName)

```

Check whether or not `fileName` is a file handle.

Also see

[io.open\(\)](#) (on page 15-119)

io.write()

This function writes data to the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```

io.write()
io.write(data1)
io.write(data1, data2)
io.write(data1, ..., dataN)

```

<i>data1</i>	The data to be written
<i>data2</i>	The data to be written
<i>dataN</i>	The data to be written
...	One or more values separated by commas

Details

All data parameters must be either strings or numbers.

NOTE

Data is not immediately written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB flash drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

print("\n*** io.write")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
    fileHandle = io.output(fileName)
    io.write("Line 2")
end
io.close(fileHandle)
os.remove(fileName)
```

Writes data to the default output file.

Also see

[io.flush\(\)](#) (on page 15-117)

lan.applysettings()

This function re-initializes the LAN interface with new settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.applysettings()
```

Details

Disconnects all existing LAN connections to the instrument and re-initializes the LAN with the present configuration settings.

This function initiates a background operation. LAN configuration could be a lengthy operation. Although the function returns immediately, the LAN initialization continues to run in the background.

Even though the LAN configuration settings may not have changed since the LAN was last connected, new settings may take effect due to the dynamic nature of dynamic host configuration protocol (DHCP) or dynamic link local addressing (DLLA) configuration.

Re-initialization takes effect even if the configuration has not changed since the last time the instrument connected to the LAN.

Example

```
lan.applysettings()
Re-initialize the LAN interface with new settings.
```

Also see

None

lan.autoconnect

This attribute is used to enable or disable link monitoring.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.autoconnect
lan.autoconnect = state
```

<i>state</i>	LAN link monitoring state: 1 or <code>lan.ENABLE</code> : Enables automatic link reconnection and monitoring 0 or <code>lan.DISABLE</code> : Disables automatic link reconnection and monitoring

Details

This attribute sets the LAN link monitoring and automatic connection state.

When this is set to `lan.ENABLE`, all connections are closed if the link to the LAN is lost for more than the time specified by `lan.linktimeout`.

Set this attribute to `lan.ENABLE` to automatically reset the LAN connection after the LAN link is established.

Example

```
lan.autoconnect = lan.ENABLE
```

Enable LAN link monitoring.

Also see

[lan.linktimeout](#) (on page 15-133)

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.dns.address[N]

Configures DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
dnsAddress = lan.config.dns.address[N]
lan.config.dns.address[N] = "dnsAddress"
```

<i>dnsAddress</i>	DNS server IP address
<i>N</i>	Entry index (1 or 2)

Details

This attribute is an array of Domain Name System (DNS) server addresses. These addresses take priority for DNS lookups and are consulted before any server addresses that are obtained using DHCP. This allows local DNS servers to be specified that take priority over DHCP-configured global DNS servers.

You can specify up to two addresses. The address specified by 1 is consulted first for DNS lookups. *dnsAddress* must be a string specifying the IP address of the DNS server in dotted decimal notation.

Unused entries are returned as "0.0.0.0" when read. To disable an entry, set its value to "0.0.0.0" or the empty string "".

Although only two addresses may be manually specified here, the instrument uses up to three DNS server addresses. If two are specified here, only one that is given by a DHCP server is used. If no entries are specified here, up to three addresses that are given by a DHCP server are used.

Example

```
dnsaddress = "164.109.48.173"
lan.config.dns.address[1] = dnsaddress
```

Set the DNS address 1 to 164.109.48.173.

Also see

[lan.config.dns.domain](#) (on page 15-127)

[lan.config.dns.dynamic](#) (on page 15-127)

[lan.config.dns.hostname](#) (on page 15-128)

[lan.config.dns.verify](#) (on page 15-129)

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.dns.domain

Configures the dynamic DNS domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	""

Usage

```
domain = lan.config.dns.domain
lan.config.dns.domain = "domain"
```

<i>domain</i>	Dynamic DNS registration domain; use a string of 255 characters or less

Details

This attribute holds the domain to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the domain specified in this attribute with the DNS server.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 255 characters are allowed, you must make sure the combined length is also no more than 255 characters.

Example

```
print(lan.config.dns.domain)
```

Outputs the present dynamic DNS domain. For example, if the domain is "Matrix", the response is:

```
Matrix
```

Also see

[lan.config.dns.dynamic](#) (on page 15-127)
[lan.config.dns.hostname](#) (on page 15-128)
[lan.config.dns.verify](#) (on page 15-129)
[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.dns.dynamic

Enables or disables the dynamic DNS registration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.config.dns.dynamic
lan.config.dns.dynamic = state
```

<i>state</i>	The dynamic DNS registration state. It may be one of the following values: 1 or <code>lan.ENABLE</code>: Enabled 0 or <code>lan.DISABLE</code>: Disabled

Details

Dynamic DNS registration works with DHCP to register the host name with the DNS server. The host name is specified in the `lan.config.dns.hostname` attribute.

Example

```
print(lan.config.dns.dynamic)
```

Outputs the dynamic registration state.

If dynamic DNS registration is enabled, the response is:

```
1.00000e+00
```

Also see

[lan.config.dns.hostname](#) (on page 15-128)

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.dns.hostname

This attribute defines the dynamic DNS host name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	Instrument specific (see Details)

Usage

```
hostName = lan.config.dns.hostname
lan.config.dns.hostname = hostName
```

hostName

The host name to use for dynamic DNS registration; the host name must:

- be a string of 63 characters or less
- start with a letter
- end with a letter or digit
- contain only letters, digits, and hyphens

Details

This attribute holds the host name to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the host name specified in this attribute with the DNS server.

The factory default value for *hostName* is "k-<model number>-<serial number>", where <model number> and <serial number> are replaced with the actual model number and serial number of the instrument (for example, "k-2606B-1234567"). Note that hyphens separate the characters of *hostName*.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 63 characters can be entered here, care must be taken to be sure the combined length is no more than 255 characters.

Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero, or one consisting entirely of whitespace characters) will revert the host name to the factory default value.

Example

```
print(lan.config.dns.hostname)
```

Outputs the present dynamic DNS host name.

Also see

[lan.config.dns.dynamic](#) (on page 15-127)

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.dns.verify

This attribute defines the DNS host name verification state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.config.dns.verify
lan.config.dns.verify = state
```

state

DNS hostname verification state:

- 1 or lan.ENABLE: DNS host name verification enabled
- 0 or lan.DISABLE: DNS host name verification disabled

Details

When this is enabled, the instrument performs DNS lookups to verify that the DNS host name matches the value specified by `lan.config.dns.hostname`.

Example

```
print(lan.config.dns.verify)
```

Outputs the present DNS host name verification state.
If it is enabled, the output is:
1.000000e+00

Also see

[lan.config.dns.hostname](#) (on page 15-128)

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.duplex

This attribute defines the LAN duplex mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.FULL)

Usage

```
duplex = lan.config.duplex
lan.config.duplex = duplex
```

duplex

LAN duplex setting can be one of the following values:

- 1 or `lan.FULL`: Selects full-duplex operation
- 0 or `lan.HALF`: Selects half-duplex operation

Details

This attribute does not indicate the actual setting currently in effect. Use the `lan.status.duplex` attribute to determine the present operating state of the LAN.

Example

```
lan.config.duplex = lan.FULL
```

Set the LAN duplex mode to full.

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.gateway

This attribute contains the LAN default gateway address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
gatewayAddress = lan.config.gateway
lan.config.gateway = "gatewayAddress"
```

gatewayAddress

LAN default gateway address; must be a string specifying the default IP address of the gateway in dotted decimal notation

Details

This attribute specifies the default gateway IP address to use when manual or DLLA configuration methods are used to configure the LAN. If DHCP is enabled, this setting is ignored.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.gateway` attribute to determine the present operating state of the LAN.

The IP address must be formatted in four groups of numbers, each separated by a decimal.

Example

```
print(lan.config.gateway)
```

Outputs the default gateway address. For example, you might see the output:
192.168.0.1

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

[lan.status.gateway](#) (on page 15-139)

lan.config.ipaddress

This command specifies the LAN IP address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"192.168.0.2"

Usage

```
ipAddress = lan.config.ipaddress
lan.config.ipaddress = "ipAddress"
```

<i>ipAddress</i>	LAN IP address; must be a string specifying the IP address in dotted decimal notation

Details

This command specifies the LAN IP address to use when the LAN is configured using the manual configuration method. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.ipaddress` attribute to determine the present operating state of the LAN.

Example

```
ipaddress = lan.config.ipaddress
Retrieves the presently set LAN IP address.
```

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

[lan.status.ipaddress](#) (on page 15-139)

lan.config.method

This attribute contains the LAN settings configuration method.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0 (lan.AUTO)

Usage

```
method = lan.config.method
lan.config.method = method
```

<i>method</i>	The method for configuring LAN settings; it can be one of the following values: ■ 0 or <code>lan.AUTO</code>: Selects automatic sequencing of configuration methods ■ 1 or <code>lan.MANUAL</code>: Use only manually specified configuration settings

Details

This attribute controls how the LAN IP address, subnet mask, default gateway address, and DNS server addresses are determined.

When method is `lan.AUTO`, 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, it uses the manually specified settings.

When method is `lan.MANUAL`, only the manually specified settings are used. Neither DHCP nor DLLA are attempted.

Example

```
print(lan.config.method)
```

Outputs the present method.

For example:

```
1.00000e+00
```

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.config.speed

This attribute contains the LAN speed used when restarting in manual configuration mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	100 (100 Mbps)

Usage

```
speed = lan.config.speed
```

```
lan.config.speed = speed
```

```
speed
```

LAN speed setting in Mbps (10 or 100)

Details

This attribute stores the speed that is used if the LAN is restarted for manual configuration operation.

This attribute does not indicate the actual setting presently in effect. Use the `lan.status.speed` attribute to determine the present operating state of the LAN.

The LAN speed is measured in megabits per second (Mbps).

Example

```
lan.config.speed = 100
```

Configure LAN speed for 100.

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

[lan.status.speed](#) (on page 15-142)

lan.config.subnetmask

This attribute contains the LAN subnet mask.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"255.255.255.0"

Usage

```
mask = lan.config.subnetmask
lan.config.subnetmask = "mask"
```

<i>mask</i>	String that specifies the LAN subnet mask value in dotted decimal notation

Details

This attribute specifies the LAN subnet mask that is used when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting presently in effect. Use the `lan.status.subnetmask` attribute to determine the present operating state of the LAN.

Example

```
print(lan.config.subnetmask)
```

Outputs the LAN subnet mask, such as:
255.255.255.0

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)
[lan.status.subnetmask](#) (on page 15-143)

lan.linktimeout

This attribute contains the LAN link timeout period.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	20 (20 s)

Usage

```
timeout = lan.linktimeout
lan.linktimeout = timeout
```

<i>timeout</i>	The LAN link monitor time-out period (in seconds)

Details

You must enable the command `lan.autoconnect` before you can use this attribute.

The *timeout* value represents the amount of time that passes before the instrument disconnects from the LAN due to the loss of the LAN link integrity.

The LAN interface does not disconnect if the connection to the LAN is reestablished before the *timeout* value expires.

If the LAN link integrity is not restored before the *timeout* value expires, the instrument begins to monitor for a new connection.

Example

```
print(lan.linktimeout)
```

Outputs the present LAN link timeout setting.

Also see

[lan.autoconnect](#) (on page 15-125)

[lan.restoredefaults\(\)](#) (on page 15-136)

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.nagle

This attribute controls the state of the LAN Nagle algorithm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (lan.DISABLE)

Usage

```
state = lan.nagle
lan.nagle = state
```

state

The state of the Nagle algorithm:

- 1 or `lan.ENABLE`: Enable the LAN Nagle algorithm for TCP connections
- 0 or `lan.DISABLE`: Disable the Nagle algorithm for TCP connections

Details

This attribute enables or disables the use of the LAN Nagle algorithm on transmission control protocol (TCP) connections.

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.reset()

This function resets the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.reset()
```

Details

This function resets the LAN interface. It performs the commands `lan.restoredefaults()` and `lan.applysettings()`. It also resets the LAN password.

Also see

[lan.applysettings\(\)](#) (on page 15-125)
[lan.restoredefaults\(\)](#) (on page 15-136)
[localnode.password](#) (on page 15-156)

lan.restoredefaults()

This function resets LAN settings to default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.restoredefaults()
```

Details

The settings that are restored are shown in the following table.

Settings that are restored to default	
Attribute	Default setting
lan.autoconnect	lan.ENABLE
lan.config.dns.address[N]	"0.0.0.0"
lan.config.dns.domain	""
lan.config.dns.dynamic	lan.ENABLE
lan.config.dns.hostname	"K-<model number>-<serial number>"
lan.config.dns.verify	lan.ENABLE
lan.config.duplex	lan.FULL
lan.config.gateway	"0.0.0.0"
lan.config.ipaddress	"192.168.0.2"
lan.config.method	lan.AUTO
lan.config.speed	100
lan.config.subnetmask	"255.255.255.0"
lan.linktimeout	20 (seconds)
lan.lxidomain	0
lan.nagle	lan.DISABLE
lan.timedwait	20 (seconds)

This command is run when `lan.reset()` is sent.

Example

```
lan.restoredefaults()
Restores the LAN defaults.
```

Also see

[lan.reset\(\)](#) (on page 15-135)

[localnode.password](#) (on page 15-156)

lan.status.dns.address[N]

This attribute contains the DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
dnsAddress = lan.status.dns.address[N]
```

<i>dnsAddress</i>	DNS server IP address
<i>N</i>	Entry index (1, 2, or 3)

Details

This attribute is an array of DNS server addresses. The instrument can use up to three addresses.

Unused or disabled entries are returned as "0.0.0.0" when read. The *dnsAddress* returned is a string specifying the IP address of the DNS server in dotted decimal notation.

You can only specify two addresses manually. However, the instrument uses up to three DNS server addresses. If two are specified, only the one given by a DHCP server is used. If no entries are specified, up to three address given by a DHCP server are used.

The value of `lan.status.dns.address[1]` is referenced first for all DNS lookups. The values of `lan.status.dns.address[2]` and `lan.status.dns.address[3]` are referenced second and third, respectively.

Example

```
print(lan.status.dns.address[1])
```

Outputs DNS server address 1, for example:
164.109.48.173

Also see

[lan.status.dns.name](#) (on page 15-137)

lan.status.dns.name

This attribute contains the present DNS fully qualified host name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
hostName = lan.status.dns.name
```

<i>hostName</i>	Fully qualified DNS host name that can be used to connect to the instrument

Details

A fully qualified domain name (FQDN) specifies its exact location in the tree hierarchy of the Domain Name System (DNS).

A FQDN is the complete domain name for a specific computer or host on the LAN. The FQDN consists of two parts: The host name and the domain name.

If the DNS host name for an instrument is not found, this attribute stores the IP address in dotted decimal notation.

Example

```
print(lan.status.dns.name)
```

Outputs the dynamic DNS host name.

Also see

[lan.config.dns.address\[N\]](#) (on page 15-126)

[lan.config.dns.hostname](#) (on page 15-128)

lan.status.duplex

This attribute contains the duplex mode presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
duplex = lan.status.duplex
```

duplex

LAN duplex setting can be one of the following values:

- 0 or `lan.HALF`: half-duplex operation
- 1 or `lan.FULL`: full-duplex operation

Example

```
print(lan.status.duplex)
```

Outputs the present LAN duplex mode, such as:
1.000000e+00

Also see

None

lan.status.gateway

This attribute contains the gateway address presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
gatewayAddress = lan.status.gateway
```

<code>gatewayAddress</code>	LAN gateway address presently being used

Details

The value of `gatewayAddress` is a string that indicates the IP address of the gateway in dotted decimal notation.

Example

```
print(lan.status.gateway)
```

Outputs the gateway address, such as:
192.168.0.1

Also see

[lan.config.gateway](#) (on page 15-130)

lan.status.ipaddress

This attribute contains the LAN IP address presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
ipAddress = lan.status.ipaddress
```

<code>ipAddress</code>	LAN IP address specified in dotted decimal notation

Details

The IP address is a character string that represents the IP address assigned to the instrument.

Example

```
print(lan.status.ipaddress)
```

Outputs the LAN IP address currently in use, such as:
192.168.0.2

Also see

[lan.config.ipaddress](#) (on page 15-131)

lan.status.macaddress

This attribute contains the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
macAddress = lan.status.macaddress
```

<code>macAddress</code>	The instrument MAC address

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 (see Example).

Example

```
print(lan.status.macaddress)
```

Outputs the MAC address of the instrument, for example:
08:00:11:00:00:57

Also see

None

lan.status.port.dst

This attribute contains the LAN dead socket termination port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.dst
```

<code>port</code>	Dead socket termination socket port number

Details

This attribute holds the TCP port number used to reset all other LAN socket connections.

To reset all LAN connections, open a connection to the DST port number.

Example

```
print(lan.status.port.dst)
```

Outputs the LAN dead socket termination port number, such as:
5.03000e+03

Also see

None

lan.status.port.rawsocket

This attribute contains the LAN raw socket connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.rawsocket
```

port	Raw socket port number

Details

The TCP port number used to connect the instrument and to control the instrument over a raw socket communication interface.

Example

```
print(lan.status.port.rawsocket)
```

Outputs the LAN raw socket port number, such as:
5.02500e+03

Also see

None

lan.status.port.telnet

This attribute contains the LAN Telnet connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.telnet
```

port	Telnet port number

Details

This attribute holds the TCP port number used to connect to the instrument to control it over a Telnet interface.

Example

```
print(lan.status.port.telnet)
```

Get the LAN Telnet connection port number.
Output:
2.30000e+01

Also see

None

lan.status.port.vxi11

This attribute contains the LAN VXI-11 connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.vxi11
```

port	LAN VXI-11 port number

Details

This attribute stores the TCP port number used to connect to the instrument over a VXI-11 interface.

Example

```
print(lan.status.port.vxi11)
```

Outputs the VXI-11 number, such as:
1.02400e+03

Also see

None

lan.status.speed

This attribute contains the LAN speed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
speed = lan.status.speed
```

speed	LAN speed in Mbps, either 10 or 100

Details

This attribute indicates the transmission speed currently in use by the LAN interface.

Example

```
print(lan.status.speed)
```

Outputs the transmission speed of the instrument presently in use, such as:
1.00000e+02

Also see

None

lan.status.subnetmask

This attribute contains the LAN subnet mask that is presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
mask = lan.status.subnetmask
```

<code>mask</code>	A string specifying the subnet mask in dotted decimal notation

Details

Use this attribute to determine the present operating state of the LAN. This attribute returns the present LAN subnet mask value if the LAN is manually configured, or when DLLA or DHCP is used.

Example

```
print(lan.status.subnetmask)
```

Outputs the subnet mask of the instrument that is presently in use, such as:
255.255.255.0

Also see

[lan.config.subnetmask](#) (on page 15-133)

lan.timedwait

This attribute contains the LAN timed-wait state interval.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	20 (20 s)

Usage

```
timeout = lan.timedwait  
lan.timedwait = timeout
```

<code>timeout</code>	The LAN timed-wait state interval in seconds

Details

This attribute controls the amount of time that resources are allocated to closed TCP connections. When a TCP connection is closed, the connection is put in a timed-wait state and resources remain allocated for the connection until the timed-wait state ends. During the timed-wait interval, the instrument processes delayed packets that arrive after the connection is closed.

Use this attribute to tailor the timed-wait state interval for the instrument.

Example

<code>lan.timedwait = 30</code>	Set the amount of time resources are allocated to TCP connection to 30 s.

Also see

[lan.restoredefaults\(\)](#) (on page 15-136)

lan.trigger[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event ID.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.trigger[N].assert()
```

<i>N</i>	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

```
lan.trigger[5].assert()
```

Creates a trigger with LAN packet 5.

Also see

[lan.lxidomain](#) (on page 15-134)

[lan.trigger\[N\].clear\(\)](#) (on page 15-144)

[lan.trigger\[N\].mode](#) (on page 15-148)

[lan.trigger\[N\].overrun](#) (on page 15-149)

[lan.trigger\[N\].stimulus](#) (on page 15-151)

[lan.trigger\[N\].wait\(\)](#) (on page 15-152)

[Understanding hardware value and pseudo line state](#) (on page 7-22)

lan.trigger[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

```
lan.trigger[N].clear()
```

<i>N</i>	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

```
lan.trigger[5].clear()
```

Clears the event detector with LAN packet 5.

Also see

[lan.trigger\[N\].assert\(\)](#) (on page 15-144)

[lan.trigger\[N\].overrun](#) (on page 15-149)

[lan.trigger\[N\].stimulus](#) (on page 15-151)

[lan.trigger\[N\].wait\(\)](#) (on page 15-152)

lan.trigger[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

```
lan.trigger[N].connect()
```

<i>N</i>	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

```
lan.trigger[1].protocol = lan.MULTICAST  
lan.trigger[1].connect()  
lan.trigger[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

[lan.trigger\[N\].assert\(\)](#) (on page 15-144)

[lan.trigger\[N\].ipaddress](#) (on page 15-147)

[lan.trigger\[N\].overrun](#) (on page 15-149)

[lan.trigger\[N\].protocol](#) (on page 15-150)

[lan.trigger\[N\].stimulus](#) (on page 15-151)

[lan.trigger\[N\].wait\(\)](#) (on page 15-152)

lan.trigger[N].connected

This attribute stores 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 = lan.trigger[N].connected
```

<i>connected</i>	The LAN event connection state: ■ <code>true</code>: Connected ■ <code>false</code>: Not connected
<i>N</i>	The LAN event number (1 to 8)

Details

This read-only attribute is set to `true` when the LAN trigger is connected and ready to send trigger events following a successful `lan.trigger[N].connect()` command; if the LAN trigger is not ready to send trigger events, this value is `false`.

This attribute is also `false` when either `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes are changed or the remote connection closes the connection.

Example

```
lan.trigger[1].protocol = lan.MULTICAST
print(lan.trigger[1].connected)
```

Outputs `true` if connected, or `false` if not connected.
Example output:
`false`

Also see

[lan.trigger\[N\].connect\(\)](#) (on page 15-145)

[lan.trigger\[N\].ipaddress](#) (on page 15-147)

[lan.trigger\[N\].protocol](#) (on page 15-150)

lan.trigger[N].disconnect()

This function disconnects the LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.trigger[N].disconnect()
```

<i>N</i>	The LAN event number (1 to 8)

Details

For TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes for this event are changed.

Also see

[lan.trigger\[N\].ipaddress](#) (on page 15-147)

[lan.trigger\[N\].protocol](#) (on page 15-150)

lan.trigger[N].EVENT_ID

This constant is the event identifier used to route the LAN trigger to other subsystems (using stimulus properties).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
lan.trigger[N].EVENT_ID
```

<i>N</i>	The LAN event number (1 to 8)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to incoming LAN trigger packets.

Example

```
digio.trigger[14].stimulus = lan.trigger[1].EVENT_ID
Route occurrences of triggers on LAN trigger 1 to digital I/O trigger 14.
```

Also see

None

lan.trigger[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	Instrument reset LAN trigger N reset Recall setup	Not saved	"0.0.0.0"

Usage

```
ipAddress = lan.trigger[N].ipaddress
lan.trigger[N].ipaddress = "ipAddress"
```

<i>ipAddress</i>	The LAN address for this attribute as a string in dotted decimal notation
<i>N</i>	The LAN event number (1 to 8)

Details

Sets the IP address for outgoing trigger events.

Set to "0.0.0.0" for multicast.

After changing this setting, the `lan.trigger[N].connect()` command must be called before outgoing messages can be sent.

Example

```
lan.trigger[3].protocol = lan.TCP
lan.trigger[3].ipaddress = "192.168.1.100"
lan.trigger[3].connect()
```

Set the protocol for LAN trigger 3 to be `lan.TCP` when sending LAN triggers.
Use IP address "192.168.1.100" to connect the LAN trigger.

Also see

[lan.trigger\[N\].connect\(\)](#) (on page 15-145)

lan.trigger[N].mode

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	Instrument reset LAN trigger N reset Recall setup	Not saved	0 (lan.TRIG_EITHER)

Usage

```
mode = lan.trigger[N].mode
lan.trigger[N].mode = mode
```

<i>mode</i>	A number representing the trigger mode (0 to 7); see the Details section for more information
<i>N</i>	A number representing 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	Number	Trigger packets detected as input	LAN trigger packet generated for output with a...
lan.TRIG_EITHER	0	Rising or falling edge (positive or negative state)	negative state
lan.TRIG_FALLING	1	Falling edge (negative state)	negative state
lan.TRIG_RISING	2	Rising edge (positive state)	positive state
lan.TRIG_RISINGA	3	Rising edge (positive state)	positive state
lan.TRIG_RISINGM	4	Rising edge (positive state)	positive state
lan.TRIG_SYNCHRONOUS	5	Falling edge (negative state)	positive state
lan.TRIG_SYNCHRONOUSA	6	Falling edge (negative state)	positive state
lan.TRIG_SYNCHRONOUSM	7	Rising edge (positive state)	negative state

`lan.TRIG_RISING` and `lan.TRIG_RISINGA` are the same.

`lan.TRIG_RISING` and `lan.TRIG_RISINGM` are the same.

Use of either `lan.TRIG_SYNCHRONOUSA` or `lan.TRIG_SYNCHRONOUSM` instead of `lan.TRIG_SYNCHRONOUS` is preferred. Use of `lan.TRIG_SYNCHRONOUS` is provided for compatibility with older products and other Keithley Instruments products.

Example

```
print(lan.trigger[1].mode)
```

Outputs the present LAN trigger mode of LAN event 1.

Also see

[Digital I/O](#) (on page 7-37)

[TSP-Link system expansion interface](#) (on page 14-52)

lan.trigger[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 LAN trigger N reset Instrument reset Recall setup	Not applicable	Not applicable

Usage

```
overrun = lan.trigger[N].overrun
```

<code>overrun</code>	The trigger overrun state for the specified LAN packet (<code>true</code> or <code>false</code>)
<code>N</code>	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. Output trigger overrun indications are provided in the status model.

Example

```
overrun = lan.trigger[5].overrun
print(overrun)
```

Checks the overrun status of a trigger on LAN5 and outputs the value, such as:
`false`

Also see

[lan.trigger\[N\].assert\(\)](#) (on page 15-144)

[lan.trigger\[N\].clear\(\)](#) (on page 15-144)

[lan.trigger\[N\].stimulus](#) (on page 15-151)

[lan.trigger\[N\].wait\(\)](#) (on page 15-152)

lan.trigger[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	Instrument reset LAN trigger N reset Recall setup	Not saved	0 (lan.TCP)

Usage

```
protocol = lan.trigger[N].protocol
lan.trigger[N].protocol = protocol
```

<i>protocol</i>	The protocol to use for messages from the trigger: 0 or lan.TCP 1 or lan.UDP 2 or lan.MULTICAST
<i>N</i>	The LAN event number (1 to 8)

Details

The LAN trigger listens for trigger messages on all supported protocols, but uses the designated protocol for sending outgoing messages. After changing this setting, `lan.trigger[N].connect()` must be called before outgoing event messages can be sent.

When the `lan.MULTICAST` protocol is selected, the `lan.trigger[N].ipaddress` attribute is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

```
print(lan.trigger[1].protocol)
```

Get LAN protocol to use for sending trigger messages for LAN event 1.

Also see

[lan.trigger\[N\].connect\(\)](#) (on page 15-145)

[lan.trigger\[N\].ipaddress](#) (on page 15-147)

lan.trigger[N].pseudostate

This attribute sets the simulated line state for the LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Not saved	1

Usage

```
pseudostate = lan.trigger[N].pseudostate
lan.trigger[N].pseudostate = pseudostate
```

<i>pseudostate</i>	The simulated line state (0 or 1)
<i>N</i>	A number representing the LAN event number (1 to 8)

Details

This attribute can be set to initialize the pseudo line state to a known value.

Setting this attribute does not cause the LAN trigger to generate any events or output packets.

Example

```
print(lan.trigger[1].pseudostate)
```

Get the present simulated line state for the LAN event 1.

Also see

None

lan.trigger[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	Instrument reset LAN trigger N reset Recall setup	Not saved	0

Usage

```
triggerStimulus = lan.trigger[N].stimulus
lan.trigger[N].stimulus = triggerStimulus
```

<i>triggerStimulus</i>	The LAN event identifier used to trigger the event
<i>N</i>	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 *triggerStimulus* to one of the trigger event IDs, which are shown in the following table.

Trigger event IDs*	
Event ID	Event description
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>

Trigger event IDs*	
Event ID	Event description
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Setting this attribute to zero disables automatic trigger generation.

If any events are detected prior to calling `lan.trigger[N].connect()`, the event is ignored and the action overrun is set.

Example

```
lan.trigger[5].stimulus = trigger.timer[1].EVENT_ID
```

Use timer 1 trigger event as the source for LAN packet 5 trigger stimulus.

Also see

[lan.trigger\[N\].assert\(\)](#) (on page 15-144)
[lan.trigger\[N\].clear\(\)](#) (on page 15-144)
[lan.trigger\[N\].connect\(\)](#) (on page 15-145)
[lan.trigger\[N\].overrun](#) (on page 15-149)
[lan.trigger\[N\].wait\(\)](#) (on page 15-152)

lan.trigger[N].wait()

This function waits for an input trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = lan.trigger[N].wait(timeout)
```

<code>triggered</code>	Trigger detection indication (true or false)
<code>N</code>	The trigger packet over LAN to wait for (1 to 8)
<code>timeout</code>	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 `lan.trigger[N].wait()` or `lan.trigger[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 = lan.trigger[5].wait(3)
```

Wait for a trigger with LAN packet 5 with a timeout of 3 seconds.

Also see

[lan.trigger\[N\].assert\(\)](#) (on page 15-144)

[lan.trigger\[N\].clear\(\)](#) (on page 15-144)

[lan.trigger\[N\].overrun](#) (on page 15-149)

[lan.trigger\[N\].stimulus](#) (on page 15-151)

localnode.autolinefreq

This attribute enables or disables automatic power line frequency detection at start-up.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
flag = localnode.autolinefreq
localnode.autolinefreq = flag
```

flag

The auto line frequency detection setting:

- `true`: Enable automatic line frequency detection at start-up
- `false`: Disable automatic line frequency detection at start-up

Details

When this attribute is set to `true`, the power line frequency is detected automatically the next time the 2606B powers up. After the power line frequency is automatically detected at power-up, the `localnode.linefreq` attribute is set automatically to 50 or 60.

If the `localnode.linefreq` attribute is explicitly set, `localnode.autolinefreq` is automatically set to `false`.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].autolinefreq`.

Also see

[localnode.linefreq](#) (on page 15-155)

localnode.description

This attribute stores a user-defined description and mDNS service name of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	Instrument specific (see Details)

Usage

```
localnode.description = "description"
description = localnode.description
```

<i>description</i>	User-defined description and mDNS service name of the instrument; use a string of 63 characters or less

Details

This attribute stores a string that contains a description of the instrument. This value appears on LXI welcome page of the instrument. The value of this attribute is also used as the mDNS service name of the instrument.

The factory default value of this attribute is "Keithley Instruments SMU <model number> - <serial number>", where <model number> and <serial number> are replaced with the actual model number and serial number of the instrument. Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero or a string consisting entirely of whitespace characters) reverts the description to the factory default value.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].description`.

Example

```
description = "System in Lab 05"
localnode.description = description
Set description to System in Lab 05.
```

Also see

None

localnode.license

This attribute returns the product license agreements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

```
license_agreement = localnode.license
```

<i>license_agreement</i>	The text of the license agreements

Example

```
print(localnode.license)
Returns the license agreements for the 2606B.
```

Also see

None

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 (RW)	Yes	Not applicable	Nonvolatile memory	60 (60 Hz)

Usage

```
frequency = localnode.linefreq
localnode.linefreq = frequency
```

<i>frequency</i>	An integer representing the detected or specified line frequency of the instrument

Details

To achieve optimum noise rejection when performing measurements at integer NPLC apertures, set the line frequency attribute to match the frequency (50 Hz or 60 Hz) of the ac power line.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].linefreq`. When this attribute is set, the `localnode.autolinefreq` attribute is automatically set to `false`. You can have the instrument automatically detect the ac power line frequency and set this attribute with the line frequency detected when the instrument power is turned on by setting the `localnode.autolinefreq` attribute to `true`.

Example 1

<code>frequency = localnode.linefreq</code>	Reads line frequency setting.

Example 2

<code>localnode.linefreq = 60</code>	Sets the line frequency to 60 Hz.

Also see

[localnode.autolinefreq](#) (on page 15-153)

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
```

<i>model</i>	The model number of the instrument

Example

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

Outputs the model number of the local node. For example:
2606B

Also see

[localnode.serialno](#) (on page 15-161)

localnode.password

This attribute stores the remote access password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	LAN reset LAN restore defaults	Nonvolatile memory	""

Usage

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

password

A string that contains the remote interface password, up to 255 characters

Details

This write-only attribute stores the password that is set for any remote interface. When password usage is enabled (`localnode.passwordmode`), you must supply a password to change the configuration or to control an instrument from a remote command interface.

The instrument continues to use the old password for all interactions until the command to change it executes. When changing the password, give the instrument time to execute the command before attempting to use the new password.

You cannot retrieve a lost password from any command interface.

You can reset the password by resetting the LAN from the virtual front panel or by sending the `lan.reset()` command.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].password`.

Example

```
localnode.password = "N3wpa55w0rd"
```

Changes the remote interface password to N3wpa55w0rd.

Also see

[lan.reset\(\)](#) (on page 15-135)

[localnode.passwordmode](#) (on page 15-157)

localnode.passwordmode

This attribute stores the password enable mode for remote access to the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	1 (localnode.PASSWORD_WEB)

Usage

```
mode = localnode.passwordmode
localnode.passwordmode = mode
```

<i>mode</i>	The remote password enable mode

Details

This attribute controls if and where remote access passwords are required. Set this attribute to one of the values below to enable password checking:

- `localnode.PASSWORD_NONE` or 0: Disable passwords everywhere
- `localnode.PASSWORD_WEB` or 1: Use passwords on the web interface only
- `localnode.PASSWORD_LAN` or 2: Use passwords on the web interface and all LAN interfaces
- `localnode.PASSWORD_ALL` or 3: Use passwords on the web interface and all remote command interfaces

NOTE

When a password is set for the web interface, you cannot make changes using the web interface options Virtual Front Panel, Flash Upgrade, or TSB Embedded.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].passwordmode`.

If you enable password mode, you must also assign a password.

Example

```
mode = localnode.PASSWORD_WEB
localnode.passwordmode = mode
localnode.password = "SMU1234"
```

Sets value of `mode` to `PASSWORD_WEB`.
Allows use of passwords on the web interface only.
Set the password to `SMU1234`.

Also see

[localnode.password](#) (on page 15-156)

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)	Yes	Power cycle	Not saved	0 (disabled)

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting

Prompting mode:

- Do not generate prompts: 1
- Generate prompts: 0

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 error queue 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 queue when the command processing was complete.
- **>>>>**. 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 are not 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 stops responding when you communicate using Test Script Builder because it is waiting for a common complete prompt from Test Script Builder.

Example

```
localnode.prompts = 1
Enable prompting.
```

Also see[localnode.showerrors](#) (on page 15-161)[tsplink.reset\(\)](#) (on page 15-380)

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)	Yes	Power cycle	Not saved	1 (enabled)

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

prompting

IEEE Std 488.2 prompting mode:

- Disable prompting: 0
- Enable prompting: 1

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 do 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 = 0
```

Disables IEEE Std 488.2 common command prompting.

Also see[localnode.prompts](#) (on page 15-158)

localnode.reset()

This function resets the local node instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.reset()
```


Details

If you want to reset a specific instrument or a subordinate node, use the `node[X].reset()` command.

A local node reset includes:

- Source-measure unit (SMU) attributes affected by a SMU reset are reset
- Other settings are restored back to factory default settings

A `localnode.reset()` is different than a `reset()` because `reset()` resets the entire system.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].reset()`.

Example

```
localnode.reset()
```

Resets the local node.

Also see

[reset\(\)](#) (on page 15-176)

[smuX.reset\(\)](#) (on page 15-231)

localnode.revision

This attribute stores the firmware revision level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
revision = localnode.revision
```

```
revision
```

Firmware revision level

Details

This attribute indicates the revision number of the firmware that is presently running in the instrument.

When using this command from a remote node, replace `localnode` with the node reference. For example, `node[5].revision`.

Example

```
print(localnode.revision)
```

Outputs the present revision level.

Sample output:

```
1.0.0
```

Also see

[localnode.description](#) (on page 15-154)

[localnode.model](#) (on page 15-155)

[localnode.serialno](#) (on page 15-161)

localnode.serialno

This attribute stores the serial number of the module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
serialno = localnode.serialno
```

<code>serialno</code>	The serial number of the module

Details

This indicates the module serial number.

The instrument serial number is not available through a remote interface.

Example

<pre>display.clear() display.settext(localnode.serialno)</pre>	Clears the virtual front panel display. Places the serial number of the module on the top line of its display.

Also see

[localnode.description](#) (on page 15-154)

[localnode.model](#) (on page 15-155)

[localnode.revision](#) (on page 15-160)

localnode.showerrors

This attribute sets whether or not the instrument automatically sends generated errors.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (disabled)

Usage

```
errorMode = localnode.showerrors
localnode.showerrors = errorMode
```

<code>errorMode</code>	Show error setting: - Show errors: 1 - Do not show errors: 0

Details

If this attribute is set to 1, the instrument automatically sends any generated errors stored in the error queue, and then clears the queue. Errors are processed after executing a command message (just before issuing a prompt if prompts are enabled).

If this attribute is set to 0, errors are left in the error queue and must be explicitly read or cleared.

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].showerrors`.

Example

```
localnode.showerrors = 1
```

Enables sending of generated errors.

Also see

[localnode.prompts](#) (on page 15-158)

makegetter()

This function creates a function to get the value of an attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
getter = makegetter(table, "attributeName")
```

<i>getter</i>	The return value
<i>table</i>	Read-only table where the attribute is located
<i>attributeName</i>	A string representing the name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the function created with `makegetter()` executes more quickly than accessing the attribute directly.

Creating a getter function is only useful if it is going to be called several times. Otherwise, the overhead of creating the getter function outweighs the overhead of accessing the attribute directly.

Example

```
getlevel = makegetter(smua.source, "levelv")
v = getlevel()
```

Creates a getter function called `getlevel`.
When `getlevel()` is called, it returns the value of `smua.source.levelv`.

Also see

[makesetter\(\)](#) (on page 15-163)

makesetter()

This function creates a function that, when called, sets the value of an attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
setter = makesetter(table, "attributeName")
```

<i>setter</i>	Function that sets the value of the attribute
<i>table</i>	Read-only table where the attribute is located
<i>attributeName</i>	The string name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the *setter* function execute more quickly than accessing the attribute directly.

Creating a *setter* function is only useful if it is going to be called several times. If you are not calling the *setter* function several times, it is more efficient to access the attribute directly.

Example

```
setlevel = makesetter(smua.source, "levelv")
for v = 1, 10 do
    setlevel(v)
end
```

Creates a setter function called *setlevel*.

Using *setlevel()* in the loop sets the value of *smua.source.levelv*, performing a source sweep.

Also see

[makegetter\(\)](#) (on page 15-162)

meminfo()

This function returns the present amount of available memory and the total amount of memory in the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
freeMem, totalMem = meminfo()
```

<i>freeMem</i>	The amount of free dynamically allocated memory available
<i>totalMem</i>	The total amount of dynamically allocated memory in the instrument

Details

This function returns two values:

- The amount of free dynamically allocated memory available in kilobytes
- The total amount of dynamically allocated memory on the instrument in kilobytes

The difference between the two values is the amount presently used.

Example

```
print(meminfo())
```

Retrieve the amount of free and total memory in the instrument.
Output:
2.89840e+04 3.27680e+04

Also see

None

node[N].execute()

This function starts test scripts from a remote node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
node[N].execute("scriptCode")
```

<i>N</i>	The node number of this instrument
<i>scriptCode</i>	A string containing the source code

Details

Only the remote master node can use the execute command to run a script on this node. This function does not run test scripts on the master node; only on this 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.

This command should only be used from a remote master when controlling this instrument over a TSP-Link®.

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

[TSP advanced features](#) (on page 14-57)

[tsplink.group](#) (on page 15-377)

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")
```

<i>value</i>	The value of the variable
<i>N</i>	The node number of this instrument (1 to 64)
<i>name</i>	The global variable name

Details

This function retrieves the value of a global variable from the runtime 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 15-166)

[TSP advanced features](#) (on page 14-57)

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)
```

<i>N</i>	The node number of this instrument (1 to 64)
<i>name</i>	The global variable name to set
<i>value</i>	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 15-165)

[TSP advanced features](#) (on page 14-57)

opc()

This function sets the operation complete status bit when all overlapped commands are completed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
opc ( )
```

Details

This function causes the operation complete bit in the Standard 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

[Status model](#) (on page 10-1)

[waitcomplete\(\)](#) (on page 15-407)

os.remove()

This function deletes the file or directory with a given name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
success, msg = os.remove("filename")
```

<i>success</i>	A success indicator (true or nil)
<i>msg</i>	A message value (nil or an error message)
<i>filename</i>	A string representing the name of the file or directory to delete

Details

Directories must be empty before using the `os.remove()` function to delete them.

If this function fails, it returns `nil` (for *success*) and an error message string (for *msg*).

Example

```
os.remove("testFile")
```

Delete the file named testFile.

Also see

[os.rename\(\)](#) (on page 15-167)

os.rename()

This function renames an existing file or directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
success, msg = os.rename("oldname", "newname")
```

<i>success</i>	A success indicator (true or nil)
<i>msg</i>	A message value (nil or an error message)
<i>oldname</i>	String representing the name of the file or directory to rename
<i>newname</i>	String representing the new name of the file or directory

Details

If this function fails, it returns `nil` (for *success*) and an error message string (for *msg*).

Example

```
os.rename("testFile", "exampleFile")
```

Changes the name of the existing file `testFile` to the name `exampleFile`.

Also see

[os.remove\(\)](#) (on page 15-167)

os.time()

This function generates a time value in UTC time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
utcTime = os.time()
utcTime = os.time(timespec)
```

<code>utcTime</code>	Time value in UTC time
<code>timespec</code>	The date and time (year, month, day, hour, and minute)

Details

The `timespec` is a table using the fields listed in the table below.

<code>year</code>	The year (1970 or later)
<code>month</code>	The month (1 to 12)
<code>day</code>	The day (1 to 31)
<code>hour</code>	The hour (00 to 23)
<code>min</code>	The minute (00 to 59)
<code>sec</code>	The second (00 to 59)

If the time (hour, minute, and second) options are not used, they default to noon for that day. When called without a parameter (the first form), the function returns the current time.

Set the time zone before calling the `os.time()` function.

Example

```
systemTime = os.time({year = 2019,
  month = 3,
  day = 31,
  hour = 14,
  min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2019 at 2:25 pm.

Also see

[settime\(\)](#) (on page 15-188)

[settimezone\(\)](#) (on page 15-189)

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)
```

<i>value1</i>	The first argument to output
<i>value2</i>	The second argument to output
<i>valueN</i>	The last argument to output
<i>...</i>	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 15-100)

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)
```

<i>startIndex</i>	Beginning index of the buffer to print; this must be more than one and less than <i>endIndex</i>
<i>endIndex</i>	Ending index of the buffer to print; this must be more than <i>startIndex</i> and less than the index of the last entry in the tables
<i>bufferVar</i>	First table or reading buffer subtable to print
<i>bufferVar2</i>	Second table or reading buffer subtable to print
<i>bufferVarN</i>	The last table or reading buffer subtable to print
...	One or more tables or reading buffer subtables separated with commas

Details

If $startIndex \leq 1$, 1 is used as *startIndex*. If $n < endIndex$, *n* is used as *endIndex*.

When any given reading buffers are used in overlapped commands that have not yet completed (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 measurements 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 response message is stored in the output queue.

The `format.data` attribute controls the format of the response message.

Example

```
format.data = format.ASCII
format.asciiprecision = 6
printbuffer(1, rb1.n, rb1)
```

This assumes that `rb1` is a valid reading buffer in the runtime environment. The use of `rb1.n` (*bufferVar.n*) indicates that the instrument should output all readings in the reading buffer. In this example, `rb1.n` equals 10.

Example of output data (`rb1.readings`):

```
4.07205e-05, 4.10966e-05, 4.06867e-05, 4.08865e-05, 4.08220e-05, 4.08988e-05,
4.08250e-05, 4.09741e-05, 4.07174e-05, 4.07881e-05
```

Also see

[bufferVar.n](#) (on page 15-28)
[bufferVar.readings](#) (on page 15-29)
[format.asciiprecision](#) (on page 15-100)
[format.byteorder](#) (on page 15-100)
[format.data](#) (on page 15-102)

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)

```

<i>value1</i>	First value to print in the configured format
<i>value2</i>	Second value to print in the configured format
<i>valueN</i>	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 15-100)
[format.byteorder](#) (on page 15-100)
[format.data](#) (on page 15-102)
[print\(\)](#) (on page 15-169)
[printbuffer\(\)](#) (on page 15-170)

PulseIMeasureV()

This KIPulse factory script function performs a specified number of pulse I, measure V cycles.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
PulseIMeasureV(smu, bias, level, ton, toff, points)
```

<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>bias</i>	Bias level in amperes
<i>level</i>	Pulse level in amperes
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles

Details

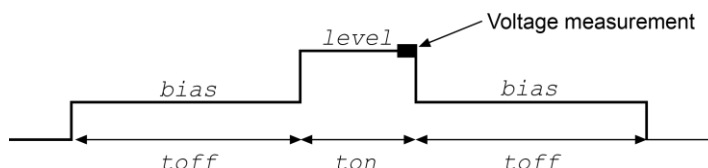
Data for pulsed voltage measurements, current levels, and timestamps are stored in `smua.nvbuffer1`.

If any parameters are omitted or `nil`, the operator is prompted to enter them using the front panel.

To perform the specified number of pulse I, measure V cycles, this function:

1. Sets the *smu* to output *bias* amperes and dwell for *toff* seconds.
2. Sets the *smu* to output *level* amperes and dwell for *ton* seconds.
3. Performs voltage measurement with source at *level* amperes.
4. Sets the *smu* to output *bias* amperes for *toff* seconds.
5. Repeats steps 2 through 4 for all remaining *points* pulse-measure cycles.

Figure 143: PulseIMeasureV



Example

```
PulseIMeasureV(smua, 0.001, 1.0,
  20e-3, 40e-3, 10)
```

SMU A outputs 1 mA and dwells for 40 ms, outputs 1 A and dwells for 20 ms. The voltage measurements occur during each 20 ms dwell period. After the measurement, the output returns to 1 mA and dwells for 40 ms. This pulse-measure process repeats nine more times.

Also see

[KIPulse factory script](#) (on page 13-20)

PulseVMeasureI()

This KIPulse factory script function performs a specified number of pulse V, measure I cycles.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`PulseVMeasureI(smu, bias, level, ton, toff, points)`

<i>smu</i>	Instrument channel (for example, <i>smua</i> refers to SMU channel A)
<i>bias</i>	Bias level in volts
<i>level</i>	Pulse level in volts
<i>ton</i>	Pulse on time in seconds
<i>toff</i>	Pulse off time in seconds
<i>points</i>	Number of pulse-measure cycles

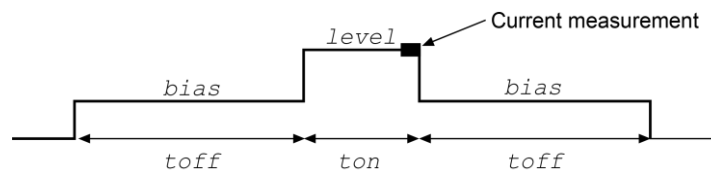
Details

If any parameters are omitted or `nil`, the operator is prompted to enter them using the front panel. Data for pulsed current measurements, voltage levels, and timestamps are stored in `smuX.nvbuffer1`.

To perform the specified number of pulse V, measure I cycles, this function:

1. Sets the *smu* to output *bias* volts and dwell for *toff* seconds
2. Sets the *smu* to output *level* volts and dwell for *ton* seconds
3. Performs voltage measurement with source at *level* volts
4. Sets the *smu* to output *bias* volts for *toff* seconds
5. Repeats steps 2 through 4 for the remaining *points* pulse-measure cycles

Figure 144: PulseVMeasureI()



Example

```
smua.measure.nplc = 0.001
PulseVMeasureI(smua, -1, 1, 1E-3, 2E-3, 20)
```

SMU A outputs -1 V and dwells for 2 ms, outputs 1 V and dwells for 1 ms. The current measurements occur during each 1 ms dwell period. After the measurement, the output returns to -1 V and dwells for 2 ms. This pulse-measure process repeats 19 more times.

Also see

[KIPulse factory script](#) (on page 13-20)

QueryPulseConfig()

This KIPulse factory script function allows you to inspect the settings of the preconfigured pulse train assigned to *tag*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tbl = QueryPulseConfig(tag)
```

<i>tag</i>	Numeric identifier to be assigned to the defined pulse train
<i>tbl</i>	Returned table

Details

Once a pulse train is configured and assigned to a *tag*, you can use the `QueryPulseConfig()` command to inspect the settings of this preconfigured pulse train.

This function returns a table that contains the settings associated with the *tag* input parameter.

Return values:	
<code>toString()</code>	A function that returns most settings in a string that is convenient for printing
<i>tag</i>	Identifying tag for this pulse train
<i>smu</i>	The SMU configured for pulsing
<i>func</i>	Pulse function: <code>smuX.OUTPUT_DCAMPS</code> or <code>smuX.OUTPUT_DCVOLTS</code>
<i>bias</i>	Pulse bias level
<i>level</i>	Pulse level for non-sweeping pulses
<i>start</i>	Starting level for sweep pulses
<i>stop</i>	Ending level for sweep pulses
<i>limit</i>	Limit value
<i>ton</i>	On time in seconds
<i>toff</i>	Off time in seconds
<i>points</i>	The number of points in this pulse train
<i>buf</i>	Reference to the buffer that contains measurement data
<i>sync_in</i>	The <i>sync_in</i> digio line, if used
<i>sync_out</i>	The <i>sync_out</i> digio line, if used
<i>sourcevalues</i>	A table containing the source value for each point in the pulse train

Example

```

smua.reset()

smua.source.rangev = 5
smua.source.rangei = 1
smua.source.levelv = 0

smua.measure.rangev = 5
smua.measure.rangei = 1
smua.measure.nplc = 0.01
smua.measure.autozero = smua.AUTOZERO_ONCE

smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode = 1

smua.source.output = smua.OUTPUT_ON

f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5,
    1, 0.002, 0.2, 10, smua.nvbuffer1, 1)

print(QueryPulseConfig(1).toString())

```

Configure channel A to generate a pulse train, query configuration, and then display as a string. Channel A pulses voltage from a bias level of 0 V to a pulse level of 5 V. The pulse level is present for 2 ms, and the bias level for 200 ms with a 1 A limit setting. A total of 10 pulses is generated, and the measurement data is stored in smua.nvbuffer1. This pulse train is assigned to tag = 1.

Output:

```

>> tag = 1
>> smu = smua
>> func = volts
>> type = pulse
>> bias = 0
>> level = 5
>> limit = 1
>> time on = 0.002
>> time off = 0.2
>> points = 10
>> measure = yes
>> sync_in = 0
>> sync_out = 0
>> sync_in_timeout = 0
>> sync_out_abort = 0
>> { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 }

```

Also see

[ConfigPulseMeasureV\(\)](#) (on page 15-37)
[ConfigPulseMeasureVSweepLin\(\)](#) (on page 15-39)
[ConfigPulseMeasureVSweepLog\(\)](#) (on page 15-41)
[ConfigPulseVMeasureI\(\)](#) (on page 15-43)
[ConfigPulseVMeasureISweepLin\(\)](#) (on page 15-45)
[ConfigPulseVMeasureISweepLog\(\)](#) (on page 15-47)
[KIPulse factory script](#) (on page 13-20)

reset()

This function resets commands to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
reset()
reset(system)
```

system

What to reset:

- `true`: If the node is the master, the entire system is reset (default)
- `false`: Only the local group is reset

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 either the `localnode.reset()` or `node[X].reset()` command. Use the `localnode.reset()` command for the local instrument. Use the `node[X].reset()` command to reset an instrument on a subordinate node.

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.

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 is generated.

Also see

[localnode.reset\(\)](#) (on page 15-159)

savebuffer()

This KISavebuffer factory script function saves a specified reading buffer as either a CSV file or an XML file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
savebuffer(buffer, "formatType", "fileName")
```

<i>buffer</i>	The reading buffer to save
<i>formatType</i>	A string indicating which file type to use: <code>csv</code> or <code>xml</code>
<i>fileName</i>	The file name of the saved buffer

Details

Use this function to save the specified buffer to a USB flash drive.

This function only saves to a USB flash drive.

You are not required to qualify the path to the USB flash drive, but you can add `/usb1/` before the *fileName* (see Example 2).

Example 1

```
savebuffer(smua.nvbuffer1, "csv", "mybuffer.csv")
```

Save smua dedicated reading buffer 1 as a CSV file named `mybuffer.csv`.

Example 2

```
savebuffer(smua.nvbuffer1, "csv", "/usb1/mybuffer.csv")
```

Save smua dedicated reading buffer 1 to an installed USB flash drive as a CSV file named `mybuffer.csv`.

Also see

[KISavebuffer factory script](#) (on page 13-22)

[smuX.savebuffer\(\)](#) (on page 15-231)

script.anonymous

This is a reference to the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	See Details	See Details	Not applicable

Usage

```
scriptVar = script.anonymous
```

<code>scriptVar</code>	The name of the variable that references the script

Details

You can use the `script.anonymous` script like any other script. Also, you can save the anonymous script as a user script by giving it a name.

This script is replaced by loading a script with the `loadscript` or `loadandruncscript` commands when they are used without a name.

Example 1

```
script.anonymous.list()
```

Displays the content of the anonymous script.

Example 2

```
print(script.anonymous.source)
```

Retrieves the source of the anonymous script.

Also see

[Anonymous scripts](#) (on page 14-4)
[scriptVar.autorun](#) (on page 15-184)
[scriptVar.list\(\)](#) (on page 15-185)
[scriptVar.name](#) (on page 15-185)
[scriptVar.run\(\)](#) (on page 15-186)
[scriptVar.save\(\)](#) (on page 15-187)
[scriptVar.source](#) (on page 15-188)

script.delete()

This function deletes a script from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.delete("scriptName")
```

<i>scriptName</i>	The string that represents the name of the script

Example

```
script.delete("test8")
```

Deletes a user script named `test8` from nonvolatile memory.

Also see

[Delete user scripts from the instrument](#) (on page 14-49)
[scriptVar.save\(\)](#) (on page 15-187)

script.factory.catalog()

This function returns an iterator that can be used in a `for` loop to iterate over all the factory scripts.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
for name in script.factory.catalog() do body end
```

<i>name</i>	String representing the name of the script
<i>body</i>	Code that implements the body of the <code>for</code> loop to process the names in the catalog

Details

Accessing this catalog of scripts allows you to process the factory scripts. 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 factory scripts. The `for` loop repeats until all scripts have been iterated.

Example

```
for name in script.factory.catalog() do
  print(name)
end
```

Retrieve the catalog listing for factory scripts.

Also see

None

script.load()

This function creates a script from a specified file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.load("file")
scriptVar = script.load("file", "name")
```

<i>scriptVar</i>	The created script; this is <i>nil</i> if an error is encountered
<i>file</i>	The path and file name of the script file to load
<i>name</i>	The name of the script to be created

Details

The file path may be absolute or relative to the current working directory. The root folder of the USB flash drive has the absolute path `"/usb1/"`. Both the forward slash (/) and backslash (\) are supported as directory separators.

The file to be loaded must start with the `loadscript` or `loadandruncscript` keywords, contain the body of the script, and end with the `endscript` keyword.

Script naming:

- If the *name* parameter is an empty string, or *name* is absent (or *nil*) and the script name cannot be extracted from the file, *scriptVar* is the only handle to the created script.
- If *name* is given (and not *nil*), any script name embedded in the file is ignored.
- If *name* conflicts with the name of an existing script in the `script.user.scripts` table, the existing script's name attribute is set to an empty string before it is replaced in the `script.user.scripts` table by the new script.
- If *name* is absent or *nil*, the command attempts to extract the name of the script from the file. Any conflict between the extracted name and that of an existing script in the scripts table generates an error. If the script name cannot be extracted, the created script's name attribute is initialized to the empty string and must be set to a valid nonempty string before saving the script to nonvolatile memory.

Example

```
myTest8 =
    script.load("/usb1/filename.tsp",
        "myTest8")
```

Loads the script `myTest8` from the USB flash drive.

Also see

[script.new\(\)](#) (on page 15-180)

script.new()

This function creates a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.new("code")
scriptVar = script.new("code", "name")
```

<code>scriptVar</code>	The name of the variable that references the script
<code>code</code>	A string containing the body of the script
<code>name</code>	The name of the script

Details

The `name` parameter is the name that is added to the `script.user.scripts` table. If `name` is not provided, an empty string is used, and the script is unnamed. If the name already exists in `script.user.scripts`, the `name` attribute of the existing script is set to an empty string before it is replaced by the new script.

Note that `name` is the value that is used for the instrument front panel display. If this value is not defined, the script is not available from the virtual front panel.

You must save the new script into nonvolatile memory to keep it when the instrument is turned off.

Example 1

```
myTest8 = script.new(
    "display.clear() display.settext('Hello from myTest8')", "myTest8")
myTest8()
```

Creates a new script referenced by the variable `myTest8` with the name `myTest8`.
Runs the script. The instrument displays `Hello from myTest8`.

Example 2

```
autoexec = script.new(
    "display.clear() display.settext('Hello from autoexec')", 'autoexec')
```

Creates a new `autoexec` script that clears the display when the instrument is turned on and displays `Hello from autoexec`.

Also see

[Create a script using the script.new\(\) command](#) (on page 14-43)

[Global variables and the script.user.scripts table](#) (on page 14-42)

[Named scripts](#) (on page 14-4)

[scriptVar.save\(\)](#) (on page 15-187)

[script.newautorun\(\)](#) (on page 15-181)

script.newautorun()

This function creates a script and enables autorun.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.newautorun("code")
scriptVar = script.newautorun("code", "name")
```

<i>scriptVar</i>	The name of the variable that references the script
<i>code</i>	A string that contains the body of the script
<i>name</i>	The name of the script

Details

The *name* parameter is the name that is added to the `script.user.scripts` table. If *name* is not provided, an empty string is used, and the script is unnamed. If the name already exists in `script.user.scripts`, the *name* attribute of the existing script is set to an empty string before it is replaced by the new script.

Note that *name* is the value that is used for the instrument front panel display. If this value is not defined, the script is not available from the virtual front panel.

You must save the new script into nonvolatile memory to keep it when the instrument is turned off.

The script is run automatically immediately after it is created.

This command is the same as the `script.new()` function except that the script is automatically run.

Example

```
NewAuto = script.newautorun("print('Hello from new auto run command')", 'NewAuto')
print(NewAuto.autorun)
print(NewAuto.name)
```

Creates a new script called `NewAuto` that automatically has the `autorun` attribute set to `yes` after it is created. The *name* is set to `NewAuto`.

Output:

```
Hello from new auto run command
yes
NewAuto
```

Also see

[Create a script using the script.new\(\) command](#) (on page 14-43)

[Global variables and the script.user.scripts table](#) (on page 14-42)

[Named scripts](#) (on page 14-4)

[script.new\(\)](#) (on page 15-180)

[scriptVar.autorun](#) (on page 15-184)

[scriptVar.save\(\)](#) (on page 15-187)

script.restore()

This function restores a script that was removed from the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.restore(name)
```

<i>name</i>	The name of the script to be restored

Details

This command copies the script from nonvolatile memory into the runtime environment. It also creates a global variable with the same name as the name of the script.

Example

```
script.restore("test9")
```

Restores a script named `test9` from nonvolatile memory.

Also see

[script.delete\(\)](#) (on page 15-178)

script.run()

This function runs the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.run()
```

```
run()
```

Details

Each time the `script.run()` command is given, the anonymous script is executed. This script can be run using this command many times without having to re-send it.

Example

```
run()
```

Runs the anonymous script.

Also see

[script.anonymous](#) (on page 15-177)

script.user.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.user.catalog() do body end
```

<i>name</i>	String representing the name of the script
<i>body</i>	Code that implements the body of the <code>for</code> loop to process the names in the catalog

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.user.catalog() do
  print(name)
end
```

Retrieve the catalog listing for user scripts.

Also see

None

scriptVar.autorun

This attribute controls the autorun state of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	See Details	See Details

Usage

```
scriptVar.autorun = "state"  
state = scriptVar.autorun
```

<i>scriptVar</i>	The name of the variable that references the script
<i>state</i>	String that indicates whether or not the script runs automatically when powered on: ■ <i>yes</i>: Script runs automatically ■ <i>no</i>: Script does not run automatically

Details

Autorun scripts run automatically when the instrument is turned on. You can set any number of scripts to autorun.

The run order for autorun scripts is arbitrary, so make sure the run order is not important.

The default value for *scriptVar.autorun* depends on how the script was loaded. The default is *no* if the script was loaded with `loadscript` or `script.new()`. It is *yes* for scripts loaded with `loadandruncscript` or `script.newautorun()`.

NOTE

Make sure to save the script in nonvolatile memory after setting the *autorun* attribute so that the instrument keeps the setting.

Example

```
test5.autorun = "yes"  
test5.save()
```

Assume a script named `test5` is in the runtime environment.

The next time the instrument is turned on, `test5` script automatically loads and runs.

Also see

None

scriptVar.list()

This function generates a script listing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.list()
```

<i>scriptVar</i>	The name of the variable that references the script

Details

This function generates output in the form of a sequence of response messages (one message for each line of the script). It also generates output of the script control messages (`loadscript` or `loadandrunscript` and `endscript`).

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')",  
    "test7")  
test7()  
test7.save()  
test7.list()
```

The above example code creates a script named `test7` that displays text on the virtual front panel and lists the script with the following output:

```
loadscript test7  
display.clear() display.settext("Hello from my test")  
endscript
```

Also see

[Load a script by sending commands over the remote interface](#) (on page 14-4)

[Retrieve source code one line at a time](#) (on page 14-48)

scriptVar.name

This attribute contains the name of a script in the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not applicable	Not applicable

Usage

```
scriptVar.name = "scriptName"  
scriptName = scriptVar.name
```

<i>scriptVar</i>	Name of the variable that references the script
<i>scriptName</i>	A string that represents the name of the script

Details

When setting the script name, this attribute renames the script that the variable *scriptVar* references.

This attribute must be either a valid Lua identifier or the empty string. Changing the name of a script changes the index that is used to access the script in the `script.user.scripts` table. Setting the attribute to an empty string removes the script from the table completely, and the script becomes an unnamed script.

As long as there are variables referencing an unnamed script, the script can be accessed through those variables. When all variables that reference an unnamed script are removed, the script is removed from the runtime environment.

If the new name is the same as a name that is already used for another script, the name of the other script is set to an empty string, and that script becomes unnamed.

NOTE

Changing the name of a script does not change the name of any variables that reference that script. The variables still reference the script, but the names of the script and variables may not match.

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
test7()
print(test7.name)

test7.name = "test7"
print(test7.name)

test7.save()
```

This example calls the `script.new()` function to create a script with no name, runs the script, names the script `test7`, and then saves the script in nonvolatile memory.

Also see

[Rename a script](#) (on page 14-46)

[script.new\(\)](#) (on page 15-180)

[scriptVar.save\(\)](#) (on page 15-187)

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.run()
scriptVar()
```

<i>scriptVar</i>	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()`.

To run a factory script, use `script.factory.scripts.scriptName()`, replacing `scriptName` with the name of the factory script.

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")
```

<code>scriptVar</code>	The name of variable that references the script
<code>filename</code>	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 file name parameter is an empty string), the script is saved to internal nonvolatile memory. Only a script with `filename` defined can be saved to internal nonvolatile memory. If a `filename` is given, the script is saved to the USB flash drive.

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

[Save a user script](#) (on page 14-11)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW) (see Details)	No	Not applicable	Not saved	Not applicable

Usage

```
code = scriptVar.source
scriptVar.source = nil
```

<i>scriptVar</i>	The name of the variable that references the script that contains the source code
<i>code</i>	A string that contains the body of the script

Details

The `loadscript` or `loadandrunscript` and `endscript` keywords are not included in the source code.

The body of the script is a single string with lines separated by the new line character.

The instrument automatically stores the source for all scripts that are loaded on the instrument. To free up memory or to obfuscate the code, assign `nil` to the source attribute of the script. Although this attribute is writable, it can only be set to the `nil` value.

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
print(test7.source)
```

This example creates a script called `test7` that displays a message on the virtual front panel and retrieves the source code.

Output:

```
display.clear() display.settext('Hello from my test')
```

Also see

[scriptVar.list\(\)](#) (on page 15-185)

settime()

This function sets the real-time clock (sets present time of the system).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
settime(time)
```

<i>time</i>	The time in seconds since January 1, 1970 UTC

Details

This function sets the date and time of the instrument based on the *time* parameter (specified 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

```
systemTime = os.time({year = 2020,
    month = 3,
    day = 31,
    hour = 14,
    min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2020 at 2:25 pm.

Also see

[gettimezone\(\)](#) (on page 15-108)

[os.time\(\)](#) (on page 15-168)

[settimezone\(\)](#) (on page 15-189)

settimezone()

This function sets the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
settimezone(offset)
settimezone("offset", "dstOffset", "dstStart", "dstEnd")
```

<i>offset</i>	String representing offset from UTC
<i>dstOffset</i>	String representing the daylight savings offset from UTC
<i>dstStart</i>	String representing when daylight savings time starts
<i>dstEnd</i>	String representing when daylight savings time ends

Details

You only need to set the time zone if you use the `os.time()` and `os.date()` functions.

If only one parameter is given, the same time offset is used throughout the year. If four parameters are given, time is adjusted twice during the year for daylight savings time.

offset and *dstOffset* are strings of the form "[+|-]hh[:mm[:ss]]" that indicate how much time must be added to the local time to get UTC time:

- *hh* is a number between 0 and 23 that represents hours
- *mm* is a number between 0 and 59 that represents minutes
- *ss* is a number between 0 and 59 that represents seconds

The minute, second, +, and – fields are optional.

For example, to set the UTC-5 time zone, you specify the string "5", because UTC-5 is 5 hours behind UTC and you must add 5 hours to the local time to determine UTC time. To specify the time zone UTC4, you specify "-4", because UTC4 is 4 hours ahead of UTC and 4 hours must be subtracted from the local time to determine UTC.

dstStart and *dstEnd* are strings of the form "MM.w.dw/hh[:mm[:ss]]" that indicate when daylight savings time begins and ends respectively:

- *MM* is a number between 1 and 12 that represents the month
- *w* is a number between 1 and 5 that represents the week in the month
- *dw* is a number between 0 and 6 that represents the day of the week (where 0 is Sunday)

The rest of the fields represent the time of day that the change takes effect:

- *hh* represents hours
- *mm* represents minutes
- *ss* represents seconds

The minutes and seconds fields are optional.

The week of the month and day of the week fields are not specific dates.

Example

```
settimezone("8", "1", "3.3.0/02", "11.2.0/02")
settimezone(offset)
```

Sets *offset* to equal +8 hours, +1 hour for DST, starts on Mar 14 at 2:00 am, ends on Nov 7 at 2:00 am.
Sets local time zone to *offset*.

Also see

[gettimezone\(\)](#) (on page 15-108)

[os.time\(\)](#) (on page 15-168)

[settime\(\)](#) (on page 15-188)

setup.poweron

This attribute specifies which saved setup to recall when the instrument is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0

Usage

```
id = setup.poweron
setup.poweron = id
```

<i>id</i>	An integer that specifies the setup to recall when the instrument power is turned on (0 to 5)

Details

When *id* = 0, the instrument uses the factory default setup when it is turned on. When *id* is set to 1 to 5, it uses the setup saved with `setup.save()`.

Only setups stored in nonvolatile memory are available (you cannot recall a script from a USB flash drive with this command).

To save a script that is used when the instrument is powered on, you can create a configuration script and name it `autoexec`.

Example

```
setup.poweron = 0
```

Set the instrument to use the factory default setup when power is turned on.

Also see

[setup.save\(\)](#) (on page 15-192)

[Start-up \(power-on\) configuration](#) (on page 4-15)

setup.recall()

This function recalls settings from a saved setup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setup.recall(id)
```

id

An integer or string that specifies the location of the setup to recall:

- Factory default setup: 0
- User-saved setup in nonvolatile memory: 1 to 5
- User-saved setup on a USB flash drive: `"/path/filename"`

Details

When the *id* parameter is an integer (*n*), it is interpreted as the setup number to restore from the instrument's nonvolatile memory. When *n* = 0, the instrument recalls the factory default setup; when *n* = 1 to 5, the instrument recalls a user-saved setup.

When the *id* parameter is a string, it is interpreted as the path and file name of the setup to restore from a file on a USB flash drive. The path may be absolute or relative to the current working directory.

Before a setup is recalled, an instrument reset is performed.

Example 1

```
setup.recall(1)
```

Recall the user-saved setup at location 1.

Example 2

```
setup.recall("/usb1/KEITHLEY_30730.set")
```

Recall a user-saved setup stored in a file named KEITHLEY_30730 on a USB flash drive.

Also see

[Saved setups](#) (on page 4-13)
[setup.save\(\)](#) (on page 15-192)

setup.save()

This function saves the present setup as a user-saved setup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setup.save(id)
```

id

An integer or string specifying where to save the user setup:

- Save in nonvolatile memory (1 to 5)
- Save as user-saved setup on a USB flash drive ("*/path/filename*")

Details

When the *id* parameter is an integer (*n*), it is interpreted as the setup number to save to the nonvolatile memory of the instrument.

NOTE

When you save to a specified integer (1 to 5) in nonvolatile memory, the previous setup at that same location is overwritten.

When the *id* parameter is a string, it is interpreted as the path and file name of the location to save the present setup on a USB flash drive. The path may be absolute or relative to the current working directory.

Example

```
setup.save(5)
```

Saves the present setup to the internal memory of the instrument at location 5.

Also see

[Saved setups](#) (on page 4-13)
[setup.recall\(\)](#) (on page 15-191)

smuX.abort()

This function terminates all overlapped operations on the specified source-measure unit (SMU).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.abort()
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.abort()</code> applies to SMU channel A)

Details

The `smuX.abort()` function does not turn the output off or change any settings.

If this function is used to abort a sweep, when it is executed, the SMU exits its trigger model immediately and returns to the idle state of the trigger model.

Example

```
smua.abort()
```

 Terminates all overlapped operations on SMU channel A.

Also see

[smuX.measure.overlappedY\(\)](#) (on page 15-223)

[smuX.trigger.initiate\(\)](#) (on page 15-256)

smuX.buffer.getstats()

This function returns the statistics for a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
statistics = smuX.buffer.getstats(bufferVar)
```

<i>statistics</i>	The statistical data about the data in the reading buffer
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.buffer.getstats()</code> specifies SMU channel A)
<i>bufferVar</i>	The reading buffer to process

Details

This function returns a table with statistical data about the data that is placed in the buffer.

The SMU automatically updates reading buffer statistics as data is added to the reading buffer. When the reading buffer is configured to wrap around and overwrite older data with new data, the buffer statistics include the data that was overwritten.

The table returned from this function is a snapshot. Although the SMU continues to update the statistics, the table returned is not updated. To get fresh statistics, call this function again.

The *statistics* parameter has the attributes described in the following table.

Attribute	When returned	Description
n	Always	The number of data points on which the statistics are based
mean	When n > 0	The average of all readings added to the buffer
stddev	When n > 1	The standard deviation of all readings (samples) added to the buffer
min	When n > 0	A table containing data about the minimum reading value added to the buffer
max	When n > 0	A table containing data about the maximum reading value added to the buffer

If n equals zero (0), all other attributes are nil. If n equals 1, the stddev attribute is nil because the standard deviation of a sample size of 1 is undefined.

The min and max entries each have the attributes defined in the following table.

Attribute	Description
measurefunction	String indicating the function that was measured for the reading (current, voltage, ohms, or watts)
measurerange	The full-scale range value for the measurement range used when the measurement was made
reading	The reading value
sourcefunction	String indicating the source function at the time of the measurement (current or voltage)
sourceoutputstate	String indicating the state of the source (off or on)
sourcerange	Full-scale range value for the source range used when the measurement was made
sourcevalue	If <i>bufferVar.collectsourcevalues</i> is enabled, the sourced value in effect at the time of the reading
status	Status value for the reading; the status value is a floating-point number that encodes the status value into a floating-point value
timestamp	If <i>bufferVar.collecttimestamps</i> is enabled, the timestamp, in seconds, between when the reading was acquired and when the first reading in the buffer was acquired; adding this value to the base timestamp produces the actual time the measurement was acquired

Example

```

reset()
smua.nvbuffer1.clear()
smua.measure.count = 10
smua.measure.v(smua.nvbuffer1)
stats = smua.buffer.getstats(smua.nvbuffer1)
print("n= " .. stats.n)
print("mean= " .. stats.mean)
print("stddev= " .. stats.stddev)
print("min= " .. stats.min.reading)
print("max= " .. stats.max.reading)

```

Make measurements and store them in `nvbuffer1`. Print the statistics for the data.

Example output:

```

n= 10
mean= -2.3851394871599e-05
stddev= 4.406545187484e-07
min= -2.4557113647461e-05
max= -2.322196996829e-05

```

Also see

[smuX.buffer.recalculatestats\(\)](#) (on page 15-195)

smuX.buffer.recalculatestats()

This function recalculates the statistics of the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.buffer.recalculatestats(bufferVar)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.buffer.recalculatestats()</code> specifies SMU channel A)
<i>bufferVar</i>	The reading buffer to process

Details

This function causes the SMU to regenerate the reading buffer statistics about the specified reading buffer. Because the SMU automatically updates reading buffer statistics when data is added to the reading buffer, this function is generally not needed. When the reading buffer is configured to wrap around and overwrite older data with new data, the buffer statistics include the data that was overwritten. Use this function to recalculate the statistics that include only the data that is presently stored in the buffer.

Example

```

smua.buffer.recalculatestats(smua.nvbuffer1)
Recalculates the statistics of buffer smua.nvbuffer1.

```

Also see

[bufferVar.fillmode](#) (on page 15-25)

[smuX.buffer.getstats\(\)](#) (on page 15-193)

smuX.cal.adjustdate

This attribute stores the date of the last calibration adjustment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU cal. restore	SMU nonvolatile memory	Initially set to factory calibration date

Usage

```
adjustDate = smuX.cal.adjustdate
smuX.cal.adjustdate = adjustDate
```

<i>adjustDate</i>	Date of the last adjustment
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.cal.adjustdate</code> applies to SMU channel A)

Details

This attribute stores the adjustment date associated with the active calibration set. The adjustment date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock()` function.

You cannot change the adjustment date without first making a change to the calibration constants.

Once you change any calibration constants, you must set the adjustment date before you can save the calibration data to the nonvolatile memory of the SMU.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute reflects the date stored with that set.

`smuX.cal.adjustdate` must be set to the date the adjustment was done using the UTC time and date. The date is stored as the number of seconds since UTC, 12:00 am Jan 1, 1970.

Due to the internal storage format, `smuX.cal.adjustdate` is only accurate to within a few minutes of the value set.

Example

<code>smua.cal.adjustdate = os.time()</code>	Sets the adjustment date for SMU channel A to the current time set on the instrument.

Also see

[Adjustment](#) (on page 17-12)
[os.time\(\)](#) (on page 15-168)
[smuX.cal.date](#) (on page 15-197)
[smuX.cal.due](#) (on page 15-198)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.save\(\)](#) (on page 15-202)
[smuX.cal.state](#) (on page 15-203)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.date

This attribute stores the calibration date of the active calibration set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU calibration restore	SMU nonvolatile memory	Initially set to factory calibration date

Usage

```
calDate = smuX.cal.date
smuX.cal.date = calDate
```

<i>calDate</i>	The calibration date of the active calibration set
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.cal.date</code> applies to SMU channel A)

Details

This attribute stores the calibration date that is associated with the active calibration set. The calibration date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock()` function. It is typically set to the date when the instrument was calibrated.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute reflects the date stored with that set.

`smuX.cal.date` must be set to the date the calibration was done using the UTC time and date. The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.

Due to the internal storage format, `smuX.cal.date` is accurate to within a few minutes of the value set.

Example

<code>smua.cal.date = os.time()</code>	Sets calibration date for SMU channel A to the present time set on the instrument.

Also see

[Adjustment](#) (on page 17-12)
[os.time\(\)](#) (on page 15-168)
[smuX.cal.adjustdate](#) (on page 15-196)
[smuX.cal.due](#) (on page 15-198)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.save\(\)](#) (on page 15-202)
[smuX.cal.state](#) (on page 15-203)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.due

This attribute stores the calibration due date for the next calibration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU cal. restore	SMU nonvolatile memory	0

Usage

```
calDue = smuX.cal.due
smuX.cal.due = calDue
```

<i>calDue</i>	Due date of next calibration (0 indicates that no date is set)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.cal.due</code> applies to SMU channel A)

Details

This attribute stores the calibration due date associated with the active calibration set. The calibration due date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock()` function. It is typically set to the date when the next calibration should be performed.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute reflects the due date stored with that set.

`smuX.cal.due` must be set to the date the next calibration is required using the UTC time and date. The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.

Due to the internal storage format, `smuX.cal.due` is only accurate to within a few minutes of the value set.

Example

```
smua.cal.due = os.time() + 365 * 24 * 60 * 60
```

Sets the SMU channel A calibration due date equal to one year from the present time set on the instrument.

Also see

[Adjustment](#) (on page 17-12)
[os.time\(\)](#) (on page 15-168)
[smuX.cal.adjustdate](#) (on page 15-196)
[smuX.cal.date](#) (on page 15-197)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.state](#) (on page 15-203)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.lock()

This function disables the commands that change calibration settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.lock()
```

X	Source-measure unit (SMU) channel (for example, <code>smua.cal.lock()</code> specifies SMU channel A)

Details

Before you can lock calibration, the calibration constants must be written to nonvolatile memory or a previous calibration set must be restored. Error code 5012, "Cal data not saved - save or restore before lock," results if this function is called when the calibration state is `smuX.CALSTATE_CALIBRATING`.

Example

<code>smua.cal.lock()</code>	Disables calibration functions for SMU channel A.

Also see

[Adjustment](#) (on page 17-12)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.save\(\)](#) (on page 15-202)
[smuX.cal.state](#) (on page 15-203)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.password

This attribute stores the password required to enable calibration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	Not applicable	SMU nonvolatile memory	"KI0026XX"

Usage

```
smuX.cal.password = "newPassword"
```

X	SMU channel (for example, <code>smua.cal.password</code> applies to SMU channel A)
<i>newPassword</i>	A string that contains the new password

Details

A new password can only be assigned when calibration has been unlocked.
 The calibration password is write-only and cannot be read.

Example

<code>smua.cal.password = "LetMeIn"</code>	Assigns a new calibration password for SMU channel A.

Also see

[Adjustment](#) (on page 17-12)

[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.polarity

This attribute controls which calibration constants are used for all subsequent measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset SMU cal. lock Recall setup	Not saved	0 (smuX.CAL_AUTO)

Usage

```
calPolarity = smuX.cal.polarity
smuX.cal.polarity = calPolarity
```

<i>calPolarity</i>	The polarity to use for measurements. Set to one of the following values: 0 or smuX.CAL_AUTO: Automatic polarity detection 1 or smuX.CAL_POSITIVE: Measure with positive polarity calibration constants 2 or smuX.CAL_NEGATIVE: Measure with negative polarity calibration constants
<i>X</i>	SMU channel (for example, smua.cal.polarity applies to SMU channel A)

Details

This attribute controls which polarity calibration constants are used to make all subsequent measurements.

This attribute does not affect the `smuX.measure.calibrateY()` command. The polarity for `smuX.measure.calibrateY()` is dictated by the range parameter that is defined for it. The measurement calibration commands require the measurements provided to have been made using the polarity being calibrated.

When making measurements for calibration points far away from zero, the desired polarity constants are inherently used. However, when making measurements near zero, it is possible that the instrument could use the calibration constants from the wrong polarity. Setting `smuX.cal.polarity` to positive or negative forces measurements to be made using the calibration constants for a given polarity, rather than basing the choice on the raw measurement data.

This attribute can only be set to positive or negative when calibration is unlocked. This attribute is automatically set to `smuX.CAL_AUTO` when calibration is locked.

Example

```
smua.cal.polarity = smua.CAL_POSITIVE
```

Selects positive calibration constants for all subsequent measurements on SMU channel A.

Also see

[Adjustment](#) (on page 17-12)
[reset\(\)](#) (on page 15-176)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.unlock\(\)](#) (on page 15-204)
[smuX.measure.calibrateY\(\)](#) (on page 15-214)
[smuX.reset\(\)](#) (on page 15-231)
[smuX.source.calibrateY\(\)](#) (on page 15-234)

smuX.cal.restore()

This function loads a stored set of calibration constants.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.restore()
smuX.cal.restore(calset)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smuA.cal.restore()</code> applies to SMU channel A)
<i>calset</i>	The calibration set to be loaded; set <i>calset</i> to one of the following values: ■ 0 or <code>smuX.CALSET_NOMINAL</code>: A set of calibration constants that are uncalibrated, but set to nominal values to allow rudimentary functioning of the instrument ■ 1 or <code>smuX.CALSET_FACTORY</code>: The calibration constants when the instrument left the factory ■ 2 or <code>smuX.CALSET_DEFAULT</code>: The normal calibration set ■ 3 or <code>smuX.CALSET_PREVIOUS</code>: The calibration set that was used before the last default set was overwritten

Details

This function overwrites the present set of calibration constants with constants read from nonvolatile memory.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

If *calset* is not specified, `smuX.CALSET_DEFAULT` is used.

Example

<code>smuA.cal.restore()</code>	Restores factory calibration constants for SMU channel A.

Also see

[Adjustment](#) (on page 17-12)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.save()

This function stores the active calibration constants to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.save()
```

X	Source-measure unit (SMU) channel (for example, <code>smua.cal.save()</code> applies to SMU channel A)

Details

This function stores the active set of calibration constants to nonvolatile memory. The previous calibration constants (from the default calibration set) are copied to the previous calibration set (`smuX.CALSET_PREVIOUS`) before overwriting the default calibration set.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

If any of the calibration constants have been changed, this function is disabled unless the calibration date, the calibration due date, and the calibration adjust date have been assigned new values.

Example

<code>smua.cal.save()</code>	Stores calibration constants for SMU channel A in nonvolatile memory.

Also see

[Adjustment](#) (on page 17-12)
[smuX.cal.adjustdate](#) (on page 15-196)
[smuX.cal.date](#) (on page 15-197)
[smuX.cal.due](#) (on page 15-198)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.state

This attribute returns the present calibration state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
calState = smuX.cal.state
```

<i>calState</i>	The present calibration state; when reading this attribute, <i>calState</i> has one of the following values: 0 or <code>smuX.CALSTATE_LOCKED</code>: Calibration is locked 1 or <code>smuX.CALSTATE_CALIBRATING</code>: The calibration constants or dates have been changed but not yet saved to nonvolatile memory 2 or <code>smuX.CALSTATE_UNLOCKED</code>: Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.cal.state</code> applies to SMU channel A)

Details

This read-only attribute indicates the calibration state of the instrument: Locked, calibrating, or unlocked.

Example

<pre>calstate = smua.cal.state print(calstate)</pre>	Reads calibration state for SMU channel A. Output: 0.000000e+00 The above output indicates that calibration is locked.

Also see

[Adjustment](#) (on page 17-12)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.save\(\)](#) (on page 15-202)
[smuX.cal.unlock\(\)](#) (on page 15-204)

smuX.cal.unlock()

This function enables the commands that change calibration settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.unlock(password)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.cal.unlock()</code> applies to SMU channel A)
<i>password</i>	Calibration password

Details

This function enables the calibration functions to change the calibration settings.

The password when the instrument is shipped from the factory is "KI0026XX".

Example

```
smua.cal.unlock("KI0026XX")
```

Unlocks calibration for SMU channel A.

Also see

[Adjustment](#) (on page 17-12)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.password](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.state](#) (on page 15-203)

smuX.contact.calibratehi()

This function adjusts the high/sense high contact check measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.contact.calibratehi(cp1Measured, cp1Reference, cp2Measured, cp2Reference)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.contact.calibratehi()</code> applies to SMU channel A)
<i>cp1Measured</i>	The value measured by this SMU for point 1
<i>cp1Reference</i>	The reference measurement for point 1 as measured externally
<i>cp2Measured</i>	The value measured by this SMU for point 2
<i>cp2Reference</i>	The reference measurement for point 2 as measured externally

Details

Contact check measurement calibration does not require range information.

Typically, points one and two are near 0 Ω and 50 Ω , respectively.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the calibration set that is active. If not, corruption of the calibration constants may result.

The new calibration constants are activated immediately but are not written to nonvolatile memory. Use `smuX.cal.save()` to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

Example

```
-- Short SENSE LO and LO terminals.
-- Short SENSE HI and HI terminals.
-- Allow readings to settle, then get measurements.
r0_hi, r0_lo = smua.contact.r()

-- Connect 50 OHM resistor between SENSE LO and LO.
-- Connect 50 OHM resistor between SENSE HI and HI.
-- Allow readings to settle, then get measurements.
r50_hi, r50_lo = smua.contact.r()
smua.contact.calibratelo(r0_lo, Z_actual_lo, r50_lo, 50_ohm_actual_lo)
smua.contact.calibratehi(r0_hi, Z_actual_hi, r50_hi, 50_ohm_actual_hi)
```

The instrument performs a contact check.

Install and measure two resistors.

The user sends the contact check LO calibration command.

The user sends the contact check HI calibration command.

Also see

[Adjustment](#) (on page 17-12)

[smuX.cal.restore\(\)](#) (on page 15-201)

[smuX.cal.save\(\)](#) (on page 15-202)

[smuX.cal.unlock\(\)](#) (on page 15-204)

[smuX.contact.calibratelo\(\)](#) (on page 15-206)

smuX.contact.calibratelo()

This function adjusts the low/sense low contact check measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.contact.calibratelo(cp1Measured, cp1Reference, cp2Measured, cp2Reference)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.contact.calibratelo()</code> applies to SMU channel A)
<i>cp1Measured</i>	The value measured by this SMU for point 1
<i>cp1Reference</i>	The reference measurement for point 1 as measured externally
<i>cp2Measured</i>	The value measured by this SMU for point 2
<i>cp2Reference</i>	The reference measurement for point 2 as measured externally

Details

Contact check measurement adjustment does not require range information.

Typically, points one and two are near 0 Ω and 50 Ω , respectively.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the active calibration set. If not, corruption of the calibration constants may result.

The new calibration constants are activated immediately but are not written to nonvolatile memory. Use `smuX.cal.save()` to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

Example

```
-- Short SENSE LO and LO terminals.
-- Short SENSE HI and HI terminals.
-- Allow readings to settle, then get measurements.
r0_hi, r0_lo = smua.contact.r()
-- Connect 50 OHM resistor between SENSE LO and LO.
-- Connect 50 OHM resistor between SENSE HI and HI.
-- Allow readings to settle, then get measurements.
r50_hi, r50_lo = smua.contact.r()
smua.contact.calibratelo(r0_lo, Z_actual_lo, r50_lo, 50_ohm_actual_lo)
smua.contact.calibratehi(r0_hi, Z_actual_hi, r50_hi, 50_ohm_actual_hi)
```

The instrument performs a contact check.

Install and measure two resistors.

The user sends the contact check LO calibration command.

The user sends the contact check HI calibration command.

Also see

[Adjustment](#) (on page 17-12)

[smuX.cal.restore\(\)](#) (on page 15-201)

[smuX.cal.save\(\)](#) (on page 15-202)

[smuX.cal.unlock\(\)](#) (on page 15-204)

[smuX.contact.calibratehi\(\)](#) (on page 15-204)

smuX.contact.check()

This function determines if contact resistance is lower than the threshold.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.contact.check()
```

X	Source-measure unit (SMU) channel (for example, <code>smua.contact.check()</code> applies to SMU channel A)

Details

This function returns `true` if the contact resistance is below the threshold; this function returns `false` if it is above the threshold. The threshold value is set by the `smuX.contact.threshold` attribute.

An error is generated when the output is on and:

- SMU is a current source with current range set to less than 1 mA (error code 5065, "I range too low for contact check")
- SMU is a voltage source with current limit set to less than 1 mA (error code 5050, "I limit too low for contact check")

An error is generated when the output is off and:

- The output off mode is High-Z (error code 5048, "Contact check not valid with HIGH-Z OUTPUT off")
- The output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCVOLTS` and the off current limit set to less than 1 mA (error code 5066, "source.offlimit too low for contact check")
- The output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCAMPS` and the source range is less than 1 mA (error code 5065, "I range too low for contact check")

Example

```
if not smua.contact.check() then
  -- take action
end
```

Takes action if contact check on SMU channel A fails.

Also see

[Contact check measurements](#) (on page 5-21)
[smuX.contact.speed](#) (on page 15-209)
[smuX.contact.threshold](#) (on page 15-210)
[smuX.source.offfunc](#) (on page 15-241)

smuX.contact.r()

This function measures aggregate contact resistance.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
rhi, rlo = smuX.contact.r()
```

<i>rhi</i>	The measured aggregate contact resistance on the HI/sense HI side
<i>rlo</i>	The measured aggregate contact resistance on the LO/sense LO side
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.contact.r()</code> applies to SMU channel A)

Details

If you attempt to perform a contact resistance measurement when any of the following conditions exist, an error is generated.

- When the output is on and SMU is a current source with current range set to less than 1 mA
- When the output is on and SMU is a voltage source with current limit set to less than 1 mA
- When the output is off and the output off mode is High-Z
- When the output is off and the output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCVOLTS` and the off current limit set to less than 1 mA
- When the output is off and the output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCAMPS` and the source range is less than 1 mA

Example

```
if not smua.contact.check() then
    smua.contact.speed = smua.CONTACT_SLOW
    rhi, rlo = smua.contact.r()
    print(rhi, rlo)
    exit()
end
```

Check contacts against threshold.
Set speed for SMU channel A to slow.
Get resistance readings.
Output contact resistances to the host.
Terminate execution.

Also see

[Contact check measurements](#) (on page 5-21)
[smuX.contact.check\(\)](#) (on page 15-207)
[smuX.contact.speed](#) (on page 15-209)

smuX.contact.speed

This attribute stores the speed setting for contact check measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	0 (smuX.CONTACT_FAST)

Usage

```
speedSetting = smuX.contact.speed  
smuX.contact.speed = speedSetting
```

<i>speedSetting</i>	The speed setting. Set to one of the following: 0 or smuX.CONTACT_FAST 1 or smuX.CONTACT_MEDIUM 2 or smuX.CONTACT_SLOW
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.contact.speed applies to SMU channel A)

Details

This setting controls the aperture of measurements made for contact check. It does not affect the smuX.measure.nplc aperture setting.

The speed setting can have a dramatic effect on the accuracy of the measurement (see specifications).

Example

```
smua.contact.speed = smua.CONTACT_SLOW
```

Configure contact check for higher accuracy on SMU channel A.

Also see

[Contact check measurements](#) (on page 5-21)
[reset\(\)](#) (on page 15-176)
[smuX.contact.check\(\)](#) (on page 15-207)
[smuX.contact.r\(\)](#) (on page 15-208)
[smuX.reset\(\)](#) (on page 15-231)

smuX.contact.threshold

This attribute stores the resistance threshold for the `smuX.contact.check()` function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	50 (50 Ω)

Usage

```
rValue = smuX.contact.threshold
smuX.contact.threshold = rValue
```

<i>rValue</i>	The resistance above which contact check fails (measured in ohms)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.contact.threshold</code> applies to SMU channel A)

Details

Set the threshold to less than 1 k Ω .

Example

```
smua.contact.threshold = 5
```

Set the contact check threshold for SMU channel A to 5 Ω .

Also see

[Contact check measurements](#) (on page 5-21)
[reset\(\)](#) (on page 15-176)
[smuX.contact.check\(\)](#) (on page 15-207)
[smuX.reset\(\)](#) (on page 15-231)

smuX.makebuffer()

This function creates a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar = smuX.makebuffer(bufferSize)
```

<i>bufferVar</i>	The created reading buffer
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.makebuffer()</code> applies to SMU channel A)
<i>bufferSize</i>	Maximum number of readings that can be stored

Details

You can use this function to create and dynamically allocate reading buffers. Use *bufferSize* to designate the number of readings the buffer can store.

You can use dynamically allocated reading buffers interchangeably with the `smuX.nvbufferY` buffers.

To delete a reading buffer, set all references to the reading buffer equal to `nil`, then run the garbage collector (see the `collectgarbage()` function in [Standard libraries](#) (on page 14-28)).

Example

<code>mybuffer2 = smua.makebuffer(200)</code>	Creates a 200 element reading buffer (<code>mybuffer2</code>) for SMU channel A.

Also see

`collectgarbage()` in [Base library functions](#) (on page 14-29)

[Remote reading buffer programming](#) (on page 6-6)

[savebuffer\(\)](#) (on page 15-176)

[smuX.nvbufferY](#) (on page 15-230)

smuX.measure.autorangeY

This attribute stores the measurement autorange setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1 (<code>smuX.AUTORANGE_ON</code>)

Usage

```
autoRange = smuX.measure.autorangeY
smuX.measure.autorangeY = autoRange
```

<i>autoRange</i>	The state of the measurement autorange setting; set to one of the following values: 0 or <code>smuX.AUTORANGE_OFF</code>: Disabled 1 or <code>smuX.AUTORANGE_ON</code>: Enabled 2 or <code>smuX.AUTORANGE_FOLLOW_LIMIT</code>: Measure range automatically set to the limit range
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.autorangev</code> applies to SMU channel A)
<i>Y</i>	SMU measure function (<i>v</i> = voltage, <i>i</i> = current)

Details

This attribute indicates the measurement autorange state. Its value is `smuX.AUTORANGE_OFF` when the SMU measure circuit is on a fixed range and `smuX.AUTORANGE_ON` when it is in autorange mode.

Setting this attribute to `smuX.AUTORANGE_OFF` puts the SMU on a fixed range. The fixed range is the present SMU measure range.

Setting this attribute to `smuX.AUTORANGE_ON` puts the SMU measure circuit in autorange mode. It remains on its present measure range until the next measurement is requested.

If source high capacitance mode is enabled, current autorange is set to `smuX.AUTORANGE_FOLLOW_LIMIT` and cannot be changed.

Example

```
smua.measure.autorangev = smua.AUTORANGE_ON
```

Enables voltage measurement autoranging for SMU channel A.

Also see

[Autoranging](#) (on page 5-30)
[Range](#) (on page 5-29)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.rangeY](#) (on page 15-224)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.autozero

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	Instrument reset SMU reset Recall setup	Saved setup	2 (<code>smuX.AUTOZERO_AUTO</code>)

Usage

```
azMode = smuX.measure.autozero
smuX.measure.autozero = azMode
```

<i>azMode</i>	Indicates status of autozero; set to one of the following values: 0 or <code>smuX.AUTOZERO_OFF</code>: Autozero disabled 1 or <code>smuX.AUTOZERO_ONCE</code>: Performs autozero once, then disables autozero 2 or <code>smuX.AUTOZERO_AUTO</code>: Automatic checking of reference and zero measurements; an autozero is performed when needed
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.autozero</code> applies to SMU channel A)

Details

The analog-to-digital converter (ADC) uses a ratiometric A/D conversion technique. To ensure the accuracy of readings, the instrument must periodically obtain new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture being used for measurements. The 2606B uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made. If the reference measurements have expired when a signal measurement is made, the instrument automatically takes two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes more time than normal.

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 in these situations, you can use the `smuX.measure.autozero` attribute to disable the automatic reference measurements.

Disabling automatic reference measurements may allow the instrument to gradually drift out of specification. To minimize the drift, make a reference and zero measurement immediately before any critical test sequences. You can use the `smuX.AUTOZERO_ONCE` setting to force a refresh of the reference and zero measurements that are used for the present aperture setting.

The 2606B stores the reference measurements for the last ten NPLC settings that were used in a reference cache. If an NPLC setting is selected and an entry for it is not in the cache, the oldest (least recently used) entry is discarded to make room for the new entry.

Example

```
smua.measure.autozero = smua.AUTOZERO_ONCE
```

Performs autozero once for SMU channel A.

Also see

[Autozero](#) (on page 5-10)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.nplc](#) (on page 15-223)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.calibrateY()

This function generates and activates new measurement calibration constants.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.measure.calibrateY(range, cp1Measured, cp1Reference, cp2Measured, cp2Reference)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.calibratev()</code> applies to SMU channel A)
<i>Y</i>	SMU measurement function (<i>v</i> = voltage, <i>i</i> = current)
<i>range</i>	The measurement range to adjust
<i>cp1Measured</i>	The value measured by this SMU for point 1
<i>cp1Reference</i>	The reference measurement for point 1 as measured externally
<i>cp2Measured</i>	The value measured by this SMU for point 2
<i>cp2Reference</i>	The reference measurement for point 2 as measured externally

Details

This function generates and activates new calibration constants for the given range.

The positive and negative polarities of the instrument must be adjusted separately. Use a positive value for the *range* parameter to adjust the positive polarity and a negative value for the *range* parameter to adjust the negative polarity.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the calibration set that is active. Corruption of the calibration constants may result if this is ignored.

The new calibration constants are activated immediately but they are not written to nonvolatile memory. Use the `smuX.cal.save()` function to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is only available when calibration is unlocked using `smuX.cal.unlock()`.

Example

```
smua.measure.calibratev(1, 1e-4, 1e-5, 0.92, 0.903)
```

Adjust SMU channel A voltage measurement using the following values:

- 1 V calibration range
- 1e-4 for +zero measurement reading
- 1e-5 for +zero DMM measurement reading
- 0.92 for +FS measurement reading
- 0.903 for the +FS DMM measurement reading

Also see

[Adjustment](#) (on page 17-12)
[smuX.cal.lock\(\)](#) (on page 15-199)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.save\(\)](#) (on page 15-202)
[smuX.cal.unlock\(\)](#) (on page 15-204)
[smuX.source.calibrateY\(\)](#) (on page 15-234)

smuX.measure.count

This attribute sets the number of measurements made when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1

Usage

```
count = smuX.measure.count
smuX.measure.count = count
```

<i>count</i>	Number of measurements
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.count</code> applies to SMU channel A)

Details

This attribute controls the number of measurements made any time a measurement is requested. When using a reading buffer with a measure command, this attribute also controls the number of readings to be stored.

If the count is set to a value greater than 1, any measurement delay set by `smuX.measure.delay` only occurs before the first measurement, while the `smuX.measure.interval` controls the interval between successive measurements.

Example

<code>smua.measure.count = 10</code>	Sets the SMU channel A measure count to 10.

Also see

[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.delay](#) (on page 15-216)
[smuX.measure.interval](#) (on page 15-221)
[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.delay

This attribute controls the measurement delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	-1 (smuX.DELAY_AUTO)

Usage

```
mDelay = smuX.measure.delay
smuX.measure.delay = mDelay
```

<i>mDelay</i>	Set to the measurement delay value in seconds (for example, to specify an additional 10 ms measurement delay, set the value to 0.010) You can also set it to one of the following values: 0 or smuX.DELAY_OFF: No delay -1 or smuX.DELAY_AUTO: Automatic delay value
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.measure.delay applies to SMU channel A)

Details

This attribute allows for additional delay (settling time) before making a measurement. If you define the value instead of using the automatic delay value, the delay you set is used regardless of the range.

The smuX.DELAY_AUTO setting causes a current range-dependent delay to be inserted when a current measurement is requested. This happens when a current measurement command is executed, when the measure action is being performed in a sweep, or after changing ranges during an autoranged measurement.

If smuX.measure.count is greater than 1, the measurement delay is only inserted before the first measurement.

Example

smua.measure.delay = 0.010	Sets a 10 ms measurement delay for SMU channel A.

Also see

[Measure autodelay](#) (on page 5-29)
[reset\(\)](#) (on page 15-176)
[smuX.measure.count](#) (on page 15-215)
[smuX.measure.delayfactor](#) (on page 15-217)
[smuX.source.delay](#) (on page 15-236)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.delayfactor

This attribute stores a multiplier to the delays that are used when `smuX.measure.delay` is set to `smuX.DELAY_AUTO`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	1

Usage

```
delayFactor = smuX.measure.delayfactor
smuX.measure.delayfactor = delayFactor
```

<i>delayFactor</i>	The delay factor multiplier
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.delayfactor</code> applies to SMU channel A)

Details

The delay factor is only applied when `smuX.measure.delay = smuX.DELAY_AUTO`.

This attribute can be set to a value less than 1 (for example, 0.5) to decrease the automatic delay.

This attribute can be set to a value greater than 1 (for example, 1.5 or 2.0) to increase the automatic delay.

Setting this attribute to zero disables delays when `smuX.measure.delay = smuX.DELAY_AUTO`.

Example

```
smua.measure.delayfactor = 2.0
```

Doubles the measure delay for SMU channel A.

Also see

[Measure autodelay](#) (on page 5-29)
[reset\(\)](#) (on page 15-176)
[smuX.measure.delay](#) (on page 15-216)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.filter.count

This command sets the number of measured readings that are required to yield one filtered measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1

Usage

```
filterCount = smuX.measure.filter.count
smuX.measure.filter.count = filterCount
```

<i>filterCount</i>	The number of readings required for each filtered measurement (1 to 100)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.filter.count</code> applies to SMU channel A)

Details

This attribute sets the size of the stack used for filtered measurements.

Example

```
smua.measure.filter.count = 10
smua.measure.filter.type = smua.FILTER_MOVING_AVG
smua.measure.filter.enable = smua.FILTER_ON
```

Sets the filter count for SMU channel A to 10.
Sets the filter type to moving average.
Enables the filter.

Also see

[Filters](#) (on page 5-49)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.filter.enable](#) (on page 15-218)
[smuX.measure.filter.type](#) (on page 15-219)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.filter.enable

This command enables or disables filtered measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	0 (smuX.FILTER_OFF)

Usage

```
filterState = smuX.measure.filter.enable
smuX.measure.filter.enable = filterState
```

<i>filterState</i>	The filter status; set to one of the following values: 0 or smuX.FILTER_OFF: Disables the filter 1 or smuX.FILTER_ON: Enables the filter
<i>X</i>	SMU channel (for example, smua.measure.filter.enable applies to SMU channel A)

Details

This command enables or disables the filter.

Example

```
smua.measure.filter.count = 10
smua.measure.filter.type = smua.FILTER_MOVING_AVG
smua.measure.filter.enable = smua.FILTER_ON
```

Sets the filter count for SMU channel A to 10.
Sets the filter type to moving average.
Enables the filter.

Also see

[Filters](#) (on page 5-49)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.filter.count](#) (on page 15-217)
[smuX.measure.filter.type](#) (on page 15-219)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.filter.type

This command sets the type of filter used for measurements when the measurement filter is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1 (smuX.FILTER_REPEAT_AVG)

Usage

```

filterType = smuX.measure.filter.type
smuX.measure.filter.type = filterType

```

<i>filterType</i>	The filter type to use when filtering is enabled. Set to one of the following values: 0 or <code>smuX.FILTER_MOVING_AVG</code>: Selects the moving filter 1 or <code>smuX.FILTER_REPEAT_AVG</code>: Selects the repeat filter 2 or <code>smuX.FILTER_MEDIAN</code>: Selects the median filter
<i>X</i>	SMU channel (for example, <code>smua.measure.filter.type</code> applies to SMU Channel A)

Details

The 2606B provides a moving average, repeating average, and median filter type.

For the repeating filter, the stack (filter count) is filled, and the conversions are averaged to yield a reading. The stack is then cleared, and the process starts over.

The moving average filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the measurement conversions are averaged, yielding a reading. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The stack is re-averaged, yielding a new reading.

The median filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the reading nearest to the middle is returned. For each subsequent conversion placed into the stack, the oldest reading is discarded. The stack is then re-sorted, yielding a new reading. If the filter count is an even number, the reading returned is the average of the two middle readings.

Example

```

smua.measure.filter.count = 10
smua.measure.filter.type = smua.FILTER_MOVING_AVG
smua.measure.filter.enable = smua.FILTER_ON

```

Sets the filter count for SMU channel A to 10.
 Sets the filter type to moving average.
 Enables the filter.

Also see

[Filters](#) (on page 5-49)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.filter.count](#) (on page 15-217)
[smuX.measure.filter.enable](#) (on page 15-218)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.highcrangedelayfactor

This attribute contains a delay multiplier that is only used during range changes when the high-capacitance mode is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	10

Usage

```
delayFactor = smuX.measure.highcrangedelayfactor  
smuX.measure.highcrangedelayfactor = delayFactor
```

<i>delayFactor</i>	The delay factor; set to a value between 1 and 99
<i>X</i>	SMU channel (for example, <code>smua.measure.highcrangedelayfactor</code> applies to SMU Channel A)

Details

This delay multiplier is only active when the high-capacitance mode is active.

Example

<code>smua.measure.highcrangedelayfactor = 5</code>	Increases the delay used during range changes for SMU channel A by a factor of 5.

Also see

[High-capacitance mode](#) (on page 5-52)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.reset\(\)](#) (on page 15-231)
[smuX.source.highc](#) (on page 15-237)

smuX.measure.interval

This attribute sets the interval between multiple measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	0 (0 s)

Usage

```
interval = smuX.measure.interval  
smuX.measure.interval = interval
```

<i>interval</i>	The interval value (in seconds); set to a value between 0 and 1
<i>X</i>	SMU channel (for example, <code>smua.measure.interval</code> applies to SMU Channel A)

Details

This attribute sets the time interval between measurements when `smuX.measure.count` is set to a value greater than 1. The SMU attempts to start each measurement when scheduled. If the SMU cannot keep up with the interval setting, measurements are made as quickly as possible.

If filtered measurements are being made, the time interval is from the start of the first measurement for the filtered reading to the first measurement for a subsequent filtered reading. Extra measurements made to satisfy a filtered reading are not paced by this interval.

Example

```
smua.measure.interval = 0.5
```

Sets the measure interval for SMU channel A to 0.5 s.

Also see

[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.count](#) (on page 15-215)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.lowrangeY

This attribute sets the lowest measurement range that is used when the instrument is autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	Current: 100e-9 (100 nA) Voltage: 100e-3 (100 mV)

Usage

```
lowRange = smuX.measure.lowrangeY
smuX.measure.lowrangeY = lowRange
```

<i>lowRange</i>	The lowest voltage or current measurement range used during autoranging
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.lowrangev</code> applies to SMU channel A)
<i>Y</i>	SMU measure function (<i>v</i> = voltage, <i>i</i> = current)

Details

This attribute is used with autoranging to put a lower bound on the range used. Since lower ranges generally require greater settling times, setting a lowest range limit might make measurements require less settling time.

If the instrument is set to autorange and it is on a range lower than the one specified, the range is changed to the *lowRange* range value.

Example

<code>smua.measure.lowrangev = 1</code>	Sets voltage low range for SMU channel A to 1 V.

Also see

[Range](#) (on page 5-29)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.autorangeY](#) (on page 15-211)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.nplc

This command sets the integration aperture for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1.0

Usage

```
nplc = smuX.measure.nplc
smuX.measure.nplc = nplc
```

<i>nplc</i>	The integration aperture; set from 0.001 to 25
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.nplc</code> applies to SMU channel A)

Details

This attribute controls the integration aperture for the analog-to-digital converter (ADC).

The integration aperture is based on the number of power line cycles (NPLC), where 1 PLC for 60 Hz is 16.67 ms (1/60) and 1 PLC for 50 Hz is 20 ms (1/50).

Example

```
smua.measure.nplc = 0.5
```

Sets the integration time for SMU channel A to 0.5.

Also see

[reset\(\)](#) (on page 15-176)

[setup.recall\(\)](#) (on page 15-191)

[smuX.reset\(\)](#) (on page 15-231)

[Speed](#) (on page 5-33)

smuX.measure.overlappedY()

This function starts an asynchronous (background) measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
smuX.measure.overlappedY(rbuffer)
smuX.measure.overlappeddiv(ibuffer, vbuffer)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.overlappedv()</code> applies to SMU channel A)
<i>Y</i>	SMU measurement type (<i>v</i> = voltage, <i>i</i> = current, <i>r</i> = resistance, <i>p</i> = power)
<i>rbuffer</i>	A reading buffer object where the readings are stored
<i>ibuffer</i>	A reading buffer object where current readings are stored
<i>vbuffer</i>	A reading buffer object where voltage readings are stored

Details

This function starts a measurement and returns immediately. The measurements, as they are performed, are stored in a reading buffer (along with any other information that is being acquired). If the instrument is configured to return multiple readings where one is requested, the readings are available as they are made. Measurements are in the following units of measure: *v* = volts, *i* = amperes, *r* = ohms, *p* = watts.

The second form of this function, `smuX.measure.overlappediv()`, stores current readings in *ibuffer* and voltage readings in *vbuffer*.

This function is an overlapped command. Script execution continues while the measurements are made in the background. Attempts to access result values that have not yet been generated cause the script to block and wait for the data to become available. The `waitcomplete()` function can also be used to wait for the measurements to complete before continuing.

If a given reading buffer contains any data, it is cleared before making any measurements, unless the reading buffer has been configured to append data.

Example

```
smua.measure.overlappedv(smua.nvbuffer1)
```

Starts background voltage measurements for SMU channel A.

Also see

[Reading buffers](#) (on page 6-1)

[smuX.measure.Y\(\)](#) (on page 15-228)

[smuX.nvbufferY](#) (on page 15-230)

[waitcomplete\(\)](#) (on page 15-407)

smuX.measure.rangeY

This attribute contains the positive full-scale value of the measurement range for voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	Voltage: 100e-3 (100 mV) Current: 100e-3 (100 mA)

Usage

```
rangeValue = smuX.measure.rangeY
```

```
smuX.measure.rangeY = rangeValue
```

<i>rangeValue</i>	Set to the maximum expected voltage or current to be measured
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.rangev</code> applies to SMU channel A)
<i>Y</i>	SMU measurement function (<i>v</i> = voltage, <i>i</i> = current)

Details

Reading this attribute returns the positive full-scale value of the measurement range that the SMU is currently using. Assigning a value to this attribute sets the SMU on a fixed range large enough to measure the assigned value. The instrument will select the best range for measuring a value of *rangeValue*.

This attribute is primarily intended to eliminate the time that is required by the automatic range selection performed by a measuring instrument. Because selecting a fixed range prevents autoranging, an overrange condition can occur. For example, measuring 5.0 V on the 2 V range causes an overrange. The value 9.91000E+37 is returned when this occurs.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is locked to be the same as the source range. However, the setting for the measure range is retained. If the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

For example: Assume the source function is voltage. The source range is 1 V and you set the measure range for 6 V. Since the source range is 1 V, the SMU performs voltage measurements on the 1 V range. If you now change the source function to current, voltage measurements are made on the 6 V range.

Explicitly setting a measure range will disable measure autoranging for that function. Autoranging is controlled separately for each source and measurement function: source voltage, source current, measure voltage and measure current. Autoranging is enabled for all four by default.

Changing the range while the output is off does not update the hardware settings, but querying returns the range setting that will be used once the output is turned on. Setting a range while the output is on takes effect immediately.

With measure autoranging enabled, the range is changed only when a measurement is taken. Querying the range after a measurement returns the range selected for that measurement.

Example

```
smua.measure.rangev = 0.5
```

Selects the 1 V measurement range for SMU channel A.

Also see

[Range](#) (on page 5-29)

[reset\(\)](#) (on page 15-176)

[setup.recall\(\)](#) (on page 15-191)

[smuX.measure.autorangeY](#) (on page 15-211)

[smuX.reset\(\)](#) (on page 15-231)

[smuX.source.rangeY](#) (on page 15-245)

smuX.measure.rel.enableY

This attribute turns relative measurements on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	0 (smuX.REL_OFF)

Usage

```
relEnable = smuX.measure.rel.enableY
smuX.measure.rel.enableY = relEnable
```

<i>relEnable</i>	Relative measurement control. Set <i>relEnable</i> to one of the following values: 0 or smuX.REL_OFF: Disables relative measurements 1 or smuX.REL_ON: Enables relative measurements
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.measure.rel.enablev applies to SMU channel A)
<i>Y</i>	SMU measurement function (v = voltage, i = current, r = resistance, p = power)

Details

This attribute enables or disables relative measurements. When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value specified by smuX.measure.rel.levelY. Each returned measured relative reading is the result of the following calculation:

$$\text{Relative reading} = \text{Actual measured reading} - \text{Relative offset value}$$

Example

```
smua.measure.rel.enablev = smua.REL_ON
```

Enables relative voltage measurements for SMU channel A.

Also see

[Relative offset](#) (on page 5-47)
[reset\(\)](#) (on page 15-176)
[setup.recall\(\)](#) (on page 15-191)
[smuX.measure.rel.levelY](#) (on page 15-227)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.rel.levelY

This attribute sets the offset value for relative measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	0

Usage

```
relValue = smuX.measure.rel.levelY
smuX.measure.rel.levelY = relValue
```

<i>relValue</i>	Relative measurement offset value
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.rel.levelv</code> applies to SMU channel A)
<i>Y</i>	SMU measurement function (<i>v</i> = voltage, <i>i</i> = current, <i>r</i> = resistance, <i>p</i> = power)

Details

This attribute specifies the offset value used for relative measurements. When relative measurements are enabled (see `smuX.measure.rel.enableY`), all subsequent measured readings are offset by the value of this attribute. Each returned measured relative reading is the result of the following calculation:

Relative reading = Actual measured reading - Relative offset value

Example

```
smua.measure.rel.levelv = smua.measure.v()
```

Performs a voltage measurement using SMU channel A and then uses it as the relative offset value.

Also see

[Relative offset](#) (on page 5-47)
[reset\(\)](#) (on page 15-176)
[smuX.measure.rel.enableY](#) (on page 15-226)
[smuX.reset\(\)](#) (on page 15-231)

smuX.measure.Y()

This function makes one or more measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```

reading = smuX.measure.Y()
reading = smuX.measure.Y(readingBuffer)
iReading, vReading = smuX.measure.iv()
iReading, vReading = smuX.measure.iv(iReadingBuffer)
iReading, vReading = smuX.measure.iv(iReadingBuffer, vReadingBuffer)

```

<i>reading</i>	Returned value of the last (or only) reading of the measurement process
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.v()</code> applies to SMU channel A)
<i>Y</i>	SMU measurement function (<i>v</i> = voltage, <i>i</i> = current, <i>r</i> = resistance, <i>p</i> = power)
<i>readingBuffer</i>	A reading buffer object where all readings are stored
<i>iReading</i>	The last reading of the current measurement process
<i>vReading</i>	The last reading of the voltage measurement process
<i>iReadingBuffer</i>	A reading buffer object where current readings are stored
<i>vReadingBuffer</i>	A reading buffer object where voltage readings are stored

Details

If you use this function without specifying a reading buffer, it makes one measurement and returns that measurement as *reading*. To use the additional information that is acquired while making a measurement or to return multiple readings, specify a reading buffer. If the instrument is configured to return multiple readings for a measurement and *readingBuffer* is specified, all readings are available in *readingBuffer*, but only the last measurement is returned as *reading*.

Measurements are in the following units of measure:

- *v* = volts
- *i* = amperes
- *r* = ohms
- *p* = watts

The `smuX.measure.iv()` function returns the last actual current measurement and voltage measurement as *iReading* and *vReading*, respectively. Additionally, it can store current and voltage readings if buffers are provided (*iReadingBuffer* and *vReadingBuffer*).

The `smuX.measure.count` attribute determines how many measurements are performed. When using a reading buffer, it also determines the number of readings to store in the buffer. If a reading buffer is not specified, the SMU ignores the `smuX.measure.count` attribute and only makes one measurement.

The *readingBuffer* is cleared before making any measurements unless the buffer is configured to append data.

Example

```
smua.measure.count = 10
smua.measure.v(smua.nvbuffer1)
```

Makes 10 voltage measurements using SMU channel A and stores them in a buffer.

Also see

[Reading buffers](#) (on page 6-1)

[smuX.measure.count](#) (on page 15-215)

[smuX.measure.overlappedY\(\)](#) (on page 15-223)

[smuX.nvbufferY](#) (on page 15-230)

smuX.measureYandstep()

This function makes one or two measurements and then steps the source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = smuX.measureYandstep(sourceValue)
iReading, vReading = smuX.measureivandstep(sourceValue)
```

<i>reading</i>	The measured reading before stepping the source
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measurevandstep()</code> applies to SMU channel A)
<i>Y</i>	SMU measurement function (<i>v</i> = voltage, <i>i</i> = current, <i>r</i> = resistance, <i>p</i> = power)
<i>sourceValue</i>	Source value to be set after the measurement is made
<i>iReading</i>	The current reading before stepping the source
<i>vReading</i>	The voltage reading before stepping the source

Details

The `smuX.measureYandstep()` function makes a measurement and then sets the source to *sourceValue*. Usage of the `smuX.measureivandstep()` function is similar, but makes two measurements simultaneously, one for current (*i*) and one for voltage (*v*).

Measurements are in the following units of measure: *v* = volts, *i* = amperes, *r* = ohms, *p* = watts.

Make sure the specified source value is appropriate for the selected source function. For example, if the source voltage function is selected, then *sourceValue* is expected to be a new voltage level.

Both source and measure autorange must be disabled before using this function. This function cannot be used if source high capacitance mode is enabled (high capacitance mode requires autoranging to be enabled).

This function is provided for very fast execution of source-measure loops. The measurement is made before the source is stepped. Before using this function, and before any loop this function may be used in, set the source value to its initial level.

Example

```
local ivalues = {}
smua.source.rangev = 1
smua.source.levelv = 0
smua.measure.rangei = 0.01
smua.source.output = smua.OUTPUT_ON
for index = 1, 10 do
    ivalues[index] = smua.measureiandstep(index / 10)
end
ivalues[11] = smua.measure.i()
```

This use of the SMU channel A measure and step function measures current starting at a source value of 0 V. After each current measurement, the source is stepped 100 mV for the next current measurement. The final source level is 1 V, where current is again measured.

Also see

[smuX.measure.autorangeY](#) (on page 15-211)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.source.autorangeY](#) (on page 15-233)
[smuX.trigger.source.limitY](#) (on page 15-264)
[smuX.trigger.source.linearY\(\)](#) (on page 15-265)
[smuX.trigger.source.listY\(\)](#) (on page 15-266)
[smuX.trigger.source.logY\(\)](#) (on page 15-267)
[Sweep Operation](#) (on page 5-34)

smuX.nvbufferY

This attribute contains a dedicated reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	See Details	Not applicable

Usage

```
bufferVar = smuX.nvbufferY
```

<i>bufferVar</i>	The dedicated reading buffer
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.nvbuffer1</code> applies to SMU channel A)
<i>Y</i>	SMU nonvolatile buffer (1 or 2)

Details

Each SMU channel contains two dedicated reading buffers: `smuX.nvbuffer1` and `smuX.nvbuffer2`.

All routines that return measurements can also store them in either reading buffer. Overlapped measurements are always stored in a reading buffer. Synchronous measurements return either a single-point measurement or can be stored in a reading buffer if passed to the measurement command.

The dedicated reading buffers can be saved to internal nonvolatile memory (to retain data between power cycles) using the `smuX.savebuffer()` function.

Example

```
smua.measure.overlappedv(smua.nvbuffer1)
```

Store voltage readings from SMU channel A into SMU channel A dedicated reading buffer 1.

Also see

[Configuring and running sweeps](#) (on page 5-43)
[Reading buffers](#) (on page 6-1)
[savebuffer\(\)](#) (on page 15-176)
[smuX.makebuffer\(\)](#) (on page 15-210)
[smuX.measure.overlappedY\(\)](#) (on page 15-223)
[smuX.savebuffer\(\)](#) (on page 15-231)
[smuX.trigger.measure.action](#) (on page 15-257)
[smuX.trigger.measure.set\(\)](#) (on page 15-258)
[smuX.trigger.measure.stimulus](#) (on page 15-259)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

smuX.reset()

This function turns off the output and resets the commands that begin with `smuX.` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.reset()
```

X	Source-measure unit (SMU) channel (for example, <code>smua.reset()</code> applies to SMU channel A)

Details

This function turns off the output and returns the specified SMU to its default settings.

Example

```
smua.reset()
```

Turns off the output and resets SMU channel A to its default settings.

Also see

[reset\(\)](#) (on page 15-176)

smuX.savebuffer()

This function saves one source-measure unit (SMU) dedicated reading buffer to nonvolatile memory (there are two dedicated reading buffers for each SMU).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.savebuffer(smuX.nvbufferY)
```

X	SMU channel (for example, <code>smua.savebuffer(smua.nvbuffer1)</code> applies to SMU channel A)
Y	SMU dedicated reading buffer (1 or 2)

Details

When the instrument is turned off and back on, the dedicated reading buffers are restored from nonvolatile memory to their last saved values.

Example

```
smua.savebuffer(smua.nvbuffer1)
```

Saves buffer 1 (SMU channel A) to internal memory.

Also see

[Reading buffers](#) (on page 6-1)

[savebuffer\(\)](#) (on page 15-176)

[smuX.nvbufferY](#) (on page 15-230)

smuX.sense

This attribute contains the state of the sense mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smuX.SENSE_LOCAL)

Usage

```
senseMode = smuX.sense
smuX.sense = senseMode
```

<i>senseMode</i>	The sense mode; set to one of the following: 0 or smuX.SENSE_LOCAL: Selects local sense (2-wire) 1 or smuX.SENSE_REMOTE: Selects remote sense (4-wire) 3 or smuX.SENSE_CALA: Selects calibration sense mode
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.sense applies to SMU channel A)

Details

Source-measure operations are performed using either 2-wire local sense connections or 4-wire remote sense connections. Writing to this attribute selects the sense mode.

The smuX.SENSE_CALA mode is only used for calibration and may only be selected when calibration is enabled.

The sense mode can be changed between local and remote while the output is on.

The calibration sense mode cannot be selected while the output is on.

Resetting the instrument selects the local sense mode.

Example

```
smua.sense = smua.SENSE_REMOTE
```

Selects remote sensing for SMU channel A.

Also see

[2-wire local sensing connections](#) (on page 4-6)

[4-wire remote sensing connections](#) (on page 4-6)

[Sense mode selection](#) (on page 5-11)

smuX.source.autorangeY

This attribute contains the state of the source autorange control (on/off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	1 (smuX.AUTORANGE_ON)

Usage

```
sourceAutorange = smuX.source.autorangeY
smuX.source.autorangeY = sourceAutorange
```

<i>sourceAutorange</i>	The state of the source autorange control. Set to one of the following: 0 or smuX.AUTORANGE_OFF: Disables source autorange 1 or smuX.AUTORANGE_ON: Enables source autorange
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.source.autorangev applies to SMU channel A)
<i>Y</i>	SMU source function (v = voltage, i = current)

Details

This attribute indicates the source autorange state. Its value is smuX.AUTORANGE_OFF when the SMU source circuit is on a fixed range and smuX.AUTORANGE_ON when it is in autorange mode.

Setting this attribute to smuX.AUTORANGE_OFF puts the SMU on a fixed source range. The fixed range used is the present SMU source circuit range.

Setting this attribute to smuX.AUTORANGE_ON puts the SMU source circuit into autorange mode. If the source output is on, the SMU immediately changes range to the range most appropriate for the value being sourced if that range is different than the present SMU range.

Autorange is disabled if the source level is edited from the virtual front panel. Setting the source range also turns off autorange when set by using the smuX.source.rangeY attribute.

Resetting the instrument selects the smuX.AUTORANGE_ON.

Example

smua.source.autorangev = smua.AUTORANGE_ON	Enables volts source autorange for SMU channel A.

Also see

[smuX.measure.autorangeY](#) (on page 15-211)

[smuX.source.rangeY](#) (on page 15-245)

smuX.source.calibrateY()

This function generates and activates new source calibration constants.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.source.calibrateY(range, cp1Expected, cp1Reference, cp2Expected, cp2Reference)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.calibratev()</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)
<i>range</i>	The measurement range to adjust
<i>cp1Expected</i>	The source value set for point 1
<i>cp1Reference</i>	The reference measurement for point 1 as measured externally
<i>cp2Expected</i>	The source value set for point 2
<i>cp2Reference</i>	The reference measurement for point 2 as measured externally

Details

This function generates and activates new calibration constants for the given range.

The positive and negative polarities of the source must be adjusted separately. Use a positive value for *range* to adjust the positive polarity and a negative value for range to adjust the negative polarity. Do not use 0.0 for a negative point; 0.0 is considered to be a positive number.

Typically, the two points that are used are near zero for point 1 and 90% of full scale for point 2. Full scale for point 2 should be avoided if the source of the SMU is substantially out of calibration.

The two reference measurements must be made with the source using the active calibration set. For example, source a value, measure it, and do not change the active calibration set before issuing this command.

The new calibration constants are activated immediately but they are not written to nonvolatile memory. Use the `smuX.cal.save()` function to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is only available when calibration is unlocked using `smuX.cal.unlock()`.

Example

```
smua.source.calibratev(1, 1e-10, 1e-5, 0.9, 0.903)
```

Generates and activates new source calibration constants for the 1 A range. For point 1, it uses 1e-10 as the source value and 1e-5 as the reference measurement. For point 2, it uses 0.9 for the source value and 0.903 for the reference measurement.

Also see

[Calibration](#) (on page 17-1)
[smuX.cal.restore\(\)](#) (on page 15-201)
[smuX.cal.save\(\)](#) (on page 15-202)
[smuX.cal.unlock\(\)](#) (on page 15-204)
[smuX.measure.calibrateY\(\)](#) (on page 15-214)

smuX.source.compliance

This attribute contains the state of source compliance.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
compliance = smuX.source.compliance
```

<i>compliance</i>	The state of source compliance: ■ <code>true</code>: Indicates that the limit function is in control of the source (source in compliance) ■ <code>false</code>: Indicates that the source function is in control of the output (source not in compliance)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.compliance</code> applies to SMU channel A)

Details

Reading this attribute updates the status model and the virtual front panel with generated compliance information. See Current Limit (ILMT) in the status model diagram for the [Measurement event registers](#) (on page 10-22). The Voltage Limit (VLMT) is not shown in the status model diagram for the Measurement event registers but is similar to the Current Limit (ILMT).

Example

```
compliance = smua.source.compliance
```

```
print(compliance)
```

Reads the source compliance state for SMU channel A.

Output:

```
true
```

This output indicates that a configured limit has been reached (voltage, current, or power limit).

Also see

[smuX.source.limitY](#) (on page 15-239)

smuX.source.delay

This attribute contains the source delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.DELAY_OFF)

Usage

```
sDelay = smuX.source.delay  
smuX.source.delay = sDelay
```

<i>sDelay</i>	Set to the source delay value (for example, to specify an additional 10 ms source delay, set the value to 0.010); you can also set it one of the following values: 0 or smuX.DELAY_OFF: No delay -1 or smuX.DELAY_AUTO: Automatic delay value
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.source.delay applies to SMU channel A)

Details

This attribute allows for additional delay (settling time) after an output step.

The smuX.DELAY_AUTO setting causes a range-dependent delay to be inserted when the source is changed. Range-dependent delays are based on the output settling time values as defined in the 2606B specifications.

Example

```
smua.source.delay = smua.DELAY_AUTO
```

Sets the delay for SMU channel A to automatic (a range-dependent delay is inserted whenever the source is changed).

Also see

[reset\(\)](#) (on page 15-176)
[smuX.measure.count](#) (on page 15-215)
[smuX.measure.delay](#) (on page 15-216)
[smuX.reset\(\)](#) (on page 15-231)

smuX.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	SMU reset Instrument reset Recall setup	Saved setup	1 (smuX.OUTPUT_DCVOLTS)

Usage

```
sFunction = smuX.source.func
smuX.source.func = sFunction
```

<i>sFunction</i>	The source function; set to one of the following values: 0 or smuX.OUTPUT_DCAMPS: Selects the current source function 1 or smuX.OUTPUT_DCVOLTS: Selects the voltage source function
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.source.func applies to SMU channel A)

Details

Reading this attribute indicates the output function of the source for the specified SMU channel. Setting this attribute configures the SMU channel as either a voltage source or a current source.

Example

```
smua.source.func = smua.OUTPUT_DCAMPS
```

Sets the source function of SMU channel A to be a current source.

Also see

[smuX.source.levelY](#) (on page 15-238)

[smuX.source.output](#) (on page 15-243)

smuX.source.highc

This attribute enables or disables high-capacitance mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smuX.DISABLE)

Usage

```
highC = smuX.source.highc
smuX.source.highc = highC
```

<i>highC</i>	The state of the high-capacitance mode; set to one of the following values: 0 or smuX.DISABLE: Disables high-capacitance mode 1 or smuX.ENABLE: Enables high-capacitance mode
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.source.highc applies to SMU channel A)

Details

When enabled, the high-capacitance mode has the following effects on the SMU settings:

- `smuX.measure.autorangei` is set to `smuX.AUTORANGE_FOLLOW_LIMIT` and cannot be changed
- Current ranges below 1 μ A are not accessible
- If `smuX.source.limiti` is less than 1 μ A, it is raised to 1 μ A
- If `smuX.source.rangei` is less than 1 μ A, it is raised to 1 μ A
- If `smuX.source.lowrangei` is less than 1 μ A, it is raised to 1 μ A
- If `smuX.measure.lowrangei` is less than 1 μ A, it is raised to 1 μ A

Example

```
smua.source.highc = smua.ENABLE
```

Activates high-capacitance mode for SMU channel A.

Also see

[High-capacitance mode](#) (on page 5-52)

smuX.source.levelY

This attribute sets the source level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0

Usage

```
sourceLevel = smuX.source.levelY
smuX.source.levelY = sourceLevel
```

<i>sourceLevel</i>	The source value; set to one of the following values: ■ Voltage: 0 V to ± 20 V ■ Current: 0 A to ± 3 A
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.levelv</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)

Details

This attribute configures the output level of the voltage or current source.

If the source is configured as a voltage source and the output is on, the new `smuX.source.levelv` setting is sourced immediately. If the output is off or the source is configured as a current source, the voltage level is sourced when the source is configured as a voltage source and the output is turned on.

If the source is configured as a current source and the output is on, the new `smuX.source.level` setting is sourced immediately. If the output is off or the source is configured as a voltage source, the current level is sourced when the source is configured as a current source and the output is turned on.

The sign of `sourceLevel` 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.

The `reset()` function sets the source levels to 0 V and 0 A.

Example

<code>smua.source.levelv = 1</code>	Sets voltage source of SMU channel A to 1 V.

Also see

[smuX.source.compliance](#) (on page 15-235)

[smuX.source.func](#) (on page 15-237)

[smuX.source.output](#) (on page 15-243)

[Source-measure concepts](#) (on page 8-1)

smuX.source.limitY

This attribute sets compliance limits.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Limit voltage: 20 (20 V) Limit current: 1 (1 A) Limit power: 0 (disabled)

Usage

```
limit = smuX.source.limitY
smuX.source.limitY = limit
```

<i>limit</i>	The compliance limit value; set to one of the following values: - Voltage compliance: 10 mV to 20 V - Current compliance: 10 nA to 3 A - Power compliance (in watts)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.limitv</code> applies to SMU channel A)
<i>Y</i>	SMU function (v = voltage, i = current, p = power)

Details

Use the `smuX.source.limiti` attribute to limit the current output of the voltage source. Use `smuX.source.limitv` to limit the voltage output of the current source. The SMU always uses autoranging for the limit setting. Use the `smuX.source.limitp` attribute to limit the output power of the source.

Set this attribute in the test sequence before the turning the source on.

Using a limit value of 0 results in error code 1102, "Parameter too small," for *v* and *i*. Setting this attribute to zero disables power compliance for *p*. When setting the power compliance limit to a nonzero value, the SMU adjusts the source limit where appropriate to limit the output to the specified power. The SMU uses the lower of the programmed compliance value (the compliance level that is if power compliance is disabled) or the limit calculated from the power compliance setting.

Reading this attribute indicates the presently set compliance value. Use `smuX.source.compliance` to read the state of source compliance.

Example

```
smua.source.limitv = 15
```

Sets the voltage limit of SMU channel A to 15 V.

Also see

[DUT test connections](#) (on page 4-3)

[smuX.source.compliance](#) (on page 15-235)

[smuX.source.func](#) (on page 15-237)

[smuX.source.output](#) (on page 15-243)

smuX.source.lowrangeY

This attribute sets the lowest source range that is used during autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Voltage: 100e-3 (100 mV) Current: 100e-9 (100 nA)

Usage

```
sourceRangeLow = smuX.source.lowrangeY
smuX.source.lowrangeY = sourceRangeLow
```

<i>sourceRangeLow</i>	Set to the lowest voltage (in volts) or current (in amperes) range to be used
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.lowrangev</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)

Details

This attribute is used with source autoranging to put a lower bound on the range that is used. Lower ranges generally require greater settling times. If you set a low range value, you might be able to source small values with less settling time.

If the instrument is set to autorange and it is on a range lower than the one specified by *sourceRangeLow*, the source range is changed to the range specified by *sourceRangeLow*.

Example

```
smua.source.lowrangev = 1
```

Sets volts low range for SMU A to 1 V. This prevents the source from using the 100 mV range when sourcing voltage.

Also see

[smuX.source.autorangeY](#) (on page 15-233)

smuX.source.offfunc

This attribute sets the source function that is used (source 0 A or 0 V) when the output is turned off and the source-measure unit (SMU) is in normal output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	1 (smuX.OUTPUT_DCVOLTS)

Usage

```
offfunc = smuX.source.offfunc
smuX.source.offfunc = offfunc
```

<i>offfunc</i>	Set to the source function to be used when the output is off and the SMU is in normal output-off mode. Set to one of the following values: 0 or smuX.OUTPUT_DCAMPS: Source 0 A 1 or smuX.OUTPUT_DCVOLTS: Source 0 V
<i>X</i>	SMU channel (for example, smua.source.offfunc applies to SMU channel A)

Details

This attribute controls the source function used when the output is turned off and `smuX.source.offmode` is set to `smuX.OUTPUT_NORMAL`.

Set this attribute to `smuX.OUTPUT_DCVOLTS` for the source to be a 0 V source when the output is off (`smuX.source.offlimiti` is used).

Set it to `smuX.OUTPUT_DCAMPS` for the source to be a 0 A source when the output is off (`smuX.source.offlimitv` is used).

Example

```
smua.source.offmode = smua.OUTPUT_NORMAL
smua.source.offfunc = smua.OUTPUT_DCVOLTS
```

Sets the normal output-off mode to source 0 V when the output is turned off for SMU channel A.

Also see

[Output-off states](#) (on page 5-25)
[smuX.source.offlimitY](#) (on page 15-241)
[smuX.source.offmode](#) (on page 15-242)
[smuX.source.output](#) (on page 15-243)

smuX.source.offlimitY

This attribute sets the limit (current or voltage) used when the source-measure unit (SMU) is in normal output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Current: 1e-3 (1 mA) Voltage: 20 (20 V)

Usage

```
sourceLimit = smuX.source.offlimitY
smuX.source.offlimitY = sourceLimit
```

<i>sourceLimit</i>	Set to the limit to be used when the SMU is in normal output-off mode
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.offlimiti</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)

Details

Setting the current limit to lower than 1 mA may interfere with operation of the contact check function. See `smuX.contact.check()` and `smuX.contact.r()` for details.

Example

```
smua.source.offlimiti = 10e-3
```

Changes the normal output-off mode limit to 10 mA for SMU channel A.

Also see

[smuX.contact.check\(\)](#) (on page 15-207)

[smuX.contact.r\(\)](#) (on page 15-208)

[smuX.source.offfunc](#) (on page 15-241)

[smuX.source.offmode](#) (on page 15-242)

smuX.source.offmode

This attribute sets the source output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (<code>smuX.OUTPUT_NORMAL</code>)

Usage

```
sourceOffMode = smuX.source.offmode
smuX.source.offmode = sourceOffMode
```

<i>sourceOffMode</i>	The output-off setting; set to one of the following values: 0 or <code>smuX.OUTPUT_NORMAL</code>: Configures the source function according to <code>smuX.source.offfunc</code> attribute 1 or <code>smuX.OUTPUT_ZERO</code>: Configures source to output 0 V as <code>smuX.OUTPUT_NORMAL</code> with different compliance handling (see the Details below) 2 or <code>smuX.OUTPUT_HIGH_Z</code>: Opens the output relay when the output is turned off
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.offmode</code> applies to SMU channel A)

Details

Reading this attribute returns the output-off mode of the source. Setting this attribute configures the SMU output-off mode.

The default *sourceOffMode* is `smuX.OUTPUT_NORMAL`. In this mode, the source function is configured according to the `smuX.source.offfunc` attribute. The `smuX.source.offfunc` attribute controls whether the SMU is configured as a 0 V voltage source or a 0 A current source. When the SMU is operating as a 0 A current source, the `smuX.source.offlimitv` attribute sets the voltage limit (similar to how the `smuX.source.offlimiti` attribute sets the current limit when the SMU is operating as a 0 V voltage source).

When the *sourceOffMode* is set to `smuX.OUTPUT_ZERO`, the source is configured to output 0 V just as `smuX.OUTPUT_NORMAL` mode with `smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`. If the source function is voltage, the current limit is not changed. If the source function is current, the current limit is set to the current source level or 10 percent of the current source range, whichever is greater.

When *sourceOffMode* is set to `smuX.OUTPUT_HIGH_Z`, the SMU opens the output relay when the output is turned off.

Example

```
smua.source.offmode = smua.OUTPUT_HIGH_Z
```

Sets the output-off mode for SMU channel A to open the output relay when the output is turned off.

Also see

[Output-off states](#) (on page 5-25)

[smuX.source.offfunc](#) (on page 15-241)

[smuX.source.offlimitY](#) (on page 15-241)

[smuX.source.output](#) (on page 15-243)

smuX.source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (<code>smuX.OUTPUT_OFF</code>)

Usage

```
sourceOutput = smuX.source.output
smuX.source.output = sourceOutput
```

<i>sourceOutput</i>	The output state setting of the source; set to one of the following values: 0 or <code>smuX.OUTPUT_OFF</code>: Turns off the source output 1 or <code>smuX.OUTPUT_ON</code>: Turns on the source output 2 or <code>smuX.OUTPUT_HIGH_Z</code>: Turns off the output in high Z mode (allows you to go to high Z mode without first setting the <code>smuX.source.offmode</code> attribute to <code>smuX.OUTPUT_HIGH_Z</code>)
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.source.output</code> applies to SMU channel A)

Details

Reading this attribute returns the output state of the source. Setting this attribute switches the output of the source on or off.

When the output is switched on, the SMU sources either voltage or current, as set by `smuX.source.func`.

Setting this attribute to `smuX.OUTPUT_HIGH_Z` causes the output to turn off and go to the High Z mode. If the `smuX.source.output` is read after setting this attribute to `smuX.OUTPUT_HIGH_Z`, it returns 0.

Example

```
smua.source.output = smua.OUTPUT_ON
```

Turns on the SMU channel A source output.

Also see

[DUT test connections](#) (on page 4-3)

[smuX.source.func](#) (on page 15-237)

[smuX.source.offmode](#) (on page 15-242)

smuX.source.outputenableaction

This attribute controls output enable action of the source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smuX.OE_NONE)

Usage

```
outputAction = smuX.source.outputenableaction
smuX.source.outputenableaction = outputAction
```

<i>outputAction</i>	The output enable action of the source; set to one of the following values: 0 or <code>smuX.OE_NONE</code>: No action 1 or <code>smuX.OE_OUTPUT_OFF</code>: Turns the source output off
X	Source-measure unit (SMU) channel (for example, <code>smua.source.outputenableaction</code> applies to SMU channel A)

Details

This attribute controls the action the SMU takes when the output enable line goes low (deasserted).

When set to `smuX.OE_NONE`, the SMU takes no action when the output enable line goes low.

When set to `smuX.OE_OUTPUT_OFF` and the output enable line is de-asserted, the SMU turns its output off as if the `smuX.source.output = smuX.OUTPUT_OFF` command had been received.

The SMU does not automatically turn its output on when the output enable line returns to the high state.

If the output enable line is not asserted when this attribute is set to `smuX.OE_OUTPUT_OFF` and the output is on, the output turns off immediately.

Detection of the output enable line going low does not abort any running scripts. This may cause execution errors.

Example

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

Sets SMU channel A to turn off the output if the output enable line goes low (deasserted).

Also see

[smuX.source.offmode](#) (on page 15-242)

[smuX.source.output](#) (on page 15-243)

smuX.source.rangeY

This attribute contains the source range.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Voltage: 100e-3 (100 mV) Current: 100e-9 (100 nA)

Usage

```
rangeValue = smuX.source.rangeY
smuX.source.rangeY = rangeValue
```

<i>rangeValue</i>	Set to the maximum expected voltage or current to be sourced
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.measure.rangev</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)

Details

This attribute contains a value that sets the source-measure unit (SMU) to a fixed range large enough to source the value. When read, the attribute contains the range the instrument is presently on when in autorange.

Assigning a value to this attribute sets the SMU to a fixed range large enough to source the assigned value. The instrument selects the best range for sourcing a value of *rangeValue*.

Reading this attribute returns the positive full-scale value of the source range the SMU is currently using. With source autoranging enabled, the output level controls the range. Querying the range after the level is set returns the range the instrument chose as appropriate for that source level.

This attribute is primarily intended to eliminate the time required by the automatic range selection performed by a sourcing instrument. Because selecting a fixed range prevents autoranging, an overrange condition can occur. For example, sourcing 10.0 V on the 6 V range causes an overrange condition.

Example

```
smua.source.rangev = 1
```

Selects the 1 V source range for SMU channel A.

Also see

[Range](#) (on page 5-29)

[reset\(\)](#) (on page 15-176)

[setup.recall\(\)](#) (on page 15-191)

[smuX.measure.rangeY](#) (on page 15-224)

[smuX.reset\(\)](#) (on page 15-231)

[smuX.source.autorangeY](#) (on page 15-233)

smuX.source.settling

This attribute contains the source settling mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.SETTLE_SMOOTH)

Usage

```
settleOption = smuX.source.settling
smuX.source.settling = settleOption
```

<i>settleOption</i>	Set to the source settling mode. Set to one of the following values: 0 or <code>smuX.SETTLE_SMOOTH</code>: Turns off additional settling operations (default) 1 or <code>smuX.SETTLE_FAST_RANGE</code>: Instructs the source-measure unit (SMU) to use a faster procedure when changing ranges 2 or <code>smuX.SETTLE_FAST_POLARITY</code>: Instructs the SMU to change polarity without going to zero 3 or <code>smuX.SETTLE_DIRECT_IRANGE</code>: Instructs the SMU to change the current range directly 128 or <code>smuX.SETTLE_FAST_ALL</code>: Enables all <code>smuX.SETTLE_FAST_*</code> operations
X	SMU channel (for example, <code>smua.source.settling</code> applies to SMU channel A)

Details

Using `smuX.SETTLE_FAST_RANGE` may cause the SMU to exceed the range change overshoot specification.

`smuX.SETTLE_FAST_POLARITY` does not go to zero when changing polarity and may create inconsistencies at the zero crossing.

`smuX.SETTLE_DIRECT_IRANGE` switches the SMU directly to the target range instead of the default “range-by-range” method. This option is mutually exclusive of any other `smuX.SETTLE_FAST_*` commands.

Example

```
smua.source.settling = smua.SETTLE_FAST_POLARITY
Selects fast polarity changing for SMU channel A.
```

Also see

[Range](#) (on page 5-29)

smuX.trigger.arm.count

This attribute sets the arm count in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	1

Usage

```
triggerArmCount = smuX.trigger.arm.count
smuX.trigger.arm.count = triggerArmCount
```

<i>triggerArmCount</i>	The arm count in the trigger model
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.arm.count</code> applies to SMU channel A)

Details

During a sweep, the SMU iterates through the arm layer of the trigger model this many times. After performing this many iterations, the SMU returns to an idle state.

If this count is set to zero, the SMU stays in the trigger model indefinitely until aborted.

Example

```
smua.trigger.arm.count = 5
```

Sets the SMU channel A to iterate through the arm layer of the trigger model five times and then return to the idle state.

Also see

[smuX.trigger.count](#) (on page 15-250)

smuX.trigger.arm.set()

This function sets the arm event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.arm.set()
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.arm.set()</code> applies to SMU channel A)

Details

The SMU automatically clears all the event detectors when the `smuX.trigger.initiate()` function is executed. Call this function after the sweep is initiated.

A typical example that uses this function is when you want the SMU to immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur.

This function start actions on the SMU if a trigger event is missed.

Example

```
smua.trigger.arm.set()
```

Sets the arm event detector to the detected state for SMU channel A.

Also see

[smuX.trigger.initiate\(\)](#) (on page 15-256)

[smuX.trigger.measure.set\(\)](#) (on page 15-258)

[smuX.trigger.source.set\(\)](#) (on page 15-268)

smuX.trigger.arm.stimulus

This attribute selects the event that will cause the arm event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.arm.stimulus
smuX.trigger.arm.stimulus = eventID
```

<i>eventID</i>	Event that triggers the arm detector
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.arm.stimulus</code> applies to SMU channel A)

Details

Set this attribute to the event ID of any trigger event generator to wait for that event.

Set this attribute to zero to bypass waiting for events at the arm event detector (the SMU continues uninterrupted through the remote trigger model). Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measurement action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed

Trigger event IDs*	
Event ID	Event description
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Example

```
smua.trigger.arm.stimulus = trigger.timer[1].EVENT_ID
```

An event on trigger timer 1 causes the arm event detector to enter the detected state.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.ARMED_EVENT_ID

This constant contains the number of the armed event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.ARMED_EVENT_ID
```

<code>eventID</code>	The armed event number
<code>X</code>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.ARMED_EVENT_ID</code> applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to armed events from this SMU.

Example

```
trigger.timer[1].stimulus = smua.trigger.ARMED_EVENT_ID
```

Trigger timer when the SMU passes through the arm layer.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.autoclear

This attribute turns automatic clearing of the event detectors on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.DISABLE)

Usage

```
autoClear = smuX.trigger.autoclear
smuX.trigger.autoclear = autoClear
```

<i>autoClear</i>	Auto clear setting; set to one of the following values: 0 or <code>smuX.DISABLE</code>: Turns off automatic clearing of the event detectors 1 or <code>smuX.ENABLE</code>: Turns on automatic clearing of the event detectors
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.autoclear</code> applies to SMU channel A)

Details

This attribute enables or disables automatic clearing of the trigger model state machine event detectors when the SMU transitions from the arm layer to the trigger layer.

Only the detected states of the event detectors are cleared.

The overrun statuses of the event detectors are not automatically cleared when the SMU transitions from the arm layer to the trigger layer.

The event detectors are always cleared when a sweep is initiated.

Example

```
smua.trigger.autoclear = smua.ENABLE
```

Automatically clear the event detectors for the trigger mode state.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.count

This attribute sets the trigger count in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	1

Usage

```
triggerCount = smuX.trigger.count
smuX.trigger.count = triggerCount
```

<i>triggerCount</i>	The trigger count is the number of times the source-measure unit (SMU) iterates in the trigger layer for any given sweep
<i>X</i>	SMU channel (for example, <code>smua.trigger.count</code> applies to SMU channel A)

Details

During a sweep, the SMU iterates through the trigger layer of the trigger model the number of times set by this attribute. After performing the iterations, the SMU returns to the arm layer.

If this count is set to zero (0), the SMU stays in the trigger model indefinitely until aborted.

Example

```
reset()
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
smua.trigger.source.listv( {5} )
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
pulse_timer.delay = 0.0006
pulse_timer.stimulus = period_timer.EVENT_ID
pulse_timer.count = 1
period_timer.delay = 0.005
period_timer.count = 9
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
period_timer.passthrough = true
smua.trigger.source.stimulus = period_timer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
smua.trigger.count = 1
smua.trigger.arm.count = 10
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
```

Generate a 10-point pulse train where each pulse has a width of 600 μ s and a pulse period of 5 ms.

Alias the trigger timers to use for pulse width and period.

Create a fixed level voltage sweep.

Set the pulse width and trigger the pulse width timer with a period timer.

Set the pulse period to output one pulse per period and the count to generate 10 pulses.

Trigger the pulse period timer when a sweep is initiated.

Configure the timer to output a trigger event when it starts the first delay.

Trigger the SMU source action using pulse period timer.

Trigger the SMU end pulse action using pulse width timer.

Set the trigger model counts.

Configure the SMU to execute a 10-point pulse train.

Start the trigger model.

Wait for the sweep to complete.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.endpulse.action

This attribute enables or disables pulse mode sweeps.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	1 (smuX.SOURCE_HOLD)

Usage

```
pulseAction = smuX.trigger.endpulse.action  
smuX.trigger.endpulse.action = pulseAction
```

<i>pulseAction</i>	The pulse mode setting; set to one of the following values (see Details for definition): 0 or smuX.SOURCE_IDLE 1 or smuX.SOURCE_HOLD
X	Source-measure unit (SMU) channel (for example, smua.trigger.endpulse.action applies to SMU channel A)

Details

When set to smuX.SOURCE_HOLD, this attribute disables pulse mode sweeps, holding the source level for the remainder of the step.

When set to smuX.SOURCE_IDLE, this attribute enables pulse mode sweeps, setting the source level to the programmed (idle) level at the end of the pulse.

Example

```
smua.trigger.endpulse.action = smua.SOURCE_IDLE  
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
```

Configure the end pulse action to achieve a pulse and configure trigger timer 1 to control the end of pulse.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.endpulse.set()

This function sets the end pulse event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.endpulse.set()
```

X	Source-measure unit (SMU) channel (for example, smua.trigger.endpulse.set() applies to SMU channel A)

Details

This function sets the end pulse event detector to the detected state.

The SMU automatically clears all the event detectors when the `smuX.trigger.initiate()` function is executed. Therefore, call `smuX.trigger.endpulse.set()` after the sweep is initiated. If the event detectors are configured to clear automatically because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`, make sure that `smuX.trigger.endpulse.set()` is issued after the SMU has entered the trigger layer.

Example

```
reset()
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
smua.trigger.source.listv( {5} )
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
pulse_timer.delay = 0.0006
pulse_timer.stimulus = period_timer.EVENT_ID
pulse_timer.count = 1
period_timer.delay = 0.005
period_timer.count = 9
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
period_timer.passthrough = true
smua.trigger.source.stimulus = period_timer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
smua.trigger.count = 1
smua.trigger.arm.count = 10
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
```

Generate a 10-point pulse train where each pulse has a width of 600 μ s and a pulse period of 5 ms.

Alias the trigger timers to use for pulse width and period.

Create a fixed level voltage sweep.

Set the pulse width and trigger the pulse width timer with a period timer.

Set the pulse period to output one pulse per period and the count to generate 10 pulses.

Trigger the pulse period timer when a sweep is initiated.

Configure the timer to output a trigger event when it starts the first delay.

Trigger the SMU source action using pulse period timer.

Trigger the SMU end pulse action using pulse width timer.

Set the trigger model counts.

Configure the SMU to execute a 10-point pulse train.

Start the trigger model.

Wait for the sweep to complete.

Also see

[smuX.trigger.autoclear](#) (on page 15-250)

[smuX.trigger.initiate\(\)](#) (on page 15-256)

[Triggering](#) (on page 7-1)

smuX.trigger.endpulse.stimulus

This attribute defines which event will cause the end pulse event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.endpulse.stimulus
smuX.trigger.endpulse.stimulus = eventID
```

<i>eventID</i>	Set to the event that triggers the end pulse action of the source
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.endpulse.stimulus</code> applies to SMU channel A)

Details

Set this attribute to the event ID of any trigger event generator to wait for that event. To bypass waiting for an event, set the value of this attribute 0. Set *eventID* to one of the existing trigger event IDs, which are shown in the following table.

Trigger event IDs*	
Event ID	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measurement action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected

Trigger event IDs*	
Event ID	Event description
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Example

```
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
```

Configure the end pulse action to achieve a pulse and select the event, `trigger.timer[1].EVENT_ID`, that causes the arm event detector to enter the detected state.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.endsweep.action

This attribute sets the action of the source at the end of a sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (<code>smuX.SOURCE_IDLE</code>)

Usage

```
action = smuX.trigger.endsweep.action
smuX.trigger.endsweep.action = action
```

<i>action</i>	The source action at the end of a sweep; set to one of the following values: 0 or <code>smuX.SOURCE_IDLE</code>: Sets the source level to the programmed (idle) level at the end of the sweep 1 or <code>smuX.SOURCE_HOLD</code>: Sets the source level to stay at the level of the last step
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.endsweep.action</code> applies to SMU channel A)

Details

Use this attribute to configure the source action at the end of the sweep. The SMU can be programmed to return to the idle source level or hold the last value of the sweep.

Example

```
smua.trigger.endsweep.action = smua.SOURCE_IDLE
```

Sets SMU channel A to return the source back to the idle source level at the end of a sweep.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.IDLE_EVENT_ID

This constant contains the idle event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.IDLE_EVENT_ID
```

<i>eventID</i>	The idle event number
X	Source-measure unit (SMU) channel (for example, smua.trigger.IDLE_EVENT_ID applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to idle events from this SMU.

Example

```
trigger.timer[1].stimulus = smua.trigger.IDLE_EVENT_ID  
Trigger timer 1 when the SMU returns to the idle layer.
```

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.initiate()

This function initiates a sweep operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.initiate()
```

X	Source-measure unit (SMU) channel (for example, smua.trigger.initiate() applies to SMU channel A)

Details

This function causes the SMU to clear the four trigger model event detectors and enter its trigger model (moves the SMU from the idle state into the arm layer).

To perform source actions during the sweep, before calling this function, it is necessary to configure and enable one of the following sweep source actions:

- `smuX.trigger.source.linearY()`
- `smuX.trigger.source.listY()`
- `smuX.trigger.source.logY()`

To make measurements during the sweep, you must also configure and enable the measure action using `smuX.trigger.measure.Y()`.

If you run this function more than once without reconfiguring the sweep measurements, the caches on the configured measurement reading buffers hold stale data. Use the `bufferVar.clearcache()` function to remove stale values from the reading buffer cache.

This function initiates an overlapped operation.

Example

```
smua.trigger.initiate()
```

Starts a preconfigured sweep and clears the event detectors for SMU channel A.

Also see

[bufferVar.clearcache\(\)](#) (on page 15-21)

[Configuring and running sweeps](#) (on page 5-43)

[smuX.trigger.measure.action](#) (on page 15-257)

[smuX.trigger.measure.Y\(\)](#) (on page 15-261)

[smuX.trigger.source.action](#) (on page 15-263)

[smuX.trigger.source.linearY\(\)](#) (on page 15-265)

[smuX.trigger.source.listY\(\)](#) (on page 15-266)

[smuX.trigger.source.logY\(\)](#) (on page 15-267)

[Triggering](#) (on page 7-1)

smuX.trigger.measure.action

This attribute controls measurement actions during a sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.DISABLE)

Usage

```
action = smuX.trigger.measure.action
smuX.trigger.measure.action = action
```

<i>action</i>	The sweep measurement action; set to one of the following values: ■ 0 or <code>smuX.DISABLE</code>: Do not make measurements during the sweep ■ 1 or <code>smuX.ENABLE</code>: Make measurements during the sweep ■ 2 or <code>smuX.ASYNC</code>: Make measurements during the sweep, but asynchronously with the source area of the trigger model
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.measure.action</code> applies to SMU channel A)

Details

With this attribute enabled (setting *action* to `smuX.ENABLE` or `smuX.ASYNC`), configure the measurement with one of the `smuX.trigger.measure.Y()` functions.

If this attribute is set to `smuX.ASYNC`:

- Asynchronous sweep measurements can only be used with measure autoranging turned off. To turn measure autoranging off for all measurements during the sweep, set the `smuX.measure.autorangeY` attribute to `smuX.AUTORANGE_OFF`.
- Autozero must also be turned off. To turn off autozero, set the `smuX.measure.autozero` attribute to `smuX.AUTOZERO_OFF` or `smuX.AUTOZERO_ONCE`.
- The reading buffer used by `smuX.trigger.measure.Y()` must have `bufferVar.collectsourcevalues` set to 0.

If any of the above items is incorrectly configured, the `smuX.trigger.initiate()` function generates an error.

Example

```
smua.trigger.measure.v(smua.nvbuffer1)
smua.trigger.measure.action = smua.ENABLE
```

Configure sweep voltage measurements.
Enable voltage measurements during the sweep.

Also see

[bufferVar.collectsourcevalues](#) (on page 15-22)
[smuX.trigger.autoclear](#) (on page 15-250)
[smuX.trigger.measure.Y\(\)](#) (on page 15-261)
[Triggering](#) (on page 7-1)

smuX.trigger.measure.set()

This function sets the measurement event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.measure.set()
```

X	Source-measure unit (SMU) channel (for example, <code>smua.trigger.measure.set()</code> applies to SMU channel A)

Details

This function is useful whenever you want the SMU to continue operation without waiting for a programmed trigger event. When called, this function immediately satisfies the event detector, allowing the SMU to continue through the trigger model.

For example, you might use this function to have the SMU immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur.

If the event detectors are configured to clear automatically because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`, make sure that `smuX.trigger.measure.set()` is issued after the SMU has entered the trigger layer. This function can also be used to start actions on the SMU in case of a missed trigger event.

The SMU automatically clears all event detectors when the `smuX.trigger.initiate()` function is executed. Call this function after the sweep is initiated.

Example

<code>smua.trigger.measure.set()</code>	Sets the measure event detector of SMU channel A.

Also see

[smuX.trigger.arm.set\(\)](#) (on page 15-247)
[smuX.trigger.autoclear](#) (on page 15-250)
[smuX.trigger.endpulse.set\(\)](#) (on page 15-252)
[smuX.trigger.source.set\(\)](#) (on page 15-268)

smuX.trigger.measure.stimulus

This attribute selects the event that causes the measure event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.measure.stimulus
smuX.trigger.measure.stimulus = eventID
```

<i>eventID</i>	Event that triggers the measurement detector
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.measure.stimulus</code> applies to SMU channel A)

Details

Set this attribute to the event ID of any trigger event generator to wait for that event. When set, the SMU waits for the event at the measurement event detector portion of the trigger model.

Set this attribute to zero to bypass waiting for an event (the SMU continues uninterrupted through the remote trigger model). Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measurement action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Example

<code>smua.trigger.measure.stimulus = trigger.timer[1].EVENT_ID</code>	Sets delay before measurement begins on SMU channel A.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.measure.Y()

This function configures the measurements that are to be made in a subsequent sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.measure.Y(rbuffer)
smuX.trigger.measure.iv(ibuffer, vbuffer)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.measure.v()</code> applies to SMU channel A)
<i>Y</i>	SMU measurement type (<i>v</i> = voltage, <i>i</i> = current, <i>r</i> = resistance, <i>p</i> = power)
<i>rbuffer</i>	A reading buffer object where the readings are stored
<i>ibuffer</i>	A reading buffer object where current readings are stored
<i>vbuffer</i>	A reading buffer object where voltage readings are stored

Details

As measurements are made, they are stored in a reading buffer. If the instrument is configured to return multiple readings where one is requested, the readings are available as they are made. Measurements are in the following units of measure: *v* = volts, *i* = amperes, *r* = ohms, *p* = watts.

The `smuX.trigger.measure.iv()` function stores current readings in *ibuffer* and voltage readings in *vbuffer*.

If a given reading buffer contains any data, it is cleared before making any measurements, unless the reading buffer has been configured to append data.

The SMU only retains the last call to any one of these functions and only that measurement action is performed.

After configuring the measurements to make with this function, enable the measurement action.

Example

<code>smua.trigger.measure.v(vbuffername)</code> <code>smua.trigger.measure.action = smua.ENABLE</code>	Stores voltage readings during the sweep for SMU channel A in buffer <i>vbuffername</i> .

Also see

[Reading buffers](#) (on page 6-1)
[smuX.measure.Y\(\)](#) (on page 15-228)
[smuX.nvbufferY](#) (on page 15-230)
[smuX.trigger.measure.action](#) (on page 15-257)
[Sweep Operation](#) (on page 5-34)
[Triggering](#) (on page 7-1)
[waitcomplete\(\)](#) (on page 15-407)

smuX.trigger.MEASURE_COMPLETE_EVENT_ID

This constant contains the measurement complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.MEASURE_COMPLETE_EVENT_ID
```

<i>eventID</i>	The measurement complete event number
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.trigger.MEASURE_COMPLETE_EVENT_ID applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to measure complete events from this SMU.

Example

```
trigger.timer[1].stimulus = smuX.trigger.MEASURE_COMPLETE_EVENT_ID  
Trigger the timer when the SMU completes a measurement.
```

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.PULSE_COMPLETE_EVENT_ID

This constant contains the pulse complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.PULSE_COMPLETE_EVENT_ID
```

<i>eventID</i>	The pulse complete event number
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.trigger.PULSE_COMPLETE_EVENT_ID applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to pulse complete events.

Example

```
trigger.timer[1].stimulus = smua.trigger.PULSE_COMPLETE_EVENT_ID  
Trigger a timer when the SMU completes a pulse.
```

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.source.action

This attribute enables or disables sweeping the source (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.DISABLE)

Usage

```
action = smuX.trigger.source.action  
smuX.trigger.source.action = action
```

<i>action</i>	Sweep source action. Set to one of the following values: 0 or smuX.DISABLE: Do not sweep the source 1 or smuX.ENABLE: Sweep the source
<i>X</i>	Source-measure unit (SMU) channel (for example, smua.trigger.source.action applies to SMU channel A)

Details

This attribute is used to enable or disable source level changes during a sweep. In addition to enabling the action before initiating the sweep, make sure to configure it using `smuX.trigger.source.linearY()`, `smuX.trigger.source.listY()`, or `smuX.trigger.source.logY()`.

Example

```
smua.trigger.source.listv({3, 1, 4, 5, 2})  
smua.trigger.source.action = smua.ENABLE
```

Configure list sweep for SMU channel A (sweep through 3 V, 1 V, 4 V, 5 V, and 2 V).
Enable the source action.

Also see

[smuX.trigger.source.linearY\(\)](#) (on page 15-265)
[smuX.trigger.source.listY\(\)](#) (on page 15-266)
[smuX.trigger.source.logY\(\)](#) (on page 15-267)
[Triggering](#) (on page 7-1)

smuX.trigger.source.limitY

This attribute sets the sweep source limit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.LIMIT_AUTO)

Usage

```
sweepSourceLimit = smuX.trigger.source.limitY
smuX.trigger.source.limitY = sweepSourceLimit
```

<i>sweepSourceLimit</i>	The source limit that is used during triggered operation; set to: - A user-defined value - smuX.LIMIT_AUTO
X	Source-measure unit (SMU) channel (for example, smua.trigger.source.limitv applies to SMU channel A)
Y	SMU output function (v = voltage, i = current)

Details

If this attribute is set to `smuX.LIMIT_AUTO` (or 0), the SMU uses the normal limit setting during sweeping. If this attribute is set to any other numeric value, the SMU switches in this limit at the start of the source action and returns to the normal limit setting at the end of the end pulse action.

Normally, the limit range is automatically adjusted in accordance with the limit value. During sweeping, however, the limit range is fixed to avoid the delays associated with changing range. This fixed limit range is determined by the maximum limit value needed during the sweep; that is, the greater of either the normal limit value (as specified by `smuX.source.limitY`) or the sweep limit value (as specified by `smuX.trigger.source.limitY`). The minimum limit value that can be enforced during the sweep is equal to 10% of the full-scale value of the fixed limit range. If the smaller limit value (normal or sweep) falls below this 10% threshold, the 10% value is enforced instead. Likewise, if the limit value falls below the 10% threshold as a result of power compliance, the 10% value is enforced instead.

To disable the source current limit during triggered operation, set the `smuX.trigger.source.limiti` attribute equal to `smuX.LIMIT_OFF` (or "off").

Example

```
smua.trigger.source.limitv = 10
```

Sets the voltage sweep limit of SMU channel A to 10 V.

Also see

[Configuring and running sweeps](#) (on page 5-43)
[smuX.source.limitY](#) (on page 15-239)
[Triggering](#) (on page 7-1)

smuX.trigger.source.linearY()

This function configures a linear source sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.source.linearY(startValue, endValue, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.source.linearv(0, 10, 11)</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)
<i>startValue</i>	Source value of the first point
<i>endValue</i>	Source value of the last point
<i>points</i>	The number of points used to calculate the step size

Details

This function configures the source action to be a linear source sweep in a subsequent sweep. During the sweep, the source generates a uniform series of ascending or descending voltage or current changes called steps. The number of source steps is one less than the number of sourced *points*.

The *points* parameter does not set the number of steps in a sweep. Instead, it is used to calculate source values within a subsequent sweep. If the subsequent sweep has more points than specified in *points*, the source restarts at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU satisfies the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *points*, *endValue* is not reached during the sweep. This means that if the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

In cases where the first sweep point is a nonzero value, it may be necessary to pre-charge the circuit so that the sweep returns a stable value for the first measured point without penalizing remaining points in the sweep.

With linear sweeps, it is acceptable to maintain a fixed source resolution over the entire sweep. To prevent source range changes during the sweep (especially when sweeping through 0.0), set the source range to a fixed range appropriate for the larger of either *startValue* or *endValue*.

The SMU only stores the most recent configured source action. The last call to `smuX.trigger.source.linearY()`, `smuX.trigger.source.listY()`, or `smuX.trigger.source.logY()` is used for the source action.

Source functions cannot be changed within a sweep.

After configuring the sweep source values, enable the source action by setting `smuX.trigger.source.action`.

Example

<code>smua.trigger.source.linearv(0, 10, 11)</code>	Sweeps from 0 V to 10 V in 1 V steps.

Also see

[smuX.trigger.source.action](#) (on page 15-263)

[smuX.trigger.source.logY\(\)](#) (on page 15-267)

[Sweep Operation](#) (on page 5-34)

smuX.trigger.source.listY()

This function configures an array-based source sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.source.listY(sweepList)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.source.listv({5})</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)
<i>sweepList</i>	An array of source values

Details

This function configures the source action to be a list sweep in a subsequent sweep. During the sweep, the source outputs the sequence of source values given in the *sweepList* array.

If the subsequent sweep has more points than specified in *sweepList*, the source restarts at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU satisfies the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *sweepList*, the extra values are ignored. This means that if the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

In cases where the first sweep point is a nonzero value, it may be necessary to pre-charge the circuit so that the sweep returns a stable value for the first measured point without penalizing remaining points in the sweep.

The SMU only stores the most recent configured source action. The last call to `smuX.trigger.source.linearY()`, `smuX.trigger.source.listY()`, or `smuX.trigger.source.logY()` is used for the source action.

Source functions cannot be changed within a sweep.

After configuring the sweep source values, enable the source action by setting `smuX.trigger.source.action`.

Example

<code>smua.trigger.source.listv({3, 1, 4, 5, 2})</code>	Sweeps SMU channel A through 3 V, 1 V, 4 V, 5 V, and 2 V.

Also see

[smuX.trigger.source.action](#) (on page 15-263)
[smuX.trigger.source.linearY\(\)](#) (on page 15-265)
[smuX.trigger.source.logY\(\)](#) (on page 15-267)
[Sweep operation](#) (on page 5-34)

smuX.trigger.source.logY()

This function configures an exponential (geometric) source sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.source.logY(startValue, endValue, points, asymptote)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.source.logv(1, 10, 11, 0)</code> applies to SMU channel A)
<i>Y</i>	SMU source function (<i>v</i> = voltage, <i>i</i> = current)
<i>startValue</i>	Source value of the first point
<i>endValue</i>	Source value of the last point
<i>points</i>	The number of points used to calculate the step size
<i>asymptote</i>	The asymptotic offset value

Details

This function configures the source action to be a geometric source sweep in a subsequent sweep. During the sweep, the source generates a geometric series of ascending or descending voltage or current changes called steps. Each step is larger or smaller than the previous step by a fixed proportion. The constant of proportionality is determined by the starting value, the ending value, the asymptote, and the number of steps in the sweep. The number of source steps is one less than the number of sourced *points*.

The *points* parameter does not set the number of steps in a sweep, but rather is used to calculate source values within a subsequent sweep. If the subsequent sweep has more points than specified in *points*, the source restarts at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU satisfies the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *points*, *endValue* is not reached during the sweep. This means that if the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

In cases where the first sweep point is nonzero, it may be necessary to pre-charge the circuit so that the sweep returns a stable value for the first measured point without penalizing remaining points in the sweep.

With logarithmic sweeps, it is usually necessary to allow the source to autorange to maintain good source accuracy when sweeping over more than one decade or across range boundaries.

The *asymptote* parameter customizes the inflection and offset of the source value curve. This allows log sweeps to cross zero. 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.

The SMU stores only the most recent configured source action. The last call to `smuX.trigger.source.linearY()`, `smuX.trigger.source.listY()`, or `smuX.trigger.source.logY()` is used for the source action.

Source functions cannot be changed within a sweep.

After configuring the sweep source values, enable the source action by setting `smuX.trigger.source.action`.

Example

```
smua.trigger.source.logv(1, 10, 11, 0)
```

Sweeps SMU channel A from 1 V to 10 V in 10 steps with an asymptote of 0 V.

Also see

[smuX.trigger.source.action](#) (on page 15-263)
[smuX.trigger.source.linearY\(\)](#) (on page 15-265)
[smuX.trigger.source.listY\(\)](#) (on page 15-266)
[Sweep operation](#) (on page 5-34)

smuX.trigger.source.set()

This function sets the source event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.source.set()
```

X

Source-measure unit (SMU) channel (for example, `smua.trigger.source.set()` applies to SMU channel A)

Details

This function sets the source event detector to the detected state.

The SMU automatically clears all event detectors when the `smuX.trigger.initiate()` function is executed. Call this function after the sweep is initiated. If the event detectors are configured to clear automatically because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`, make sure that `smuX.trigger.source.set()` is issued after the SMU has entered the trigger layer.

Example

```
reset()
smua.trigger.source.listv({5})
smua.trigger.source.stimulus = display.trigger.EVENT_ID
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
delay(1)
-- Continue even if the display trigger key was not pressed.
smua.trigger.source.set()
waitcomplete()
```

Sets the source event detector.

Also see

[smuX.trigger.arm.set\(\)](#) (on page 15-247)
[smuX.trigger.autoclear](#) (on page 15-250)
[smuX.trigger.endpulse.set\(\)](#) (on page 15-252)
[smuX.trigger.initiate\(\)](#) (on page 15-256)
[smuX.trigger.measure.set\(\)](#) (on page 15-258)
[Triggering](#) (on page 7-1)

smuX.trigger.source.stimulus

This attribute defines which event causes the source event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```

eventID = smuX.trigger.source.stimulus
smuX.trigger.source.stimulus = eventID

```

<i>eventID</i>	Set to the event that triggers the end pulse source off action
X	Source-measure (SMU) channel (for example, smua.trigger.source.stimulus applies to SMU channel A)

Details

Set this attribute to the event ID of any trigger event generator to wait for that event. When set, the SMU waits for the event at the source event detector portion of the trigger model. To bypass waiting for an event, set the value of this attribute to zero (0). Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	Event description
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
display.trigger.EVENT_ID	Occurs when the TRIG key on the virtual front panel is pressed

Trigger event IDs*	
Event ID	Event description
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Example

```
smua.trigger.source.stimulus = digio.trigger[2].EVENT_ID
```

Configure SMU channel A to start its source action when a trigger event occurs on digital I/O line 2.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.SOURCE_COMPLETE_EVENT_ID

This constant contains the source complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.SOURCE_COMPLETE_EVENT_ID
```

<code>eventID</code>	The source action complete event number
<code>X</code>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.SOURCE_COMPLETE_EVENT_ID</code> applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to source complete events from this source-measure unit (SMU).

Example

```
trigger.timer[1].stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID
```

Trigger the timer when the SMU updates the source level or starts a pulse.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.SWEEP_COMPLETE_EVENT_ID

This constant contains the sweep complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.SWEEP_COMPLETE_EVENT_ID
```

<i>eventID</i>	The sweep complete event number
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.SWEEP_COMPLETE_EVENT_ID</code> applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to sweep complete events from this SMU.

Example

```
digio.trigger[2].mode = digio.TRIG_RISINGA
digio.trigger[2].clear()
smua.trigger.source.stimulus = digio.trigger[2].EVENT_ID
digio.trigger[4].mode = digio.TRIG_RISINGM
digio.trigger[4].pulsewidth = 0.001
digio.trigger[4].stimulus = smua.trigger.SWEEP_COMPLETE_EVENT_ID
```

Configure the 2606B to detect a rising edge on digital I/O line 2.
Configure SMU A to start its source action when a trigger event occurs on digital I/O line 2.
Configure digital I/O line 4 to output a 1 ms rising-edge trigger pulse at the completion of the SMU sweep.

Also see

[Triggering](#) (on page 7-1)

smuX.trigger.SWEEPING_EVENT_ID

This constant contains the sweeping event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.SWEEPING_EVENT_ID
```

<i>eventID</i>	The sweeping event number
<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.trigger.SWEEPING_EVENT_ID</code> applies to SMU channel A)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to sweeping events from this SMU.

Example

```
reset()
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
smua.trigger.source.listv( {5} )
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
pulse_timer.delay = 0.0006
pulse_timer.stimulus = period_timer.EVENT_ID
pulse_timer.count = 1
period_timer.delay = 0.005
period_timer.count = 9
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
period_timer.passthrough = true
smua.trigger.source.stimulus = period_timer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
smua.trigger.count = 1
smua.trigger.arm.count = 10
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
```

Generate a 10-point pulse train where each pulse has a width of 600 μ s and a pulse period of 5 ms.

Alias the trigger timers to use for pulse width and period.

Create a fixed level voltage sweep.

Set the pulse width and trigger the pulse width timer with a period timer.

Set the pulse period to output one pulse per period and the count to generate 10 pulses.

Trigger the pulse period timer when a sweep is initiated.

Configure the timer to output a trigger event when it starts the first delay.

Trigger the SMU source action using pulse period timer.

Trigger the SMU end pulse action using pulse width timer.

Set the trigger model counts.

Configure the SMU to execute a 10-point pulse train.

Start the trigger model.

Wait for the sweep to complete.

Also see

[Triggering](#) (on page 7-1)

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 saved	Not applicable

Usage

```
statusByte = status.condition
```

<i>statusByte</i>	The status byte; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the status byte, which is returned as a numeric value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of 1.29000e+02 (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

The returned value can indicate one or more status events occurred. When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence.

The individual bits of this register have the following meanings:

Bit	Value	Description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB	Set summary bit indicates that an enabled measurement event has occurred. Bit B0 decimal value: 1
B1	status.SYSTEM_SUMMARY_BIT status.SSB	Set summary bit indicates that an enabled system event has occurred. Bit B1 decimal value: 2
B2	status.ERROR_AVAILABLE status.EAV	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB	Set summary bit indicates that an enabled questionable event has occurred. Bit B3 decimal value: 8
B4	status.MESSAGE_AVAILABLE status.MAV	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	status.EVENT_SUMMARY_BIT status.ESB	Set summary bit indicates that an enabled standard event has occurred. Bit B5 decimal value: 32

Bit	Value	Description
B6	<code>status.MASTER_SUMMARY_STATUS</code> <code>status.MSS</code>	Request Service (RQS)/Master Summary Status (MSS). Depending on how it is used, bit B6 of the status byte register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit: ■ When using the USB or VXI-11 serial poll sequence of the 2606B to obtain the status byte (serial poll byte), B6 is the RQS bit. The set bit indicates that the Request Service (RQS) bit of the status byte (serial poll byte) is set and a service request (SRQ) has occurred. ■ When using the <code>status.condition</code> register command or the <code>*STB?</code> common command to read the status byte, B6 is the MSS bit. Set bit indicates that an enabled summary bit of the status byte register is set. Bit B6 decimal value: 64
B7	<code>status.OPERATION_SUMMARY_BIT</code> <code>status.OSB</code>	Set summary bit indicates that an enabled operation event has occurred. Bit B7 decimal value: 128

In addition to the above constants, when 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).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Example

```
statusByte = status.condition
print(statusByte)
```

Returns `statusByte`.

Sample output:

```
1.290000e+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

[Status byte and service request \(SRQ\)](#) (on page 10-14)

status.measurement.*

This attribute contains the measurement event register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	10,627 (All bits set)

Usage

```

measurementRegister = status.measurement.condition
measurementRegister = status.measurement.enable
measurementRegister = status.measurement.event
measurementRegister = status.measurement.ntr
measurementRegister = status.measurement.ptr
status.measurement.enable = measurementRegister
status.measurement.ntr = measurementRegister
status.measurement.ptr = measurementRegister

```

<i>measurementRegister</i>	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes read or write the measurement event registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, assume value 257 is returned for the enable register. The binary equivalent is 0000 0001 0000 0001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	<code>status.measurement.VOLTAGE_LIMIT</code> <code>status.measurement.VLMT</code>	Set bit is a summary of the <code>status.measurement.voltage_limit</code> register. Bit B0 decimal value: 1
B1	<code>status.measurement.CURRENT_LIMIT</code> <code>status.measurement.ILMT</code>	Set bit is a summary of the <code>status.measurement.current_limit</code> register. Bit B1 decimal value: 2
B2 to B6	Not used	Not applicable
B7	<code>status.measurement.READING_OVERFLOW</code> <code>status.measurement.ROF</code>	Set bit is a summary of the <code>status.measurement.reading_overflow</code> register. Bit B7 decimal value: 128
B8	<code>status.measurement.BUFFER_AVAILABLE</code> <code>status.measurement.BAV</code>	Set bit is a summary of the <code>status.measurement.buffer_available</code> register. Bit B8 decimal value: 256
B9 to B10	Not used	Not applicable
B11	<code>status.measurement.OUTPUT_ENABLE</code> <code>status.measurement.OE</code>	Output enable line. Set bit indicates that output enable has been asserted. Bit B11 decimal value: 2,048
B12	Not used	Not applicable
B13	<code>status.measurement.INSTRUMENT_SUMMARY</code> <code>status.measurement.INST</code>	Set bit indicates that a bit in the measurement instrument summary register is set. Bit B13 decimal value: 8,192
B14 to B15	Not used	Not applicable

As an example, to set bit B8 of the measurement event enable register, set
`status.measurement.enable = status.measurement.BAV`.

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B8, set *measurementRegister* to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example

```
status.measurement.enable = status.measurement.BAV
```

Sets the BAV bit of the measurement event enable register.

Also see

[Measurement event registers](#) (on page 10-22)

status.measurement.buffer_available.*

This attribute contains the measurement event buffer available summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
measurementRegister = status.measurement.buffer_available.condition
measurementRegister = status.measurement.buffer_available.enable
measurementRegister = status.measurement.buffer_available.event
measurementRegister = status.measurement.buffer_available.ntr
measurementRegister = status.measurement.buffer_available.ptr
status.measurement.buffer_available.enable = measurementRegister
status.measurement.buffer_available.ntr = measurementRegister
status.measurement.buffer_available.ptr = measurementRegister
```

<i>measurementRegister</i>	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the measurement event buffer available summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	<code>status.measurement.buffer_available.SMUA</code>	Set bit indicates that there is at least one reading stored in either or both of the dedicated reading buffers. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	<code>status.measurement.buffer_available.SMUB</code>	Set bit indicates that there is at least one reading stored in either or both of the dedicated reading buffers. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event buffer available summary enable register, set `status.measurement.buffer_available.enable = status.measurement.buffer_available.SMUA`.

In addition to the above constants, `measurementRegister` can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set `measurementRegister` to the sum of their decimal weights. For example, to set bits B1 and B2, set `measurementRegister` to 6 (which is the sum of 2 + 4).

Example

```
status.measurement.buffer_available.enable =
status.measurement.buffer_available.SMUA
```

Sets the SMUA bit of the measurement event buffer available summary enable register.

Also see

[Measurement event registers](#) (on page 10-22)

status.measurement.current_limit.*

This attribute contains the measurement event current limit summary registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
measurementRegister = status.measurement.current_limit.condition
measurementRegister = status.measurement.current_limit.enable
measurementRegister = status.measurement.current_limit.event
measurementRegister = status.measurement.current_limit.ntr
measurementRegister = status.measurement.current_limit.ptr
status.measurement.current_limit.enable = measurementRegister
status.measurement.current_limit.ntr = measurementRegister
status.measurement.current_limit.ptr = measurementRegister
```

<i>measurementRegister</i>	The status of the measurement event current limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the measurement event current limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, assume value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.measurement.current_limit.SMUA	Set bit indicates that the SMU A current limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.measurement.current_limit.SMUB	Set bit indicates that the SMU B current limit was exceeded. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event current limit summary enable register, set `status.measurement.current_limit.enable = status.measurement.current_limit.SMUA`.

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.measurement.current_limit.enable = status.measurement.current_limit.SMUA
Sets the SMUA bit of the Measurement Event Current Limit Summary Enable Register.
```


Also see

[Measurement event registers](#) (on page 10-22)

[status.measurement.instrument.smuX.*](#) (on page 15-281)

status.measurement.instrument.*

This attribute contains the registers of the measurement event instrument summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```

measurementRegister = status.measurement.instrument.condition
measurementRegister = status.measurement.instrument.enable
measurementRegister = status.measurement.instrument.event
measurementRegister = status.measurement.instrument.ntr
measurementRegister = status.measurement.instrument.ptr
status.measurement.instrument.enable = measurementRegister
status.measurement.instrument.ntr = measurementRegister
status.measurement.instrument.ptr = measurementRegister

```

<i>measurementRegister</i>	The status of the measurement event instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the measurement event instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	<code>status.measurement.instrument.SMUA</code>	Set bit indicates one or more enabled bits of the measurement event SMU A summary register is set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	<code>status.measurement.instrument.SMUB</code>	Set bit indicates one or more enabled bits of the measurement event SMU B summary register is set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event instrument summary enable register, set `status.measurement.instrument.enable = status.measurement.instrument.SMUA`.

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.measurement.instrument.enable =
    status.measurement.instrument.SMUA
```

Sets the SMU A bit of the measurement event instrument summary enable register using a constant.

Also see

[Measurement event registers](#) (on page 10-22)

status.measurement.instrument.smuX.*

This attribute contains the registers of the measurement event SMU X summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	387 (All bits set)

Usage

```
measurementRegister = status.measurement.instrument.smuX.condition
measurementRegister = status.measurement.instrument.smuX.enable
measurementRegister = status.measurement.instrument.smuX.event
measurementRegister = status.measurement.instrument.smuX.ntr
measurementRegister = status.measurement.instrument.smuX.ptr
status.measurement.instrument.smuX.enable = measurementRegister
status.measurement.instrument.smuX.ntr = measurementRegister
status.measurement.instrument.smuX.ptr = measurementRegister
```

<i>measurementRegister</i>	The status of the instrument measurement status SMU X summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
<i>X</i>	Source-measure unit (SMU) channel (for example <code>status.measurement.instrument.smuA.enable</code> applies to SMU channel A)

Details

These attributes are used to read or write to the measurement event SMU X summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 257 is returned for the enable register. The binary equivalent is 0000 0001 0000 0001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about `.condition`, `.enable`, `.event`, `.ntr`, and `.ptr` registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0*	<code>status.measurement.instrument.smuX.VOLTAGE_LIMIT</code> <code>status.measurement.instrument.smuX.VLMT</code>	Set bit indicates that the voltage limit was exceeded. Bit B0 decimal value: 1
B1*	<code>status.measurement.instrument.smuX.CURRENT_LIMIT</code> <code>status.measurement.instrument.smuX.ILMT</code>	Set bit indicates that the current limit was exceeded. Bit B1 decimal value: 2
B2 to B6	Not used	Not applicable.
B7	<code>status.measurement.instrument.smuX.READING_OVERFLOW</code> <code>status.measurement.instrument.smuX.ROF</code>	Set bit indicates that an overflow reading has been detected. Bit B7 decimal value: 128
B8	<code>status.measurement.instrument.smuX.BUFFER_AVAILABLE</code> <code>status.measurement.instrument.smuX.BAV</code>	Set bit indicates that there is at least one reading stored in either or both dedicated reading buffers. Bit B8 decimal value: 256
B9 to B15	Not used	Not applicable.

* This bit is updated only when a measurement is made or `smuX.source.compliance` is invoked.

As an example, to set bit B0 of the measurement event SMU X summary enable register, set

```
status.measurement.instrument.smuA.enable =
```

```
status.measurement.instrument.smuA.VLMT.
```

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B8, set *measurementRegister* to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example

```
status.measurement.instrument.smuA.enable = status.measurement.instrument.smuA.VLMT
```

Sets the VLMT bit of the measurement event SMU A summary enable register using a constant.

Also see

[Measurement event registers](#) (on page 10-22)

status.measurement.reading_overflow.*

This attribute contains the measurement event reading overflow summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
measurementRegister = status.measurement.reading_overflow.condition
measurementRegister = status.measurement.reading_overflow.enable
measurementRegister = status.measurement.reading_overflow.event
measurementRegister = status.measurement.reading_overflow.ntr
measurementRegister = status.measurement.reading_overflow.ptr
status.measurement.reading_overflow.enable = measurementRegister
status.measurement.reading_overflow.ntr = measurementRegister
status.measurement.reading_overflow.ptr = measurementRegister
```

<i>measurementRegister</i>	The status of the measurement reading overflow summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the measurement event reading overflow summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 (SMUA) is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.measurement.reading_overflow.SMUA	Set bit indicates that an overflow reading has been detected for SMU A. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.measurement.reading_overflow.SMUB	Set bit indicates that an overflow reading has been detected for SMU B. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event reading overflow summary enable register, set `status.measurement.reading_overflow.enable = status.measurement.reading_overflow.SMUA`.

In addition to the above constants, *measurementRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.measurement.reading_overflow.enable =
status.measurement.reading_overflow.SMUA
```

Sets the SMU A bit of the measurement reading overflow summary enable register using a constant.

Also see

[Measurement event registers](#) (on page 10-22)

status.measurement.voltage_limit.*

This attribute contains the measurement event voltage limit summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
measurementRegister = status.measurement.voltage_limit.condition
measurementRegister = status.measurement.voltage_limit.enable
measurementRegister = status.measurement.voltage_limit.event
measurementRegister = status.measurement.voltage_limit.ntr
measurementRegister = status.measurement.voltage_limit.ptr
status.measurement.voltage_limit.enable = measurementRegister
status.measurement.voltage_limit.ntr = measurementRegister
status.measurement.voltage_limit.ptr = measurementRegister
```

<i>measurementRegister</i>	The status of the measurement voltage limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other decimal values indicate various bit settings

Details

These attributes read or write to the measurement event voltage limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	<code>status.measurement.voltage_limit.SMUA</code>	Set bit indicates the enabled VLMT bit for the SMU A measurement register is set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	<code>status.measurement.voltage_limit.SMUB</code>	Set bit indicates the enabled VLMT bit for the SMU B measurement register is set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event voltage limit summary enable register, set `status.measurement.voltage_limit.enable = status.measurement.voltage_limit.SMUA`.

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.measurement.voltage_limit.enable = status.measurement.voltage_limit.SMUA
```

Sets the SMUA bit of the measurement event voltage limit summary enable register using a constant.

Also see

[Measurement event registers](#) (on page 10-22)

status.node_enable

This attribute stores the system node enable register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
nodeEnableRegister = status.node_enable
status.node_enable = nodeEnableRegister
```

<i>nodeEnableRegister</i>	The status of the system node enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

This attribute is used to read or write to the system node enable register. Reading the system node enable register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, a summary bit is set in the appropriate system summary register. The register and bit that is set depends on the TSP-Link node number assigned to this instrument.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB Set summary bit indicates that an enabled measurement event has occurred. Bit B0 decimal value: 1
B1	Not used
B2	status.ERROR_AVAILABLE status.EAV Set summary bit indicates that an error or status message is present in the error queue. Bit B2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB Set summary bit indicates that an enabled questionable event has occurred. Bit B3 decimal value: 8
B4	status.MESSAGE_AVAILABLE status.MAV Set summary bit indicates that a response message is present in the output queue. Bit B4 decimal value: 16
B5	status.EVENT_SUMMARY_BIT status.ESB Set summary bit indicates that an enabled standard event has occurred. Bit B5 decimal value: 32
B6	status.MASTER_SUMMARY_STATUS status.MSS Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set. Bit B6 decimal value: 64
B7	status.OPERATION_SUMMARY_BIT status.OSB Set summary bit indicates that an enabled operation event has occurred. Bit B7 decimal value: 128

As an example, to set the B0 bit of the system node enable register, set
`status.node_enable = status.MSB.`

In addition to the above values, *nodeEnableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *nodeEnableRegister* to the sum of their decimal weights. For example, to set bits B0 and B7, set *nodeEnableRegister* to 129 (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Example 1

```
nodeEnableRegister = status.MSB + status.OSB
status.node_enable = nodeEnableRegister
```

Use constants to set the MSB and OSB bits of the system node enable register.

Example 2

```
-- decimal 129 = binary 10000001
nodeEnableRegister = 129
status.node_enable = nodeEnableRegister
```

Sets the MSB and OSB bits of the system node enable register using a decimal value.

Also see

[status.condition](#) (on page 15-273)

[status.system.*](#) (on page 15-342)

[Status byte and service request \(SRQ\)](#) (on page 10-14)

status.node_event

This attribute stores the status node event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

```
nodeEventRegister = status.node_event
```

```
nodeEventRegister
```

The status of the node event register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the status node event register, which is returned as a numeric value (reading this register returns a value). The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of 1.29000e+02 (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

The returned value can indicate one or more status events occurred.

Bit	Value and description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB Set summary bit indicates that an enabled measurement event has occurred. Bit B0 decimal value: 1
B1	Not used
B2	status.ERROR_AVAILABLE status.EAV Set summary bit indicates that an error or status message is present in the error queue. Bit B2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB Set summary bit indicates that an enabled questionable event has occurred. Bit B3 decimal value: 8
B4	status.MESSAGE_AVAILABLE status.MAV Set summary bit indicates that a response message is present in the output queue. Bit B4 decimal value: 16
B5	status.EVENT_SUMMARY_BIT status.ESB Set summary bit indicates that an enabled standard event has occurred. Bit B5 decimal value: 32
B6	status.MASTER_SUMMARY_STATUS status.MSS Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte register is set. Bit B6 decimal value: 64
B7	status.OPERATION_SUMMARY_BIT status.OSB Set summary bit indicates that an enabled operation event has occurred. Bit B7 decimal value: 128

In addition to the above constants, *nodeEventRegister* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *nodeEventRegister* contains the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Example

```
nodeEventRegister = status.node_event
print(nodeEventRegister)
```

Reads the status node event register.

Sample 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 (MSB) and B7 (OSB).

Also see

[Status byte and service request \(SRQ\)](#) (on page 10-14)

[status.condition](#) (on page 15-273)

[status.system.*](#) (on page 15-342)

status.operation.*

These attributes manage the operation status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	31,769 (All bits set)

Usage

```
operationRegister = status.operation.condition
operationRegister = status.operation.enable
operationRegister = status.operation.event
operationRegister = status.operation.ntr
operationRegister = status.operation.ptr
status.operation.enable = operationRegister
status.operation.ntr = operationRegister
status.operation.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes read or write the operation status registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 2.04800e+04 (which is 20,480) is read as the value of the condition register, the binary equivalent is 0101 0000 0000 0000. This value indicates that bit B14 (PROGRAM_RUNNING) and bit B12 (USER) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	<code>status.operation.CALIBRATING</code> <code>status.operation.CAL</code> Set bit indicates that the summary bit of the <code>status.operation.calibrating</code> register has been set. Bit B0 decimal value: 1
B1 to B2	Not used
B3	<code>status.operation.SWEEPING</code> <code>status.operation.SWE</code> Set bit indicates that the summary bit from the <code>status.operation.sweeping</code> register is set. Bit B3 decimal value: 8
B4	<code>status.operation.MEASURING</code> <code>status.operation.MEAS</code> Set bit indicates that the summary bit of the <code>status.operation.measuring</code> register is set. Bit B4 decimal value: 16
B5 to B9	Not used
B10	<code>status.operation.TRIGGER_OVERRUN</code> <code>status.operation.TRGOVR</code> Set bit indicates that the summary bit from the <code>status.operation.trigger_overrun</code> register is set. Bit B10 decimal value: 1,024
B11	<code>status.operation.REMOTE_SUMMARY</code> <code>status.operation.REM</code> Set bit indicates that the summary bit of the <code>status.operation.remote</code> register is set. Bit B11 decimal value: 2,048
B12	<code>status.operation.USER</code> Set bit indicates that the summary bit from the <code>status.operation.user</code> register is set. Bit B12 decimal value: 4,096
B13	<code>status.operation.INSTRUMENT_SUMMARY</code> <code>status.operation.INST</code> Set bit indicates that the summary bit from the <code>status.operation.instrument</code> register is set. Bit B13 decimal value: 8,192
B14	<code>status.operation.PROGRAM_RUNNING</code> <code>status.operation.PROG</code> Set bit indicates that a command or program is running. Bit B14 decimal value: 16,384
B15	Not used

As an example, to set bit B12 of the operation status enable register, set `status.operation.enable = status.operation.USER`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B12 and B14, set *operationRegister* to 20,480 (which is the sum of 4,096 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.operation.USER + status.operation.PROG
status.operation.enable = operationRegister
```

Uses constants to set the USER and PROG bits of the operation status enable register.

Example 2

```
-- decimal 20480 = binary 0101 0000 0000 0000
operationRegister = 20480
status.operation.enable = operationRegister
```

Uses a decimal value to set the USER and PROG bits of the operation status enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

status.operation.calibrating.*

This attribute contains the operation status calibration summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
operationRegister = status.operation.calibrating.condition
operationRegister = status.operation.calibrating.enable
operationRegister = status.operation.calibrating.event
operationRegister = status.operation.calibrating.ntr
operationRegister = status.operation.calibrating.ptr
status.operation.calibrating.enable = operationRegister
status.operation.calibrating.ntr = operationRegister
status.operation.calibrating.ptr = operationRegister
```

operationRegister The status of the operation calibrating event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status calibration summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.operation.calibrating.SMUA	Set bit indicates that SMU A is unlocked for calibration. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.operation.calibrating.SMUB	Set bit indicates that SMU B is unlocked for calibration. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

Example

```
status.operation.calibrating.enable = status.operation.calibrating.SMUA
```

Sets the SMUA bit of the operation status calibration summary enable register using a constant.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.*](#) (on page 15-290)

status.operation.instrument.*

This attribute contains the operation status instrument summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	31,750 (All bits set)

Usage

```
operationRegister = status.operation.instrument.condition
operationRegister = status.operation.instrument.enable
operationRegister = status.operation.instrument.event
operationRegister = status.operation.instrument.ntr
operationRegister = status.operation.instrument.ptr
status.operation.instrument.enable = operationRegister
status.operation.instrument.ntr = operationRegister
status.operation.instrument.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 1.02600e+03 (which is 1,026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.operation.instrument.SMUA	Set bit indicates one or more enabled bits for the operation status SMU A summary register is set. Bit B1 decimal value: 2
B2	status.operation.instrument.SMUB	Set bit indicates one or more enabled bits for the operation status SMU B summary register is set. Bit B2 decimal value: 4
B3 to B9	Not used	Not applicable.
B10	status.operation.instrument.TRIGGER_BLENDER status.operation.instrument.TRGBLND	Set bit indicates one or more enabled bits for the operation status trigger blender summary register is set. Bit B10 decimal value: 1,024.
B11	status.operation.instrument.TRIGGER_TIMER status.operation.instrument.TRGTMR	Set bit indicates one or more enabled bits for the operation status trigger timer summary register is set. Bit B11 decimal value: 2,048

Bit	Value	Description
B12	<code>status.operation.instrument.DIGITAL_IO</code> <code>status.operation.instrument.DIGIO</code>	Set bit indicates one or more enabled bits for the operation status digital I/O summary register is set. Bit B12 decimal value: 4,096
B13	<code>status.operation.instrument.TSPLINK</code>	Set bit indicates one or more enabled bits for the operation status TSP-Link summary register is set. Bit B13 decimal value: 8,192
B14	<code>status.operation.instrument.LAN</code>	Set bit indicates one or more enabled bits for the operation status LAN summary register is set. Bit B14 decimal value: 16,384
B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status instrument summary enable register, set `status.operation.instrument.enable = status.operation.instrument.SMUA`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B10, set *operationRegister* to 1,026 (which is the sum of 2 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.operation.instrument.SMUA +
    status.operation.instrument.TRGBLND
status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B10 of the operation status instrument summary enable register using constants.

Example 2

```
-- 1026 = binary 0000 0100 0000 0010
operationRegister = 1026
status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B10 of the operation status instrument summary enable register using a decimal value.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.*](#) (on page 15-290)

Condition register sets of:

- [status.operation.instrument.digio.*](#) (on page 15-296)
- [status.operation.instrument.lan.*](#) (on page 15-299)
- [status.operation.instrument.trigger_blender.*](#) (on page 15-308)
- [status.operation.instrument.trigger_timer.*](#) (on page 15-311)
- [status.operation.instrument.tsplink.*](#) (on page 15-315)

status.operation.instrument.digio.*

This attribute contains the operation status digital I/O summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.digio.condition
operationRegister = status.operation.instrument.digio.enable
operationRegister = status.operation.instrument.digio.event
operationRegister = status.operation.instrument.digio.ntr
operationRegister = status.operation.instrument.digio.ptr
status.operation.instrument.digio.enable = operationRegister
status.operation.instrument.digio.ntr = operationRegister
status.operation.instrument.digio.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status digital I/O summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status digital I/O summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used
B10	status.operation.instrument.digio.TRIGGER_OVERRUN status.operation.instrument.digio.TRGOVR Set bit indicates an enabled bit in the Operation Status Digital I/O Overrun Register is set. Bit B10 decimal value: 1,024 Binary value: 0100 0000 0010
B11 to B15	Not used

In addition to the above constant, *operationRegister* can be set to the decimal value of the bit to set.

Example 1

```
status.operation.instrument.digio.enable = status.operation.instrument.digio.TRGOVR
```

Uses a constant to set the TRGOVR bit of the operation status digital I/O summary enable register.

Example 2

```
status.operation.instrument.digio.enable = 1024
```

Uses the decimal value to set the TRGOVR bit of the operation status digital I/O summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.digio.trigger_overrun.*](#) (on page 15-297)

status.operation.instrument.digio.trigger_overrun.*

This attribute contains the operation status digital I/O overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,766 (All bits set)

Usage

```
operationRegister = status.operation.instrument.digio.trigger_overrun.condition
operationRegister = status.operation.instrument.digio.trigger_overrun.enable
operationRegister = status.operation.instrument.digio.trigger_overrun.event
operationRegister = status.operation.instrument.digio.trigger_overrun.ntr
operationRegister = status.operation.instrument.digio.trigger_overrun.ptr
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
status.operation.instrument.digio.trigger_overrun.ntr = operationRegister
status.operation.instrument.digio.trigger_overrun.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status digio I/O overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status digital I/O overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of $1.02600e+03$ (which is 1026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

A set bit indicates that the specified digital I/O line generated an action overrun when it was triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	status.operation.instrument.digio.trigger_overrun.LINE1	2
B2	status.operation.instrument.digio.trigger_overrun.LINE2	4
B3	status.operation.instrument.digio.trigger_overrun.LINE3	8
B4	status.operation.instrument.digio.trigger_overrun.LINE4	16
B5	status.operation.instrument.digio.trigger_overrun.LINE5	32
B6	status.operation.instrument.digio.trigger_overrun.LINE6	64
B7	status.operation.instrument.digio.trigger_overrun.LINE7	128
B8	status.operation.instrument.digio.trigger_overrun.LINE8	256
B9	status.operation.instrument.digio.trigger_overrun.LINE9	512
B10	status.operation.instrument.digio.trigger_overrun.LINE10	1,024
B11	status.operation.instrument.digio.trigger_overrun.LINE11	2,048
B12	status.operation.instrument.digio.trigger_overrun.LINE12	4,096
B13	status.operation.instrument.digio.trigger_overrun.LINE13	8,192
B14	status.operation.instrument.digio.trigger_overrun.LINE14	16,384
B15	Not used	Not applicable

As an example, to set bit B1 of the operation status digital I/O overrun enable register, set
`status.operation.instrument.digio.trigger_overrun.enable =`
`status.operation.instrument.digio.trigger_overrun.LINE1.`

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal values. For example, to set bits B1 and B10, set *operationRegister* to 1,026 (which is the sum of 2 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.operation.instrument.digio.trigger_overrun.LINE1 +
    status.operation.instrument.digio.trigger_overrun.LINE10
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
```

Uses constants to set bit B1 and bit B10 of the operation status digital I/O overrun enable register.

Example 2

```
operationRegister = 1026
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
```

Uses the decimal value to set bit B1 and bit B10 of the operation status digital I/O overrun enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.digio.*](#) (on page 15-296)

status.operation.instrument.lan.*

This attribute contains the operation status LAN summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1027 (All bits set)

Usage

```
operationRegister = status.operation.instrument.lan.condition
operationRegister = status.operation.instrument.lan.enable
operationRegister = status.operation.instrument.lan.event
operationRegister = status.operation.instrument.lan.ntr
operationRegister = status.operation.instrument.lan.ptr
status.operation.instrument.lan.enable = operationRegister
status.operation.instrument.lan.ntr = operationRegister
status.operation.instrument.lan.ptr = operationRegister
```

operationRegister

The status of the operation status LAN summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status LAN summary registers. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.02600e+03 (which is 1026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	status.operation.instrument.lan.CONNECTION status.operation.instrument.lan.CON Set bit indicates that the LAN cable is connected and a link has been detected. Bit B0 decimal value: 1
B1	status.operation.instrument.lan.CONFIGURING status.operation.instrument.lan.CONF Set bit indicates the LAN is performing its configuration sequence. Bit B1 decimal value: 2
B2 to B9	Not used
B10	status.operation.instrument.lan.TRIGGER_OVERRUN status.operation.instrument.lan.TRGOVR Set bit indicates one or more enabled bits for the operation status LAN trigger overrun register is set. Bit B10 decimal value: 1,024
B11 to B15	Not used

As an example, to set bit B0 of the operation status LAN summary enable register, set
status.operation.instrument.lan.enable =
status.operation.instrument.lan.CON.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B10, set *operationRegister* to 1,026 (which is the sum of 2 + 1024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.operation.instrument.lan.CONF +
    status.operation.instrument.lan.TRGOVR
status.operation.instrument.lan.enable = operationRegister
```

Use constants to set bit B1 and bit B10 of the operation status LAN summary enable register.

Example 2

```
operationRegister = 1026
status.operation.instrument.lan.enable = operationRegister
```

Use the decimal value to set bit B1 and bit B10 of the operation status LAN summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.lan.trigger_overrun.*](#) (on page 15-301)

status.operation.instrument.lan.trigger_overrun.*

This attribute contains the operation status LAN trigger overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```
operationRegister = status.operation.instrument.lan.trigger_overrun.condition
operationRegister = status.operation.instrument.lan.trigger_overrun.enable
operationRegister = status.operation.instrument.lan.trigger_overrun.event
operationRegister = status.operation.instrument.lan.trigger_overrun.ntr
operationRegister = status.operation.instrument.lan.trigger_overrun.ptr
status.operation.instrument.lan.trigger_overrun.enable = operationRegister
status.operation.instrument.lan.trigger_overrun.ntr = operationRegister
status.operation.instrument.lan.trigger_overrun.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status LAN trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status LAN trigger overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 2.58000e+02 (which is 258) is read as the value of the condition register, the binary equivalent is 0000 0001 0000 0010. This value indicates that bit B1 and bit B8 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

A set bit indicates that the specified LAN trigger generated an action overrun when triggered to generate a trigger packet.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	status.operation.instrument.lan.trigger_overrun.LAN1	2
B2	status.operation.instrument.lan.trigger_overrun.LAN2	4
B3	status.operation.instrument.lan.trigger_overrun.LAN3	8
B4	status.operation.instrument.lan.trigger_overrun.LAN4	16
B5	status.operation.instrument.lan.trigger_overrun.LAN5	32
B6	status.operation.instrument.lan.trigger_overrun.LAN6	64
B7	status.operation.instrument.lan.trigger_overrun.LAN7	128
B8	status.operation.instrument.lan.trigger_overrun.LAN8	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status LAN trigger overrun enable register, set `status.operation.instrument.lan.trigger_overrun.enable = status.operation.instrument.lan.trigger_overrun.LAN1`.

In addition to the above constants, `operationRegister` can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set `operationRegister` to the sum of their decimal weights. For example, to set bits B1 and B8, set `operationRegister` to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.operation.instrument.lan.trigger_overrun.LAN1 +
    status.operation.instrument.lan.trigger_overrun.LAN8
status.operation.instrument.lan.trigger_overrun.enable = operationRegister
```

Use constants to set bit B1 and bit B8 of the operation status LAN trigger overrun enable register.

Example 2

```
operationRegister = 258
status.operation.instrument.lan.trigger_overrun.enable = operationRegister
```

Use the decimal value to set bit B1 and bit B8 of the operation status LAN trigger overrun enable register.

Also see

[Operation Status Registers](#) (on page 10-7)
[status.operation.instrument.lan.*](#) (on page 15-299)

status.operation.instrument.smuX.*

This attribute contains the operation status SMU X summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1049 (All bits set)

Usage

```
operationRegister = status.operation.instrument.smuX.condition
operationRegister = status.operation.instrument.smuX.enable
operationRegister = status.operation.instrument.smuX.event
operationRegister = status.operation.instrument.smuX.ntr
operationRegister = status.operation.instrument.smuX.ptr
status.operation.instrument.smuX.enable = operationRegister
status.operation.instrument.smuX.ntr = operationRegister
status.operation.instrument.smuX.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status SMU X summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
<i>X</i>	Source-measure unit (SMU) channel (for example <code>status.operation.instrument.smuA.enable</code> applies to SMU channel A)

Details

These attributes are used to read or write to the operation status SMU X summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02500e+02` (which is 1,025) is read as the value of the condition register, the binary equivalent is `0000 0100 0000 0010`. This value indicates that bit B0 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

For information about `.condition`, `.enable`, `.event`, `.ntr`, and `.ptr` registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	<code>status.operation.instrument.smuX.CALIBRATING</code> <code>status.operation.instrument.smuX.CAL</code> Set bit indicates that <code>smuX</code> is unlocked for calibration. Bit B0 decimal value: 1
B1 to B2	Not used
B3	<code>status.operation.instrument.smuX.SWEEPING</code> <code>status.operation.instrument.smuX.SWE</code> Set bit indicates that <code>smuX</code> is sweeping. Bit B3 decimal value: 8
B4	<code>status.operation.instrument.smuX.MEASURING</code> <code>status.operation.instrument.smuX.MEAS</code> Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement. Bit B4 decimal value: 16
B5 to B9	Not used

Bit	Value and description
B10	<code>status.operation.instrument.smuX.TRIGGER_OVERRUN</code> <code>status.operation.instrument.smuX.TRGOVR</code> Set bit indicates an enabled bit has been set in the operation status <code>smuX</code> trigger overrun event register. Bit B10 decimal value: 1,024
B11 to B15	Not used

As an example, to set bit B0 of the operation status SMU A summary enable register, set
`status.operation.instrument.smua.enable =`
`status.operation.instrument.smua.CAL.`

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B0 and B10, set *operationRegister* to 1,025 (which is the sum of 1 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
status.operation.instrument.smua.enable = status.operation.instrument.smua.MEAS
```

Use a constant to set bit B4 of the operation status SMU A summary enable register.

Example 2

```
status.operation.instrument.smua.enable = 1025
```

Use the decimal value to set bits B0 and B10 of the operation status SMU A summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)
[status.operation.instrument.smuX.trigger_overrrun.*](#) (on page 15-306)

status.operation.instrument.smuX.trigger_overnrun.*

This attribute contains the operation status SMU X trigger overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	30 (All bits set)

Usage

```
operationRegister = status.operation.instrument.smuX.trigger_overnrun.condition
operationRegister = status.operation.instrument.smuX.trigger_overnrun.enable
operationRegister = status.operation.instrument.smuX.trigger_overnrun.event
operationRegister = status.operation.instrument.smuX.trigger_overnrun.ntr
operationRegister = status.operation.instrument.smuX.trigger_overnrun.ptr
status.operation.instrument.smuX.trigger_overnrun.enable = operationRegister
status.operation.instrument.smuX.trigger_overnrun.ntr = operationRegister
status.operation.instrument.smuX.trigger_overnrun.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status SMU X trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status SMU X trigger overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 18 is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	<code>status.operation.instrument.smuX.trigger_overrun.ARM</code> Set bit indicates that the arm event detector of the SMU was already in the detected state when a trigger was received. Bit B1 decimal value: 2
B2	<code>status.operation.instrument.smuX.trigger_overrun.SRC</code> Set bit indicates that the source event detector of the SMU was already in the detected state when a trigger was received. Bit B2 decimal value: 4
B3	<code>status.operation.instrument.smuX.trigger_overrun.MEAS</code> Set bit indicates that the measurement event detector of the SMU was already in the detected state when a trigger was received. Bit B3 decimal value: 8
B4	<code>status.operation.instrument.smuX.trigger_overrun.ENDP</code> Set bit indicates that the end pulse event detector of the SMU was already in the detected state when a trigger was received. Bit B4 decimal value: 16
B5 to B15	Not used

As an example, to set bit B1 of the operation status SMU A trigger overrun enable register, set `status.operation.instrument.smua.trigger_overrun.enable = status.operation.instrument.smua.trigger_overrun.ARM`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B4, set *operationRegister* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
status.operation.instrument.smua.trigger_overrun.enable =
    status.operation.instrument.smua.trigger_overrun.ARM
```

Uses a constant to sets the ARM bit of the operation status SMU A trigger overrun enable register.

Example 2

```
status.operation.instrument.smua.trigger_overrun.enable = 18
```

Uses the decimal value to set bits B1 and B4 of the operation status SMU A trigger overrun enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.smuX.*](#) (on page 15-303)

status.operation.instrument.trigger_blender.*

This attribute contains the operation status trigger blender summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.trigger_blender.condition
operationRegister = status.operation.instrument.trigger_blender.enable
operationRegister = status.operation.instrument.trigger_blender.event
operationRegister = status.operation.instrument.trigger_blender.ntr
operationRegister = status.operation.instrument.trigger_blender.ptr
status.operation.instrument.trigger_blender.enable = operationRegister
status.operation.instrument.trigger_blender.ntr = operationRegister
status.operation.instrument.trigger_blender.ptr = operationRegister
```

operationRegister

The status of the operation status trigger blender summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status trigger blender summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used
B10	status.operation.instrument.trigger_blender.TRIGGER_OVERRUN status.operation.instrument.trigger_blender.TRGOVR Set bit indicates one or more enabled bits for operation status trigger blender overrun register is set. Bit B10 decimal value: 1,024 Binary value: 0100 0000 0000
B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *operationRegister* to 1024.

Example

```
status.operation.instrument.trigger_blender.enable = 1024
```

Uses a decimal value to set the TRGOVR bit of the operation status trigger blender summary enable.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.trigger_blender.trigger_overrun.*](#) (on page 15-309)

status.operation.instrument.trigger_blender.trigger_overrun.*

This attribute contains the operation status trigger blender overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	126 (All bits set)

Usage

```
operationRegister =
    status.operation.instrument.trigger_blender.trigger_overrun.condition
operationRegister =
    status.operation.instrument.trigger_blender.trigger_overrun.enable
operationRegister =
    status.operation.instrument.trigger_blender.trigger_overrun.event
operationRegister =
    status.operation.instrument.trigger_blender.trigger_overrun.ntr
operationRegister =
    status.operation.instrument.trigger_blender.trigger_overrun.ptr
status.operation.instrument.trigger_blender.trigger_overrun.enable =
    operationRegister
status.operation.instrument.trigger_blender.trigger_overrun.ntr =
    operationRegister
status.operation.instrument.trigger_blender.trigger_overrun.ptr =
    operationRegister
```

<i>operationRegister</i>	The status of the operation status trigger blender overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger blender overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 18 is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit

** Most significant bit

A set bit value indicates that the specified trigger blender generated an action overrun.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	status.operation.instrument.trigger_blender.trigger_overrun.BLND1	2
B2	status.operation.instrument.trigger_blender.trigger_overrun.BLND2	4
B3	status.operation.instrument.trigger_blender.trigger_overrun.BLND3	8
B4	status.operation.instrument.trigger_blender.trigger_overrun.BLND4	16
B5	status.operation.instrument.trigger_blender.trigger_overrun.BLND5	32
B6	status.operation.instrument.trigger_blender.trigger_overrun.BLND6	64
B7 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status trigger blender overrun enable register, set `status.operation.instrument.trigger_blender.trigger_overrun.enable = status.operation.instrument.trigger_blender.trigger_overrun.BLND1`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B4, set *operationRegister* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example

```
status.operation.instrument.trigger_blender.trigger_overnrun.enable
= status.operation.instrument.trigger_blender.trigger_overnrun.BLND1
```

Uses the constant to set the bit for blender 1 of the operation status trigger blender overrun enable register.

Example

```
status.operation.instrument.trigger_blender.trigger_overnrun.enable = 18
```

Uses the decimal value to set the bits for blenders 1 and 4 of the operation status trigger blender overrun enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.trigger_blender.*](#) (on page 15-308)

status.operation.instrument.trigger_timer.*

This attribute contains the operation status trigger timer summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.trigger_timer.condition
operationRegister = status.operation.instrument.trigger_timer.enable
operationRegister = status.operation.instrument.trigger_timer.event
operationRegister = status.operation.instrument.trigger_timer.ntr
operationRegister = status.operation.instrument.trigger_timer.ptr
status.operation.instrument.trigger_timer.enable = operationRegister
status.operation.instrument.trigger_timer.ntr = operationRegister
status.operation.instrument.trigger_timer.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status trigger timer summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status trigger timer summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used
B10	status.operation.instrument.trigger_timer.TRIGGER_OVERRUN status.operation.instrument.trigger_timer.TRGOVR Set bit indicates one or more enabled bits for the operation status trigger timer overrun register is set. Bit B10 decimal value: 1,024 Binary value: 0100 0000 0000
B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *operationRegister* to 1024.

Example

```
status.operation.instrument.trigger_timer.enable = 1024
```

Uses the decimal value to set the TRGOVR bit of the operation status trigger timer summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.trigger_timer.trigger_overrun.*](#) (on page 15-312)

status.operation.instrument.trigger_timer.trigger_overrun.*

This attribute contains the operation status trigger timer overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```

operationRegister =
    status.operation.instrument.trigger_timer.trigger_overrun.condition
operationRegister =
    status.operation.instrument.trigger_timer.trigger_overrun.enable
operationRegister =
    status.operation.instrument.trigger_timer.trigger_overrun.event
operationRegister =
    status.operation.instrument.trigger_timer.trigger_overrun.ntr
operationRegister =
    status.operation.instrument.trigger_timer.trigger_overrun.ptr
status.operation.instrument.trigger_timer.trigger_overrun.enable =
    operationRegister
status.operation.instrument.trigger_timer.trigger_overrun.ntr =
    operationRegister
status.operation.instrument.trigger_timer.trigger_overrun.ptr =
    operationRegister

```

operationRegister

The status of the operation status trigger timer trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger timer overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 18 is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit

** Most significant bit

A set bit indicates the specified timer generated an action overrun because it was still processing a delay from a previous trigger when a new trigger was received.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	status.operation.instrument.trigger_timer.trigger_overrun.TMR1	2
B2	status.operation.instrument.trigger_timer.trigger_overrun.TMR2	4
B3	status.operation.instrument.trigger_timer.trigger_overrun.TMR3	8
B4	status.operation.instrument.trigger_timer.trigger_overrun.TMR4	16
B5	status.operation.instrument.trigger_timer.trigger_overrun.TMR5	32
B6	status.operation.instrument.trigger_timer.trigger_overrun.TMR6	64
B7	status.operation.instrument.trigger_timer.trigger_overrun.TMR7	128
B8	status.operation.instrument.trigger_timer.trigger_overrun.TMR8	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status trigger timer trigger overrun enable register, set `status.operation.instrument.trigger_timer.trigger_overrun.enable = status.operation.instrument.trigger_timer.trigger_overrun.TMR1`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B4, set *operationRegister* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
status.operation.instrument.trigger_timer.trigger_overrun.enable =
    status.operation.instrument.trigger_timer.trigger_overrun.TMR3
```

Uses a constant to set the timer 3 bit of the operation status trigger timer overrun enable register.

Example 2

```
status.operation.instrument.trigger_timer.trigger_overrun.enable = 18
```

Uses a constant to set timer bits B1 and B4 of the operation status trigger timer overrun enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.trigger_timer.*](#) (on page 15-311)

status.operation.instrument.tsplink.*

This attribute contains the operation status TSP-Link summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.tsplink.condition
operationRegister = status.operation.instrument.tsplink.enable
operationRegister = status.operation.instrument.tsplink.event
operationRegister = status.operation.instrument.tsplink.ntr
operationRegister = status.operation.instrument.tsplink.ptr
status.operation.instrument.tsplink.enable = operationRegister
status.operation.instrument.tsplink.ntr = operationRegister
status.operation.instrument.tsplink.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status TSP-Link summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status TSP-Link summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used
B10	status.operation.instrument.tsplink.TRIGGER_OVERRUN status.operation.instrument.tsplink.TRGOVR Set bit indicates one or more enabled bits for the operation status TSP-Link overrun register is set. Bit B10 decimal value: 1,024 Binary value: 0100 0000 0000
B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *operationRegister* to 1024.

Example

```
status.operation.instrument.tsplink.enable = 1024
```

Uses the decimal value to set the trigger overrun bit of the operation status TSP-Link summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.tsplink.trigger_overrun.*](#) (on page 15-316)

status.operation.instrument.tsplink.trigger_overrun.*

This attribute contains the operation status TSP-Link overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	14 (All bits set)

Usage

```
operationRegister =
    status.operation.instrument.tsplink.trigger_overrun.condition
operationRegister = status.operation.instrument.tsplink.trigger_overrun.enable
operationRegister = status.operation.instrument.tsplink.trigger_overrun.event
operationRegister = status.operation.instrument.tsplink.trigger_overrun.ntr
operationRegister = status.operation.instrument.tsplink.trigger_overrun.ptr
status.operation.instrument.tsplink.trigger_overrun.enable = operationRegister
status.operation.instrument.tsplink.trigger_overrun.ntr = operationRegister
status.operation.instrument.tsplink.trigger_overrun.ptr = operationRegister
```

operationRegister

The status of the operation status TSP-link overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status TSP-link overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 10 is read as the value of the condition register, the binary equivalent is 0000 0000 0000 1010. This value indicates that bit B1 and bit B3 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

* Least significant bit

** Most significant bit

A set bit indicates that the specified line generated an action overrun when triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	status.operation.instrument.tsplink.trigger_overrun.LINE1	2
B2	status.operation.instrument.tsplink.trigger_overrun.LINE2	4
B3	status.operation.instrument.tsplink.trigger_overrun.LINE3	8
B4 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status TSP-Link overrun enable register, set
`status.operation.instrument.tsplink.trigger_overrun.enable =`
`status.operation.instrument.tsplink.trigger_overrun.LINE1.`

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B3, set *operationRegister* to 10 (which is the sum of 2 + 8).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
status.operation.instrument.tsplink.trigger_overrun.enable =
    status.operation.instrument.tsplink.trigger_overrun.LINE1
```

Uses a constant to set the line 1 bit of the operation status TSP-Link overrun enable register.

Example 2

```
status.operation.instrument.tsplink.trigger_overrun.enable = 10
```

Uses the decimal value to set bits for lines 1 and 3 of the operation status TSP-Link overrun enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.instrument.trigger_timer.*](#) (on page 15-311)

status.operation.measuring.*

This attribute contains the operation status measuring summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
operationRegister = status.operation.measuring.condition
operationRegister = status.operation.measuring.enable
operationRegister = status.operation.measuring.event
operationRegister = status.operation.measuring.ntr
operationRegister = status.operation.measuring.ptr
status.operation.measuring.enable = operationRegister
status.operation.measuring.ntr = operationRegister
status.operation.measuring.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status measuring summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status measuring summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.operation.measuring.SMUA	Bit is set when SMU A is making an overlapped measurement, but it is not set when making a normal synchronous measurement. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.operation.measuring.SMUB	Bit is set when SMU B is making an overlapped measurement, but it is not set when making a normal synchronous measurement. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status measuring summary enable register, set `status.operation.measuring.enable = status.operation.measuring.SMUA`.

In addition to the above constants, `operationRegister` can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set `operationRegister` to the sum of their decimal weights. For example, to set bits B1 and B2, set `operationRegister` to 6 (which is the sum of 2 + 4).

Example

```
status.operation.measuring.enable =
    status.operation.measuring.SMUA
```

Uses a constant to set the SMUA bit of the operation status measuring summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.*](#) (on page 15-290)

status.operation.remote.*

This attribute contains the operation status remote summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2050 (All bits set)

Usage

```
operationRegister = status.operation.remote.condition
operationRegister = status.operation.remote.enable
operationRegister = status.operation.remote.event
operationRegister = status.operation.remote.ntr
operationRegister = status.operation.remote.ptr
status.operation.remote.enable = operationRegister
status.operation.remote.ntr = operationRegister
status.operation.remote.ptr = operationRegister
```

`operationRegister`

The status of the operation status remote summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status remote summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	<code>status.operation.remote.COMMAND_AVAILABLE</code> <code>status.operation.remote.CAV</code> Set bit indicates there is a command available in the execution queue. Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2 to B10	Not used
B11	<code>status.operation.remote.PROMPTS_ENABLED</code> <code>status.operation.remote.PRMPPT</code> Set bit indicates command prompts are enabled. Bit B11 decimal value: 2,048 Binary value: 0000 0100 0000 0000
B12 to B15	Not used

As an example, to set bit B1 of the operation status remote summary enable register, set `status.operation.remote.enable = status.operation.remote.CAV`.

In addition to the above constants, *operationRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal values. For example, to set bits B1 and B11, set *operationRegister* to 2,050 (which is the sum of 2 + 2,048).

Example 1

```
status.operation.remote.enable = status.operation.remote.CAV
```

Uses a constant to set the CAV bit, B1, of the operation status remote summary enable register.

Example 2

```
status.operation.remote.enable = 2050
```

Uses the decimal value to set bits B1 and B11 of the operation status remote summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.*](#) (on page 15-290)

status.operation.sweeping.*

This attribute contains the operation status sweeping summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
operationRegister = status.operation.sweeping.condition
operationRegister = status.operation.sweeping.enable
operationRegister = status.operation.sweeping.event
operationRegister = status.operation.sweeping.ntr
operationRegister = status.operation.sweeping.ptr
status.operation.sweeping.enable = operationRegister
status.operation.sweeping.ntr = operationRegister
status.operation.sweeping.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status sweeping summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status sweeping summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.operation.sweeping.SMUA	Set bit indicates that SMU A is sweeping. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.operation.sweeping.SMUB	Set bit indicates SMU B is sweeping. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status sweeping summary enable register, set `status.operation.sweeping.enable = status.operation.sweeping.SMUA`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *operationRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.operation.sweeping.enable = status.operation.sweeping.SMUA
```

Uses a constant to set the SMUA bit of the operation status sweeping summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)
[status.operation.*](#) (on page 15-290)

status.operation.trigger_overrun.*

This attribute contains the operation status trigger overrun summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	31,750 (All bits set)

Usage

```
operationRegister = status.operation.trigger_overrun.condition
operationRegister = status.operation.trigger_overrun.enable
operationRegister = status.operation.trigger_overrun.event
operationRegister = status.operation.trigger_overrun.ntr
operationRegister = status.operation.trigger_overrun.ptr
status.operation.trigger_overrun.enable = operationRegister
status.operation.trigger_overrun.ntr = operationRegister
status.operation.trigger_overrun.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status trigger overrun summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger overrun summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 1.02600e+03 (which is 1,026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

The bits in this register summarize events in other registers. A set bit in this summary register indicates that an enabled event in one of the summarized registers is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	<code>status.operation.trigger_overnrun.SMUA</code>	Set bit indicates one of the enabled bits in the operation status SMU A trigger overrun event register is set. Bit B1 decimal value: 2
B2	<code>status.operation.trigger_overnrun.SMUB</code>	Set bit indicates one of the enabled bits in the operation status SMU B trigger overrun event register is set. Bit B2 decimal value: 4
B3 to B9	Not used	Not applicable.
B10	<code>status.operation.trigger_overnrun.TRIGGER_BLENDER</code> <code>status.operation.trigger_overnrun.TRGBLND</code>	Set bit indicates one of the enabled bits in the operation status trigger blender overrun event register is set. Bit B10 decimal value: 1,024
B11	<code>status.operation.trigger_overnrun.TRIGGER_TIMER</code> <code>status.operation.trigger_overnrun.TRGTMR</code>	Set bit indicates one of the enabled bits in the operation status trigger timer overrun event register is set. Bit B11 decimal value: 2,048
B12	<code>status.operation.trigger_overnrun.DIGITAL_IO</code> <code>status.operation.trigger_overnrun.DIGIO</code>	Set bit indicates one of the enabled bits in the operation status digital I/O overrun event register is set. Bit B12 decimal value: 4,096
B13	<code>status.operation.trigger_overnrun.TSPLINK</code>	Set bit indicates one of the enabled bits in the operation status TSP-Link overrun event register is set. Bit B13 decimal value: 8,192
B14	<code>status.operation.trigger_overnrun.LAN</code>	Set bit indicates one of the enabled bits in the operation status LAN trigger overrun event register is set. Bit B14 decimal value: 16,384
B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status trigger overrun summary enable register, set `status.operation.trigger_overnrun.enable = status.operation.trigger_overnrun.SMUA`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B10, set *operationRegister* to 1,026 (which is the sum of 2 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example

```
operationRegister = status.operation.trigger_overrun.SMUA +
    status.operation.trigger_overrun.TRGBLND
status.operation.trigger_overrun.enable = operationRegister
```

Uses constants to set bit B1 and bit B10 of the operation status trigger overrun summary enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.*](#) (on page 15-290)

status.operation.user.*

These attributes manage the operation status user register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (RW)	Yes	Status reset	Not saved	0
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
operationRegister = status.operation.user.condition
operationRegister = status.operation.user.enable
operationRegister = status.operation.user.event
operationRegister = status.operation.user.ntr
operationRegister = status.operation.user.ptr
status.operation.user.condition = operationRegister
status.operation.user.enable = operationRegister
status.operation.user.ntr = operationRegister
status.operation.user.ptr = operationRegister
```

<i>operationRegister</i>	The status of the operation status user register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status user registers. Reading a status register returns a value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of $1.29000e+02$ (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bits B0 and B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	status.operation.user.BIT0	1
B1	status.operation.user.BIT1	2
B2	status.operation.user.BIT2	4
B3	status.operation.user.BIT3	8
B4	status.operation.user.BIT4	16
B5	status.operation.user.BIT5	32
B6	status.operation.user.BIT6	64
B7	status.operation.user.BIT7	128
B8	status.operation.user.BIT8	256
B9	status.operation.user.BIT9	512
B10	status.operation.user.BIT10	1,024
B11	status.operation.user.BIT11	2,048
B12	status.operation.user.BIT12	4,096
B13	status.operation.user.BIT13	8,192
B14	status.operation.user.BIT14	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the operation status user enable register, set
`status.operation.user.enable = status.operation.user.BIT0.`

In addition to the above constants, *operationRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal values. For example, to set bits B11 and B14, set *operationRegister* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.operation.user.BIT11 + status.operation.user.BIT14
status.operation.user.enable = operationRegister
```

Uses constants to set bits B11 and B14 of the operation status user enable register.

Example 2

```
-- 18432 = binary 0100 1000 0000 0000
operationRegister = 18432
status.operation.enable = operationRegister
```

Uses the decimal value to set bits B11 and B14 of the operation status user enable register.

Also see

[Operation Status Registers](#) (on page 10-7)

[status.operation.*](#) (on page 15-290)

status.questionable.*

These attributes manage the questionable status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	13,056 (All bits set)

Usage

```
questionableRegister = status.questionable.condition
questionableRegister = status.questionable.enable
questionableRegister = status.questionable.event
questionableRegister = status.questionable.ntr
questionableRegister = status.questionable.ptr
status.questionable.enable = questionableRegister
status.questionable.ntr = questionableRegister
status.questionable.ptr = questionableRegister
```

questionableRegister The status of the questionable status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status registers. Reading a status register returns a value. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.22880e+04 (which is 12,288) is read as the value of 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
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0 to B7	Not used	Not available
B8	status.questionable.CALIBRATION status.questionable.CAL	An enabled bit in the questionable status calibration summary event register is set. Bit B6 decimal value: 256
B9	status.questionable.UNSTABLE_OUTPUT status.questionable.UO	An enabled bit in the questionable status unstable output summary event register is set. Bit B9 decimal value: 512
B10 to B11	Not used	Not available
B12	status.questionable.OVER_TEMPERATUREst atus.questionable.OTEMP	An enabled bit in the questionable status over temperature summary event register is set. Bit B12 decimal value: 4,096
B13	status.questionable.INSTRUMENT_SUMMARY status.questionable.INST	An enabled bit in the questionable status instrument summary event register is set. Bit B13 decimal value: 8,192
B14 to B15	Not used	Not available

As an example, to set bit B9 of the questionable status enable register, set
`status.questionable.enable = status.questionable.UO.`

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *questionableRegister* to the sum of their decimal weights. For example, to set bits B12 and B13, set *questionableRegister* to 12,288 (which is the sum of 4,096 + 8,192).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example

```
status.questionable.enable = status.questionable.OTEMP
```

Uses a constant to set the OTEMP bit of the questionable status enable register.

Also see

[Questionable Status Registers](#) (on page 10-12)

status.questionable.calibration.*

This attribute contains the questionable status calibration summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
questionableRegister = status.questionable.calibration.condition
questionableRegister = status.questionable.calibration.enable
questionableRegister = status.questionable.calibration.event
questionableRegister = status.questionable.calibration.ntr
questionableRegister = status.questionable.calibration.ptr
status.questionable.calibration.enable = questionableRegister
status.questionable.calibration.ntr = questionableRegister
status.questionable.calibration.ptr = questionableRegister
```

<i>questionableRegister</i>	The status of the questionable status calibration summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status calibration summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.questionable.calibration.SMUA	Set bit indicates that the SMU A calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.questionable.calibration.SMUB	Set bit indicates that the SMU B calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the questionable status calibration summary enable register, set `status.questionable.calibration.enable = status.questionable.calibration.SMUA`.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.questionable.calibration.enable = status.questionable.calibration.SMUA
```

Uses a constant to set the SMUA bit of the questionable status calibration summary enable register.

Also see

[Questionable Status Registers](#) (on page 10-12)
[status.questionable.*](#) (on page 15-326)

status.questionable.instrument.*

This attribute contains the questionable status instrument summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
questionableRegister = status.questionable.instrument.condition
questionableRegister = status.questionable.instrument.enable
questionableRegister = status.questionable.instrument.event
questionableRegister = status.questionable.instrument.ntr
questionableRegister = status.questionable.instrument.ptr
status.questionable.instrument.enable = questionableRegister
status.questionable.instrument.ntr = questionableRegister
status.questionable.instrument.ptr = questionableRegister
```

<i>questionableRegister</i>	The status of the questionable status instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.questionable.instrument.SMUA	Set bit indicates one or more enabled bits for the SMU A questionable register are set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.questionable.instrument.SMUB	Set bit indicates one or more enabled bits for the SMU B questionable register are set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the questionable status instrument summary enable register, set `status.questionable.instrument.enable = status.questionable.instrument.SMUA`.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.questionable.instrument.enable =
status.questionable.instrument.SMUA
```

Uses a constant to set the SMUA bit of the questionable status instrument summary enable register.

Also see

[Questionable Status Registers](#) (on page 10-12)

[status.questionable.*](#) (on page 15-326)

status.questionable.instrument.smuX.*

This attribute contains the questionable status SMU X summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	4864 (All bits set)

Usage

```
questionableRegister = status.questionable.instrument.smuX.condition
questionableRegister = status.questionable.instrument.smuX.enable
questionableRegister = status.questionable.instrument.smuX.event
questionableRegister = status.questionable.instrument.smuX.ntr
questionableRegister = status.questionable.instrument.smuX.ptr
status.questionable.instrument.smuX.enable = questionableRegister
status.questionable.instrument.smuX.ntr = questionableRegister
status.questionable.instrument.smuX.ptr = questionableRegister
```

<i>questionableRegister</i>	The status of the questionable status SMU X summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
<i>X</i>	Source-measure unit (SMU) channel (for example <code>status.questionable.instrument.smuA.enable</code> applies to SMU channel A)

Details

These attributes are used to read or write to the questionable status instrument SMU X summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `7.68000e+02` (which is 768) is read as the value of the condition register, the binary equivalent is 0000 0011 0000 0000. This value indicates that bit B8 and bit B9 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0 to B7	Not used	Not applicable.
B8	status.questionable.instrument.smuX.CALIBRATION status.questionable.instrument.smuX.CAL	Set bit indicates that the calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B8 decimal value: 256
B9	status.questionable.instrument.smuX.UNSTABLE_OUTPUT status.questionable.instrument.smuX.UO	Set bit indicates that an unstable output condition was detected. Bit B9 decimal value: 512
B10 to B11	Not used	Not applicable
B12	status.questionable.instrument.smuX.OVER_TEMPERATURE status.questionable.instrument.smuX.OTEMP	Set bit indicates that an over temperature condition was detected. Bit B12 decimal value: 4,096
B13 to B15	Not used	Not applicable.

As an example, to set bit B8 of the questionable status SMU A summary enable register, set
`status.questionable.instrument.smua.enable =`
`status.questionable.instrument.smua.CAL.`

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *questionableRegister* to the sum of their decimal weights. For example, to set bits B8 and B9, set *questionableRegister* to 768 (which is the sum of 256 + 512).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example

```
questionableRegister = status.questionable.instrument.smua.CAL +
    status.questionable.instrument.smua.UO
status.questionable.instrument.smua.enable = questionableRegister
```

Uses constants to set bit B8 and bit B9 of the questionable status SMU A summary enable register.

Also see

[Questionable Status Registers](#) (on page 10-12)

[status.operation.*](#) (on page 15-290)

status.questionable.over_temperature.*

This attribute contains the questionable status over temperature summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
questionableRegister = status.questionable.over_temperature.condition
questionableRegister = status.questionable.over_temperature.enable
questionableRegister = status.questionable.over_temperature.event
questionableRegister = status.questionable.over_temperature.ntr
questionableRegister = status.questionable.over_temperature.ptr
status.questionable.over_temperature.enable = questionableRegister
status.questionable.over_temperature.ntr = questionableRegister
status.questionable.over_temperature.ptr = questionableRegister
```

<i>operationRegister</i>	The status of the questionable status over temperature summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status over temperature summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	<code>status.questionable.over_temperature.SMUA</code>	Set bit indicates that an over temperature condition was detected on SMU A. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	<code>status.questionable.over_temperature.SMUB</code>	Set bit indicates that an over temperature condition was detected on SMU B. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the questionable status over temperature summary enable register, set `status.questionable.instrument.enable = status.questionable.instrument.SMUA`.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
status.questionable.over_temperature.enable =
status.questionable.over_temperature.SMUA
```

Uses a constant to set the SMU A bit in the questionable status over temperature summary enable register.

Also see

[Questionable Status Registers](#) (on page 10-12)
[status.questionable.*](#) (on page 15-326)

status.questionable.unstable_output.*

This attribute contains the questionable status unstable output summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (All bits set)

Usage

```
questionableRegister = status.questionable.unstable_output.condition
questionableRegister = status.questionable.unstable_output.enable
questionableRegister = status.questionable.unstable_output.event
questionableRegister = status.questionable.unstable_output.ntr
questionableRegister = status.questionable.unstable_output.ptr
status.questionable.unstable_output.enable = questionableRegister
status.questionable.unstable_output.ntr = questionableRegister
status.questionable.unstable_output.ptr = questionableRegister
```

<i>operationRegister</i>	The status of the questionable status unstable output summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status unstable output summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.questionable.unstable_output.SMUA	Set bit indicates that an unstable output condition was detected on SMU A. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	status.questionable.unstable_output.SMUB	Set bit indicates that an unstable output condition was detected on SMU B. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the questionable status unstable output summary enable register, send:

```
status.questionable.instrument.enable = status.questionable.instrument.SMUA
```

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

status.questionable.unstable_output.enable = status.questionable.unstable_output.SMUA	Uses a constant to set the SMU A bit in the questionable status unstable output summary enable register bit.

Also see

[Questionable Status Registers](#) (on page 10-12)
[status.questionable.*](#) (on page 15-326)

status.request_enable

This attribute stores the service request (SRQ) enable register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
requestSRQEnableRegister = status.request_enable
status.request_enable = requestSRQEnableRegister
```

<i>requestSRQEnableRegister</i>	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

Details

This attribute is used to read or write to the service request enable register. Reading the service request enable register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of $1.29000e+02$ (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB	Set summary bit indicates that an enabled event in the Measurement Event Register has occurred. Bit B0 decimal value: 1
B1	status.SYSTEM_SUMMARY_BIT status.SSB	Set summary bit indicates that an enabled event in the System Summary Register has occurred. Bit B1 decimal value: 2
B2	status.ERROR_AVAILABLE status.EAV	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB	Set summary bit indicates that an enabled event in the Questionable Status Register has occurred. Bit B3 decimal value: 8

Bit	Value	Description
B4	<code>status.MESSAGE_AVAILABLE</code> <code>status.MAV</code>	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	<code>status.EVENT_SUMMARY_BIT</code> <code>status.ESB</code>	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred. Bit B5 decimal value: 32
B6	Not used	Not applicable
B7	<code>status.OPERATION_SUMMARY_BIT</code> <code>status.OSB</code>	Set summary bit indicates that an enabled event in the Operation Status Register has occurred. Bit B7 decimal value: 128

As an example, to set bit B0 of the service request enable register, set `status.request_enable = status.MSB`.

In addition to the above values, `requestSRQEnableRegister` can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set `requestSRQEnableRegister` to the sum of their decimal weights. For example, to set bits B0 and B7, set `requestSRQEnableRegister` to 129 (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

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.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the MSB and OSB bits of the service request (SRQ) enable register.

Also see

[Status byte and service request \(SRQ\)](#) (on page 10-14)

[status.condition](#) (on page 15-273)

[status.system.*](#) (on page 15-342)

status.request_event

This attribute stores the service request (SRQ) event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

```
requestSRQEventRegister = status.request_event
```

<i>requestSRQEventRegister</i>	The status of the request event register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the service request event register, which is returned as a numeric value. Reading this register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of $1.29000e+02$ (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

The returned value can indicate one or more status events occurred.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB	Set summary bit indicates that an enabled event in the Measurement Event Register has occurred. Bit B0 decimal value: 1
B1	status.SYSTEM_SUMMARY_BIT status.SSB	Set summary bit indicates that an enabled event in the System Summary Register has occurred. Bit B1 decimal value: 2
B2	status.ERROR_AVAILABLE status.EAV	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB	Set summary bit indicates that an enabled event in the Questionable Status Register has occurred. Bit B3 decimal value: 8
B4	status.MESSAGE_AVAILABLE status.MAV	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	status.EVENT_SUMMARY_BIT status.ESB	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred. Bit B5 decimal value: 32

Bit	Value	Description
B6	Not used	Not applicable
B7	status.OPERATION_SUMMARY_BIT status.OSB	Set summary bit indicates that an enabled event in the Operation Status Register has occurred. Bit B7 decimal value: 128

In addition to the above constants, *requestEventRegister* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *requestEventRegister* contains the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Example

```
requestEventRegister = status.request_event
print(requestEventRegister)
```

Reads the status request event register.

Sample 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 request event register are presently B0 (MSB) and B7 (OSB).

Also see

[status.condition](#) (on page 15-273)

[status.system.*](#) (on page 15-342)

[Status byte and service request \(SRO\)](#) (on page 10-14)

status.reset()

This function resets all bits in the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.reset()
```

Details

This function clears all status data structure registers (enable, event, NTR, and PTR) to their default values. For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18).

Example

```
status.reset()
```

Resets the instrument status model.

Also see

[Status model](#) (on page 10-1)

status.standard.*

These attributes manage the standard event status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	253 (All bits set)

Usage

```

standardRegister = status.standard.condition
standardRegister = status.standard.enable
standardRegister = status.standard.event
standardRegister = status.standard.ntr
standardRegister = status.standard.ptr
status.standard.enable = standardRegister
status.standard.ntr = standardRegister
status.standard.ptr = standardRegister

```

<i>standardRegister</i>	The status of the standard event status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the standard event status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 1.29000e+02 (which is 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.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value
B0	<code>status.standard.OPERATION_COMPLETE</code> <code>status.standard.OPC</code> Set bit indicates that all pending selected instrument operations are completed and the instrument is ready to accept new commands. The bit is set in response to an <code>*OPC</code> command. The <code>opc()</code> function can be used in place of the <code>*OPC</code> command. Bit B0 decimal value: 1
B1	Not used
B2	<code>status.standard.QUERY_ERROR</code> <code>status.standard.QYE</code> Set bit indicates that you attempted to read data from an empty Output Queue. Bit B2 decimal value: 4
B3	<code>status.standard.DEVICE_DEPENDENT_ERROR</code> <code>status.standard.DDE</code> Set bit indicates that an instrument operation did not execute properly due to some internal condition. Bit B3 decimal value: 8
B4	<code>status.standard.EXECUTION_ERROR</code> <code>status.standard.EXE</code> Set bit indicates that the instrument detected an error while trying to execute a command. Bit B4 decimal value: 16
B5	<code>status.standard.COMMAND_ERROR</code> <code>status.standard.CME</code> Set bit indicates that a command error has occurred. Command errors include: IEEE Std 488.2 syntax error: Instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard. Semantic error: Instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented. GET error: The instrument received a Group Execute Trigger (GET) inside a program message. Bit B5 decimal value: 32
B6	<code>status.standard.USER_REQUEST</code> <code>status.standard.URQ</code> Set bit indicates that the LOCAL key on the instrument front panel was pressed. Bit B6 decimal value: 64
B7	<code>status.standard.POWER_ON</code> <code>status.standard.PON</code> Set bit indicates that the instrument has been turned off and turned back on since the last time this register has been read. Bit B7 decimal value: 128
B8 to B15	Not used

As an example, to set bit B0 of the standard event status enable register, set
`status.standard.enable = status.standard.OPC`.

In addition to the above constants, *standardRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B4, set *standardRegister* to 17 (which is the sum of 1 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Example 1

```
standardRegister = status.standard.OPC + status.standard.EXE
status.standard.enable = standardRegister
```

Uses constants to set the OPC and EXE bits of the standard event status enable register.

Example 2

```
-- decimal 17 = binary 0001 0001
standardRegister = 17
status.standard.enable = standardRegister
```

Uses the decimal value to set the OPC and EXE bits of the standard event status enable register.

Also see

[Standard Event Register](#) (on page 10-19)

status.system.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 1 through 14.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system.condition
enableRegister = status.system.enable
enableRegister = status.system.event
enableRegister = status.system.ntr
enableRegister = status.system.ptr
status.system.enable = enableRegister
status.system.ntr = enableRegister
status.system.ptr = enableRegister
```

<i>enableRegister</i>	The status of the system summary register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.29000e+02 (which is 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.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	status.system.EXTENSION_BIT status.system.EXT	1
B1	status.system.NODE1	2
B2	status.system.NODE2	4
B3	status.system.NODE3	8
B4	status.system.NODE4	16
B5	status.system.NODE5	32
B6	status.system.NODE6	64
B7	status.system.NODE7	128
B8	status.system.NODE8	256
B9	status.system.NODE9	512
B10	status.system.NODE10	1,024
B11	status.system.NODE11	2,048
B12	status.system.NODE12	4,096
B13	status.system.NODE13	8,192
B14	status.system.NODE14	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary status enable register, set
`status.system.enable = status.system.enable.EXT.`

In addition to the above constants, *enableRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *enableRegister* to the sum of their decimal values. For example, to set bits B11 and B14, set *enableRegister* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
enableRegister = status.system.NODE11 + status.system.NODE14
status.system.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary enable register.

Also see

[status.system2.*](#) (on page 15-344)

[System summary and standard event registers](#) (on page 10-5)

status.system2.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 15 through 28.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system2.condition
enableRegister = status.system2.enable
enableRegister = status.system2.event
enableRegister = status.system2.ntr
enableRegister = status.system2.ptr
status.system2.enable = enableRegister
status.system2.ntr = enableRegister
status.system2.ptr = enableRegister
```

<i>enableRegister</i>	The status of the system summary 2 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.29000e+02 (which is 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.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	status.system2.EXTENSION_BIT status.system2.EXT	1
B1	status.system2.NODE15	2
B2	status.system2.NODE16	4
B3	status.system2.NODE17	8
B4	status.system2.NODE18	16
B5	status.system2.NODE19	32
B6	status.system2.NODE20	64
B7	status.system2.NODE21	128
B8	status.system2.NODE22	256
B9	status.system2.NODE23	512
B10	status.system2.NODE24	1,024
B11	status.system2.NODE25	2,048
B12	status.system2.NODE26	4,096
B13	status.system2.NODE27	8,192

Bit	Value	Decimal value
B14	<code>status.system2.NODE28</code>	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 2 enable register, set `status.system2.enable = status.system2.EXT`.

In addition to the above constants, *enableRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *enableRegister* to the sum of their decimal values. For example, to set bits B11 and B14, set *enableRegister* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
enableRegister = status.system2.NODE25 + status.system2.NODE28
status.system2.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 2 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system2.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary 2 enable register.

Also see

[status.system.*](#) (on page 15-342)

[status.system3.*](#) (on page 15-347)

[System summary and standard event registers](#) (on page 10-5)

status.system3.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 29 through 42.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system3.condition
enableRegister = status.system3.enable
enableRegister = status.system3.event
enableRegister = status.system3.ntr
enableRegister = status.system3.ptr
status.system3.enable = enableRegister
status.system3.ntr = enableRegister
status.system3.ptr = enableRegister
```

<i>enableRegister</i>	The status of the system summary 3 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0 and the most significant bit is bit B15. For example, if a value of 1.29000e+02 (which is 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.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	status.system3.EXTENSION_BIT status.system3.EXT	1
B1	status.system3.NODE29	2
B2	status.system3.NODE30	4
B3	status.system3.NODE31	8
B4	status.system3.NODE32	16

Bit	Value	Decimal value
B5	status.system3.NODE33	32
B6	status.system3.NODE34	64
B7	status.system3.NODE35	128
B8	status.system3.NODE36	256
B9	status.system3.NODE37	512
B10	status.system3.NODE38	1,024
B11	status.system3.NODE39	2,048
B12	status.system3.NODE40	4,096
B13	status.system3.NODE41	8,192
B14	status.system3.NODE42	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 3 enable register, set
`status.system3.enable = status.system3.EXT.`

In addition to the above constants, *enableRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *enableRegister* to the sum of their decimal values. For example, to set bits B11 and B14, set *enableRegister* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
enableRegister = status.system3.NODE39 + status.system3.NODE42
status.system3.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 3 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system3.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary 3 enable register.

Also see

[status.system2.*](#) (on page 15-344)

[status.system4.*](#) (on page 15-349)

[System summary and standard event registers](#) (on page 10-5)

status.system4.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 43 through 56.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system4.condition
enableRegister = status.system4.enable
enableRegister = status.system4.event
enableRegister = status.system4.ntr
enableRegister = status.system4.ptr
status.system4.enable = enableRegister
status.system4.ntr = enableRegister
status.system4.ptr = enableRegister
```

<i>enableRegister</i>	The status of the system summary 4 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.29000e+02 (which is 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.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	status.system4.EXTENSION_BIT status.system4.EXT	1
B1	status.system4.NODE43	2
B2	status.system4.NODE44	4
B3	status.system4.NODE45	8
B4	status.system4.NODE46	16
B5	status.system4.NODE47	32
B6	status.system4.NODE48	64
B7	status.system4.NODE49	128
B8	status.system4.NODE50	256
B9	status.system4.NODE51	512
B10	status.system4.NODE52	1,024
B11	status.system4.NODE53	2,048
B12	status.system4.NODE54	4,096
B13	status.system4.NODE55	8,192
B14	status.system4.NODE56	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 4 enable register, set
`status.system4.enable = status.system4.enable.EXT.`

In addition to the above constants, *enableRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *enableRegister* to the sum of their decimal values. For example, to set bits B11 and B14, set *enableRegister* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
enableRegister = status.system4.NODE53 + status.system4.NODE56
status.system2.enable = enableRegister
```

Uses constants to set bit B11 and bit B14 of the system summary 4 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system4.enable = enableRegister
```

Uses a decimal value to set bit B11 and bit B14 of the system summary 4 enable register.

Also see

[status.system3.*](#) (on page 15-347)

[status.system5.*](#) (on page 15-351)

[System summary and standard event registers](#) (on page 10-5)

status.system5.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 57 through 64.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	--	--	--	--
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```
enableRegister = status.system5.condition
enableRegister = status.system5.enable
enableRegister = status.system5.event
enableRegister = status.system5.ntr
enableRegister = status.system5.ptr
status.system5.enable = enableRegister
status.system5.ntr = enableRegister
status.system5.ptr = enableRegister
```

<i>enableRegister</i>	The status of the system summary 5 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.30000e+02 (which is 130) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0010. This value indicates that bit B1 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to [Status register set contents](#) (on page 10-1) and [Enable and transition registers](#) (on page 10-18). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	status.system5.NODE57	2
B2	status.system5.NODE58	4
B3	status.system5.NODE59	8
B4	status.system5.NODE60	16
B5	status.system5.NODE61	32
B6	status.system5.NODE62	64
B7	status.system5.NODE63	128
B8	status.system5.NODE64	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the system summary 5 enable register, set
`status.system5.enable = status.system5.NODE57`.

In addition to the above constants, *enableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *enableRegister* to the sum of their decimal weights. For example, to set bits B1 and B4, set *enableRegister* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
enableRegister = status.system5.NODE57 + status.system5.NODE60
status.system2.enable = enableRegister
```

Uses constants to set bits B1 and B4 of the system summary 5 enable register.

Example 2

```
-- decimal 18 = binary 0000 0000 0001 0010
enableRegister = 18
status.system5.enable = enableRegister
```

Uses the decimal value to set bits B1 and B4 of the system summary 5 enable register.

Also see

[status.system4.*](#) (on page 15-349)

[System summary and standard event registers](#) (on page 10-5)

SweepILinMeasureV()

This KISweep factory script function performs a linear current sweep with voltage measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepILinMeasureV(smuX, starti, stopi, stime, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <i>smua</i> refers to SMU channel A)
<i>starti</i>	Sweep start current in amperes
<i>stopi</i>	Sweep stop current in amperes
<i>stime</i>	Settling time in seconds; occurs after stepping the source and before making a measurement
<i>points</i>	Number of sweep points (must be ≥ 2)

Details

Data for voltage measurements, current source values, and timestamps are stored in *smuX.nvbuffer1*.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

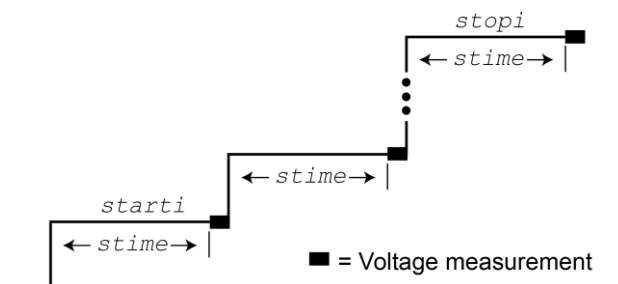
Performs a linear current sweep with voltage measured at every step (point):

1. Sets the SMU to output *starti* amperes, allows the source to settle for *stime* seconds, and then makes a voltage measurement.
2. Sets the SMU to output the next amperes step, allows the source to settle for *stime* seconds, and then makes a voltage measurement.
3. Repeats the above sequence until the voltage is measured on the *stopi* amperes step.

The linear step size is automatically calculated as follows:

$$\text{step} = (\text{stopi} - \text{starti}) / (\text{points} - 1)$$

Figure 145: SweepILinMeasureV()



Example

```
SweepILinMeasureV(smua, -1e-3, 1e-3, 0, 100)
```

This function performs a 100-point linear current sweep starting at -1 mA and stopping at $+1\text{ mA}$. Voltage is measured at every step (point) in the sweep. Because *stime* is set for 0 s, voltage is measured as quickly as possible after each current step.

Also see

[KISweep factory script](#) (on page 13-20)

SweepIListMeasureV()

This KISweep factory script function performs a current list sweep with voltage measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepIListMeasureV(smuX, ilist, stime, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <i>smua</i> refers to SMU channel A)
<i>ilist</i>	Arbitrary list of current source values; <i>ilist</i> = { <i>value1</i> , <i>value2</i> , ... <i>valueN</i> }
<i>stime</i>	Settling time in seconds; occurs after stepping the source and before making a measurement
<i>points</i>	Number of sweep points (must be ≥ 2)

Details

Data for voltage measurements, current source values, and timestamps are stored in *smuX.nvbuffer1*.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a current list sweep with voltage measured at every step (point):

1. Sets the SMU to output *ilist* amperes value, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
2. Sets the SMU to output the next *ilist* step, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
3. Repeats the above sequence until the voltage is measured for the last amperes value. The last point in the list to be measured is *points*.

Example

```
testilist = {-100e-9, 100e-9, -1e-6, 1e-6, -1e-3, 1e-3}
SweepIListMeasureV(smua, testilist, 500e-3, 6)
```

This function performs a six-point current list sweep starting at the first point in *testilist*. Voltage is measured at every step (point) in the sweep. The source is allowed to settle on each step for 500 ms before a measurement is performed.

Also see

[KISweep factory script](#) (on page 13-20)

SweepILogMeasureV()

This KISweep factory script function performs a logarithmic current sweep with voltage measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepILogMeasureV(smuX, starti, stopi, stime, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.reset()</code> applies to SMU channel A)
<i>starti</i>	Sweep start current in amperes
<i>stopi</i>	Sweep stop current in amperes
<i>stime</i>	Settling time in seconds; occurs after stepping the source and before making a measurement
<i>points</i>	Number of sweep points (must be ≥ 2)

Details

Data for voltage measurements, current source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a logarithmic current sweep with voltage measured at every step (point):

1. Sets the SMU to output *starti* amperes value, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
2. Sets the SMU to output the next amperes step, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
3. Repeats the above sequence until the voltage is measured on the *stopi* amperes step.

The source level at each step (`SourceStepLevel`) is automatically calculated as follows:

MeasurePoint = The step point number for a measurement

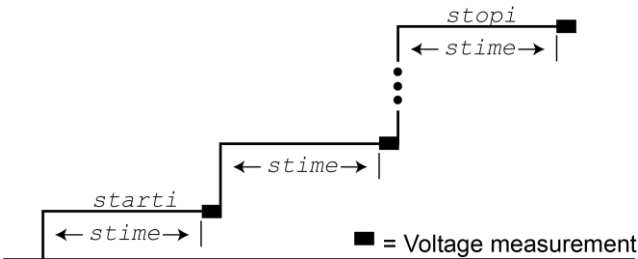
For example, for a five-point sweep (*points* = 5), a measurement is performed at *MeasurePoint* 1, 2, 3, 4, and 5.

$$\text{LogStepSize} = (\log_{10}(\text{stopi}) - \log_{10}(\text{starti})) / (\text{points} - 1)$$

$$\text{LogStep} = (\text{MeasurePoint} - 1) * (\text{LogStepSize})$$

$$\text{SourceStepLevel} = \text{antilog}(\text{LogStep}) * \text{starti}$$

Figure 146: SweepILogMeasureV()



Example

```
SweepILogMeasureV(smua, 0.01, 0.1, 0.001, 5)
```

This function performs a five-point linear current sweep starting at 10 mA and stopping at 100 mA. Voltage is measured at every step (point) in the sweep. The source is allowed to settle on each step for 1 ms before a measurement is made.

The following table contains log values and corresponding source levels for the five-point logarithmic sweep:

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
1	0.25	0.0	0.01 A
2	0.25	0.25	0.017783 A
3	0.25	0.5	0.031623 A
4	0.25	0.75	0.056234 A
5	0.25	1.0	0.1 A

Also see

[KISweep factory script](#) (on page 13-20)

SweepVLinMeasureI()

This KISweep factory script function performs a linear voltage sweep with current measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepVLinMeasureI(smuX, startv, stopv, stime, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.reset()</code> applies to SMU channel A)
<i>startv</i>	Sweep start voltage in volts
<i>stopv</i>	Sweep stop voltage in volts
<i>stime</i>	Settling time in seconds; occurs after stepping the source and before making a measurement
<i>points</i>	Number of sweep points (must be ≥ 2)

Details

Data for current measurements, voltage source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

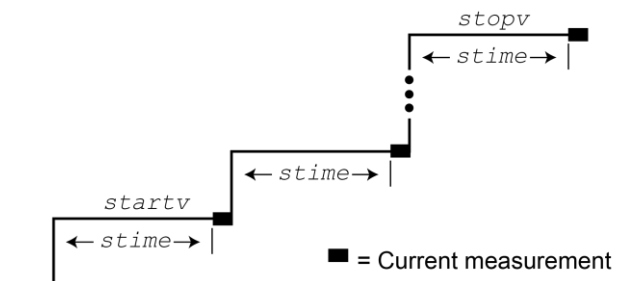
Performs a linear voltage sweep with current measured at every step (point):

1. Sets the SMU to output *startv* amperes, allows the source to settle for *stime* seconds, and then makes a current measurement.
2. Sets the SMU to output the next amperes step, allows the source to settle for *stime* seconds, and then makes a voltage measurement.
3. Repeats the above sequence until the voltage is measured on the *stopv* amperes step.

The linear step size is automatically calculated as follows:

$$\text{step} = (\text{stopv} - \text{startv}) / (\text{points} - 1)$$

Figure 147: SweepVLinMeasureI()



Example

```
SweepVLinMeasureI(smua, -1, 1, 1e-3, 1000)
```

This function performs a 1000-point linear voltage sweep starting at -1 V and stopping at +1 V. Current is measured at every step (point) in the sweep after a 1 ms source settling period.

Also see

[KISweep factory script](#) (on page 13-20)

SweepVListMeasureI()

This KISweep factory script function performs a voltage list sweep with current measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepVListMeasureI(smuX, vlist, stime, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <code>smua.reset()</code> applies to SMU channel A)
<i>vlist</i>	Arbitrary list of voltage source values; <i>vlist</i> = { <i>value1</i> , <i>value2</i> , ... <i>valueN</i> }
<i>stime</i>	Settling time in seconds; occurs after stepping the source and before making a measurement
<i>points</i>	Number of sweep points (must be ≥ 2)

Details

Data for current measurements, voltage source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a voltage list sweep with current measured at every step (point):

1. Sets the SMU to output *vlist* volts value, allows the source to settle for *stime* seconds, and then performs a current measurement.
2. Sets the SMU to output the next *vlist* volts value, allows the source to settle for *stime* seconds, and then performs a current measurement.
3. Repeats the above sequence until the current is measured for the last volts value. The last point in the list to be measured is *points*.

Example

```
myvlist = {-0.1, 0.1, -1, 1, -6, 6, -40, 40, 0, 0}
SweepVListMeasureI(smua, myvlist, 500E-3, 10)
```

This function performs a 10-point voltage list sweep starting at the first point in `myvlist`. Current is measured at every step (point) in the sweep. The source is allowed to settle on each step for 500 ms before a measurement is performed.

Also see

[KISweep factory script](#) (on page 13-20)

SweepVLogMeasureI()

This KISweep factory script function performs a logarithmic voltage sweep with current measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepVLogMeasureI(smuX, startv, stopv, stime, points)
```

<i>X</i>	Source-measure unit (SMU) channel (for example, <i>smua</i> applies to SMU channel A)
<i>startv</i>	Sweep start voltage in volts
<i>stopv</i>	Sweep stop voltage in volts
<i>stime</i>	Settling time in seconds; occurs after stepping the source and before making a measurement
<i>points</i>	Number of sweep points (must be ≥ 2)

Details

Data for current measurements, voltage source values, and timestamps are stored in *smuX.nvbuffer1*.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a logarithmic voltage sweep with current measured at every step (point):

1. Sets the SMU to output *startv* amperes, allows the source to settle for *stime* seconds, and then makes a current measurement.
2. Sets the SMU to output the next volts step, allows the source to settle for *stime* seconds, and then makes a current measurement.
3. Repeats the above sequence until the voltage is measured on the *stopv* volts step.

The source level at each step (*SourceStepLevel*) is automatically calculated as follows:

MeasurePoint = The step point number for a measurement

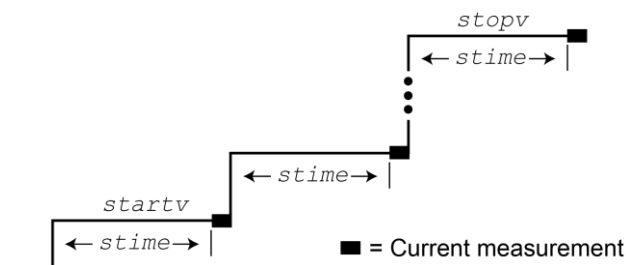
For example, for a five-point sweep (*points* = 5), a measurement is made at *MeasurePoint* 1, 2, 3, 4, and 5.

$$\text{LogStepSize} = (\log_{10}(\text{stopv}) - \log_{10}(\text{startv})) / (\text{points} - 1)$$

$$\text{LogStep} = (\text{MeasurePoint} - 1) * (\text{LogStepSize})$$

$$\text{SourceStepLevel} = \text{antilog}(\text{LogStep}) * \text{startv}$$

Figure 148: SweepVLogMeasureI()

**Example**

```
SweepVLogMeasureI(smua, 1, 10, 0.001, 5)
```

This function performs a five-point logarithmic voltage sweep starting at 1 V and stopping at 10 V. Current is measured at every step (point) in the sweep after a 1 ms source settling period.

The following table contains log values and corresponding source levels for the five-point logarithmic sweep:

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
1	0.25	0.0	1.0000 V
2	0.25	0.25	1.7783 V
3	0.25	0.5	3.1623 V
4	0.25	0.75	5.6234 V
5	0.25	1.0	10.000 V

Also see

[KISweep factory script](#) (on page 13-20)

timer.measure.t()

This function measures the elapsed time since the timer was last reset.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
time = timer.measure.t()
```

time	The elapsed time in seconds (1 µs resolution)

Example 1

```
timer.reset()
-- (intervening code)
time = timer.measure.t()
print(time)
```

This example resets the timer and measures the time since the reset.

Output:

```
1.469077e+01
```

The output varies. The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Example 2

```

beeper.enable = beeper.ON
beeper.beep(0.5, 2400)
print("reset timer")
timer.reset()
delay(0.5)
dt = timer.measure.t()
print("timer after delay:", dt)
beeper.beep(0.5, 2400)

```

Enable the beeper.
 Emit a beep and set the beeper.
 Reset the timer.
 Set a delay.
 Verify the duration of the delay before emitting another beep.
 Output:
 reset timer
 timer after delay: 5.00e-01

Also see

[timer.reset\(\)](#) (on page 15-361)

timer.reset()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timer.reset()
```

Example

```

timer.reset()
-- (intervening code)
time = timer.measure.t()
print(time)

```

Resets the timer and then measures the time since the reset.

Output:

```
1.469077e+01
```

The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Also see

[timer.measure.t\(\)](#) (on page 15-360)

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()
```

<i>N</i>	The blender number (up to six)

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].EVENT_ID

This constant contains the trigger blender event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.blender[N].EVENT_ID
```

<i>eventID</i>	Trigger event number

<i>N</i>	The blender number (up to six)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this trigger blender.

Example

```
digio.trigger[1].stimulus = trigger.blender[2].EVENT_ID
```

Set the trigger stimulus of digital I/O trigger 1 to be controlled by the trigger blender 2 event.

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	Instrument reset Trigger blender N reset Recall setup	Not saved	false (AND mode)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

<i>orenable</i>	The type of operation: ■ true: OR operation ■ false: AND operation
<i>N</i>	The blender number (up to six)

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] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
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 15-364)

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
```

<i>overrun</i>	Trigger blender overrun state (true or false)
<i>N</i>	The blender number (up to six)

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 15-364)

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()
```

<i>N</i>	The trigger event blender (up to six)

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 15-363)

[trigger.blender\[N\].overrun](#) (on page 15-363)

[trigger.blender\[N\].stimulus\[M\]](#) (on page 15-365)

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

```
eventID = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = eventID
```

<i>eventID</i>	The event that triggers the blender action; see Details
<i>N</i>	An integer representing the trigger event blender (up to six)
<i>M</i>	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event. The *eventID* parameter can be the event ID of any trigger event.

Use zero to disable the blender input.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	Event description
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
display.trigger.EVENT_ID	Occurs when the TRIG key on the virtual front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <i>device_trigger</i> ; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[N].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[N].EVENT_ID	Occurs when a delay expires

Trigger event IDs*	
Event ID	Event description
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Example

```

digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[5].mode = digio.TRIG_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID

```

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 15-364)

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)
```

<i>triggered</i>	Trigger detection indication for blender
<i>N</i>	The trigger blender (up to six) on which to wait
<i>timeout</i>	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.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[5].mode = digio.TRIG_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
print(trigger.blender[1].wait(3))

```

Generate a trigger blender 1 event when a digital I/O trigger happens either on line 3 or 5. Wait three seconds while checking if trigger blender 1 event has occurred. If the blender trigger event has happened, then `true` is output. If the trigger event has not happened, then `false` is output after the timeout expires.

Also see

[trigger.blender\[N\].clear\(\)](#) (on page 15-362)

trigger.clear()

This function clears the command interface trigger event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

The trigger event detector indicates if a trigger event has been detected since the last `trigger.wait()` call. `trigger.clear()` clears the trigger event detector and discards the history of command interface trigger events.

Also see

[trigger.wait\(\)](#) (on page 15-376)

trigger.EVENT_ID

This constant contains the command interface trigger event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.EVENT_ID
```

<code>eventID</code>	The event ID for the command interface triggers

Details

You can set the stimulus of any trigger object to the value of this constant to have the trigger object respond to command interface trigger events.

Example

```
trigger.timer[1].stimulus = trigger.EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the command interface trigger event.

Also see

None

trigger.generator[N].assert()

This function generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.generator[N].assert()
```

<i>N</i>	The generator number (1 or 2)

Details

Use this function to directly trigger events from the command interface or a script. For example, you can trigger a sweep while the instrument is under script control.

Example

<pre>trigger.generator[2].assert()</pre>	Generates a trigger event on generator 2.

Also see

[trigger.generator\[N\].EVENT_ID](#) (on page 15-368)

trigger.generator[N].EVENT_ID

This constant identifies the trigger event generated by the trigger event generator.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.generator[N].EVENT_ID
```

<i>eventID</i>	The trigger event number
<i>N</i>	The generator number (1 or 2)

Details

This constant is an identification number that identifies events generated by this generator.

To have another trigger object respond to trigger events generated by this generator, set the other object's stimulus attribute to the value of this constant.

Example

<pre>digio.trigger[5].stimulus = trigger.generator[2].EVENT_ID</pre>
Uses a trigger event on generator 2 to be the stimulus for digital I/O trigger line 5.

Also see

[trigger.generator\[N\].assert\(\)](#) (on page 15-368)

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()
```

<i>N</i>	Trigger timer number (1 to 8)

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 15-369)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Not saved	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

<i>count</i>	Number of times to repeat the trigger (0 to 1,048,575)

<i>N</i>	Trigger timer number (1 to 8)

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

```
print(trigger.timer[1].count)
```

Read trigger count for timer number 1.

Also see[trigger.timer\[N\].clear\(\)](#) (on page 15-369)[trigger.timer\[N\].delay](#) (on page 15-370)[trigger.timer\[N\].reset\(\)](#) (on page 15-373)

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	Instrument reset Recall setup Trigger timer N reset	Not saved	10e-6 (10 µs)

Usage

```
interval = trigger.timer[N].delay  
trigger.timer[N].delay = interval
```

<i>interval</i>	Delay interval in seconds (0.5 µs to 100 ks)
<i>N</i>	Trigger timer number (1 to 8)

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 is used the next time the timer is triggered.

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 15-373)

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	Instrument reset Recall setup Trigger timer <i>N</i> reset	Not saved	10e-6 (10 μ s)

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

<i>intervals</i>	Table of delay intervals in seconds
<i>N</i>	Trigger timer number (1 to 8)

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 μ s.

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 $\geq 50 \mu$ s.

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 μ s, 100 μ s, and 150 μ s).

Read the delay list on trigger timer 3.

Output:

```
5e-05
0.0001
0.00015
```

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 15-373)

trigger.timer[N].EVENT_ID

This constant specifies the trigger timer event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.timer[N].EVENT_ID
```

<i>eventID</i>	The trigger event number
<i>N</i>	Trigger timer number (1 to 8)

Details

This constant is an identification number that identifies events generated by this timer.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to events from this timer.

Example

```
trigger.timer[1].stimulus = tsplink.trigger[2].EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the TSP-Link trigger 2 event.

Also see

None

trigger.timer[N].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	Instrument reset Recall setup Trigger timer N clear Trigger timer N reset	Not applicable	false

Usage

```
overrun = trigger.timer[N].overrun
```

<i>overrun</i>	Trigger overrun state (true or false)
<i>N</i>	Trigger timer number (1 to 8)

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.

Delay overrun indications are provided in the status model.

Example

```
print(trigger.timer[1].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 15-373)

trigger.timer[N].passthrough

This attribute enables or disables the timer trigger pass-through mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Not saved	false (disabled)

Usage

```
passthrough = trigger.timer[N].passthrough
trigger.timer[N].passthrough = passthrough
```

<i>passthrough</i>	The state of pass-through mode; set to one of the following values: ■ <code>true</code>: Enabled ■ <code>false</code>: Disabled
<i>N</i>	Trigger timer number (1 to 8)

Details

When pass-through mode is enabled, triggers are passed through immediately and initiate the delay. When disabled, a trigger only initiates a delay.

Example

```
trigger.timer[1].passthrough = true
```

Enables pass-through mode on trigger timer 1.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 15-373)

trigger.timer[N].reset()

This function resets some of the trigger timer settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].reset()
```

<i>N</i>	Trigger timer number (1 to 8)

Details

The `trigger.timer[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].passthrough`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

```
trigger.timer[1].reset()
```

Resets the attributes associated with timer 1 back to factory default values.

Also see

[trigger.timer\[N\].count](#) (on page 15-369)

[trigger.timer\[N\].delay](#) (on page 15-370)

[trigger.timer\[N\].delaylist](#) (on page 15-371)

[trigger.timer\[N\].overrun](#) (on page 15-372)

[trigger.timer\[N\].passthrough](#) (on page 15-373)

[trigger.timer\[N\].stimulus](#) (on page 15-374)

trigger.timer[N].stimulus

This attribute specifies which event starts the timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Not saved	0

Usage

```
eventID = trigger.timer[N].stimulus
```

```
trigger.timer[N].stimulus = eventID
```

<i>eventID</i>	The event that triggers the timer delay
<i>N</i>	Trigger timer number (1 to 8)

Details

The *eventID* parameter may be one of the trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measurement action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Set this attribute to the *eventID* of any trigger event to cause the timer to start when that event occurs.

Set this attribute to zero (0) to disable event processing.

Example

```
print(trigger.timer[1].stimulus)
```

Prints the event that will start a trigger 1 timer action.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 15-373)

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)
```

<i>triggered</i>	Trigger detection indication
<i>N</i>	Trigger timer number (1 to 8)
<i>timeout</i>	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 15-369)

trigger.wait()

This function waits for a command interface trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
triggered = trigger.wait(timeout)
```

<i>triggered</i>	Select: ■ <code>true</code>: A trigger was detected during the timeout period ■ <code>false</code>: No triggers were detected during the timeout period
<i>timeout</i>	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 USBTMC TRIGGER message is received (USB only)
- A VXI-11 device_trigger method is invoked (VXI-11 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 seconds for a trigger.
If *false* is returned, no trigger was detected during the 10-second timeout.
If *true* is returned, a trigger was detected.

Also see

[trigger.clear\(\)](#) (on page 15-367)

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
```

<i>groupNumber</i>	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 14-60)

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
```

<code>masterNodeNumber</code>	The node number of the master node

Details

After doing a TSP-Link reset (`tsplink.reset()`), use this attribute to access the node number of the master in a set of instruments connected over TSP-Link.

Example

```
LinkMaster = tsplink.master
```

Store the TSP-Link master node number in a variable called LinkMaster.

Also see

[tsplink.reset\(\)](#) (on page 15-380)

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	1

Usage

```
nodeNumber = tsplink.node  
tsplink.node = nodeNumber
```

<code>nodeNumber</code>	The node number of the instrument or enclosure (1 to 64)

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 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.reset\(\)](#) (on page 15-380)

[tsplink.state](#) (on page 15-381)

tsplink.readbit()

This function reads the state of a TSP-Link synchronization line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readbit(N)
```

data	The state of the synchronization line
N	The trigger line (1 to 3)

Details

Returns a value of zero (0) if the line is low and 1 if the line is high.

Example

```
data = tsplink.readbit(3)
print(data)
```

Assume line 3 is set high, and it is then read.
Output:
1.000000e+00

Also see

[tsplink.readport\(\)](#) (on page 15-379)
[tsplink.writebit\(\)](#) (on page 15-390)

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

[TSP-Link trigger lines](#) (on page 7-41)
[tsplink.readbit\(\)](#) (on page 15-379)
[tsplink.writebit\(\)](#) (on page 15-390)
[tsplink.writeport\(\)](#) (on page 15-390)

tsplink.reset()

This function initializes (resets) all nodes (instruments) in the TSP-Link system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.reset()
nodesFound = tsplink.reset(expectedNodes)
```

<i>nodesFound</i>	The number of nodes actually found on the system
<i>expectedNodes</i>	The number of nodes expected on the system (1 to 64)

Details

This function erases all information regarding other nodes connected on the TSP-Link system and regenerates the system configuration. This function must be called at least once before any remote nodes can be accessed. If the node number for any instrument is changed, the TSP-Link must be reset again.

If *expectedNodes* is not given, this function generates an error if no other nodes are found on the TSP-Link network.

If *nodesFound* is less than *expectedNodes*, an error is generated. Note that the node on which the command is running is counted as a node. For example, giving an expected node count of 1 does not generate any errors, even if there are no other nodes on the TSP-Link network.

Also returns the number of nodes found.

Example

```
nodesFound = tsplink.reset(2)
print("Nodes found = " .. nodesFound)
```

Perform a TSP-Link reset and indicate how many nodes are found.
Sample output if two nodes are found:
Nodes found = 2
Sample output if fewer nodes are found and if `localnode.showerrors = 1`:
1219, TSP-Link found fewer nodes than expected
Nodes found = 1

Also see[localnode.showerrors](#) (on page 15-161)[tsplink.node](#) (on page 15-378)[tsplink.state](#) (on page 15-381)

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.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.node](#) (on page 15-378)[tsplink.reset\(\)](#) (on page 15-380)

tsplink.trigger[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event ID.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].assert()
```

N	The trigger line (1 to 3)

Details

The set pulse width determines how long the trigger is asserted.

Example

```
tsplink.trigger[2].assert()
```

Asserts trigger on trigger line 2.

Also see

[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)
[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)
[tsplink.trigger\[N\].pulsewidth](#) (on page 15-386)
[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)
[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

tsplink.trigger[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

```
tsplink.trigger[N].clear()
```

<i>N</i>	The trigger line (1 to 3) to clear

Details

The trigger event detector enters the detected state when an event is detected.

`tsplink.trigger[N].clear()` clears a trigger event detector, discards the history of the trigger line, and clears the `tsplink.trigger[N].overrun` attribute.

Example

```
tsplink.trigger[2].clear()
```

Clears trigger event on synchronization line 2.

Also see

[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)
[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)
[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

tsplink.trigger[N].EVENT_ID

This constant identifies the number that is used for the trigger events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = tsplink.trigger[N].EVENT_ID
```

<i>eventID</i>	The trigger event number
<i>N</i>	The trigger line (1 to 3)

Details

This number is used by the TSP-Link trigger line when it detects an input trigger.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this line.

Example

```
trigger.timer[1].stimulus = tsplink.trigger[2].EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the TSP-Link trigger 2 event.

Also see

None

tsplink.trigger[N].mode

This attribute defines the trigger operation and detection mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup TSP-Link trigger N reset	Not saved	0 (tsplink.TRIG_BYPASS)

Usage

```
mode = tsplink.trigger[N].mode
tsplink.trigger[N].mode = mode
```

<i>mode</i>	The trigger mode
<i>N</i>	The trigger line (1 to 3)

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger line.

The setting for the *mode* parameter can be one of the values shown in the following table.

Mode	Number value	Description
<code>tsplink.TRIG_BYPASS</code>	0	Allows direct control of the line as a digital I/O line.
<code>tsplink.TRIG_FALLING</code>	1	Detects falling-edge triggers as input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_RISING</code>	2	If the programmed state of the line is high, the <code>tsplink.TRIG_RISING</code> mode behaves similarly to <code>tsplink.TRIG_RISINGA</code>. If the programmed state of the line is low, the <code>tsplink.TRIG_RISING</code> mode behaves similarly to <code>tsplink.TRIG_RISINGM</code>. Use <code>tsplink.TRIG_RISINGA</code> if the line is in the high output state. Use <code>tsplink.TRIG_RISINGM</code> if the line is in the low output state.
<code>tsplink.TRIG_EITHER</code>	3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_SYNCHRONOUSA</code>	4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low.
<code>tsplink.TRIG_SYNCHRONOUS</code>	5	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
<code>tsplink.TRIG_SYNCHRONOUSM</code>	6	Detects rising-edge triggers as an input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_RISINGA</code>	7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_RISINGM</code>	8	Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.

When programmed to any mode except `tsplink.TRIG_BYPASS`, the output state of the I/O line is controlled by the trigger logic and the user-specified output state of the line is ignored.

When the trigger mode is set to `tsplink.TRIG_RISING`, the user-specified output state of the line is examined. If the output state selected when the mode is changed is high, the actual mode that is used is `tsplink.TRIG_RISINGA`. If the output state selected when the mode is changed is low, the actual mode that is used is `tsplink.TRIG_RISINGM`.

The *mode* parameter stores the trigger mode as a numeric value when the attribute is read.

To control the line state, use the `tsplink.TRIG_BYPASS` mode with the `tsplink.writebit()` and the `tsplink.writeport()` commands.

Example

```
tsplink.trigger[3].mode = tsplink.TRIG_RISINGM
Sets the trigger mode for synchronization line 3 to tsplink.TRIG_RISINGM.
```

Also see

[digio.writebit\(\)](#) (on page 15-63)
[digio.writeport\(\)](#) (on page 15-63)
[tsplink.trigger\[N\].assert\(\)](#) (on page 15-381)
[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)
[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)
[tsplink.trigger\[N\].reset\(\)](#) (on page 15-387)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)
[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

tsplink.trigger[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 trigger N clear TSP-Link trigger N reset	Not applicable	Not applicable

Usage

```
overrun = tsplink.trigger[N].overrun
```

<i>overrun</i>	Trigger overrun state
<i>N</i>	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. Output trigger overrun indications are provided in the status model.

Example

```
print(tsplink.trigger[1].overrun)
```

If an event was ignored, displays `true`; if an event was not ignored, displays `false`.

Also see

[tsplink.trigger\[N\].assert\(\)](#) (on page 15-381)
[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)
[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)
[tsplink.trigger\[N\].reset\(\)](#) (on page 15-387)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)
[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

tsplink.trigger[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	Instrument reset TSP-Link trigger N reset Recall setup	Not saved	10e-6 (10 µs)

Usage

```
width = tsplink.trigger[N].pulsewidth
tsplink.trigger[N].pulsewidth = width
```

<i>width</i>	The pulse width (in seconds)
<i>N</i>	The trigger line (1 to 3)

Details

Setting the pulse width to 0 (seconds) asserts the trigger indefinitely.

Example

```
tsplink.trigger[3].pulsewidth = 20e-6
Sets pulse width for trigger line 3 to 20 µs.
```

Also see

[tsplink.trigger\[N\].release\(\)](#) (on page 15-386)

tsplink.trigger[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

```
tsplink.trigger[N].release()
```

<i>N</i>	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.trigger[3].release()
Releases trigger line 3.
```

Also see

[tsplink.trigger\[N\].assert\(\)](#) (on page 15-381)
[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)
[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)
[tsplink.trigger\[N\].pulsewidth](#) (on page 15-386)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)
[tsplink.trigger\[N\].wait\(\)](#) (on page 15-389)

tsplink.trigger[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.trigger[N].reset()
```

<i>N</i>	The trigger line (1 to 3)

Details

The `tsplink.trigger[N].reset()` function resets the following attributes to their factory defaults:

- `tsplink.trigger[N].mode`
- `tsplink.trigger[N].stimulus`
- `tsplink.trigger[N].pulsewidth`

This also clears `tsplink.trigger[N].overrun`.

Example

```
tsplink.trigger[3].reset()
```

Resets TSP-Link trigger line 3 attributes back to factory default values.

Also see

[tsplink.trigger\[N\].mode](#) (on page 15-383)
[tsplink.trigger\[N\].overrun](#) (on page 15-385)
[tsplink.trigger\[N\].pulsewidth](#) (on page 15-386)
[tsplink.trigger\[N\].stimulus](#) (on page 15-388)

tsplink.trigger[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	Instrument reset Recall setup TSP-Link trigger N reset	Not saved	0

Usage

```
eventID = tsplink.trigger[N].stimulus
tsplink.trigger[N].stimulus = eventID
```

<i>eventID</i>	The event identifier for the triggering event
<i>N</i>	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to zero (0).

Do not use this attribute when triggering under script control. Use

`tsplink.trigger[N].assert()` instead.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	Event description
<code>smuX.trigger.SWEEPING_EVENT_ID</code>	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
<code>smuX.trigger.ARMED_EVENT_ID</code>	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
<code>smuX.trigger.SOURCE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a source action
<code>smuX.trigger.MEASURE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a measurement action
<code>smuX.trigger.PULSE_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a pulse
<code>smuX.trigger.SWEEP_COMPLETE_EVENT_ID</code>	Occurs when the SMU completes a sweep
<code>smuX.trigger.IDLE_EVENT_ID</code>	Occurs when the SMU returns to the idle state
<code>digio.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a digital I/O line
<code>tsplink.trigger[N].EVENT_ID</code>	Occurs when an edge is detected on a TSP-Link line
<code>lan.trigger[N].EVENT_ID</code>	Occurs when the appropriate LXI trigger packet is received on LAN trigger object <i>N</i>
<code>display.trigger.EVENT_ID</code>	Occurs when the TRIG key on the virtual front panel is pressed
<code>trigger.EVENT_ID</code>	Occurs when a *TRG command is received on the remote interface USB only: Occurs when a USBTMC TRIGGER message is received VXI-11 only: Occurs with the VXI-11 command <code>device_trigger</code> ; reference the VXI-11 standard for additional details on the device trigger operation
<code>trigger.blender[N].EVENT_ID</code>	Occurs after a collection of events is detected

Trigger event IDs*	
Event ID	Event description
<code>trigger.timer[N].EVENT_ID</code>	Occurs when a delay expires
<code>trigger.generator[N].EVENT_ID</code>	Occurs when the <code>trigger.generator[N].assert()</code> function is executed
* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).	

Example

```
print(tsplink.trigger[3].stimulus)
```

Prints the event that will start TSP-Link trigger line 3 action.

Also see

[tsplink.trigger\[N\].assert\(\)](#) (on page 15-381)

[tsplink.trigger\[N\].reset\(\)](#) (on page 15-387)

tsplink.trigger[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = tsplink.trigger[N].wait(timeout)
```

<i>triggered</i>	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
<i>N</i>	The trigger line (1 to 3)
<i>timeout</i>	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events were detected since the last time `tsplink.trigger[N].wait()` or `tsplink.trigger[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 = tsplink.trigger[3].wait(10)
print(triggered)
```

Waits up to 10 seconds for a trigger on TSP-Link® line 3.

If `false` is returned, no trigger was detected during the 10-second timeout.

If `true` is returned, a trigger was detected.

Also see

[tsplink.trigger\[N\].clear\(\)](#) (on page 15-382)

tsplink.writebit()

This function sets a TSP-Link trigger line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writebit(N, data)
```

<i>N</i>	The trigger line (1 to 3)
<i>data</i>	The value to write to the bit: ■ Low: 0 ■ High: 1

Details

Use `tsplink.writebit()` and `tsplink.writeport()` to control the output state of the trigger line when trigger operation is set to `tsplink.TRIG_BYPASS`.

If the output line is write-protected by the `tsplink.writeprotect` attribute, this command is ignored.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

```
tsplink.writebit(3, 0)
Sets trigger line 3 low (0).
```

Also see

[tsplink.readbit\(\)](#) (on page 15-379)
[tsplink.readport\(\)](#) (on page 15-379)
[tsplink.writeprotect](#) (on page 15-391)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writeport(data)
```

<i>data</i>	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).

Write-protected lines are not changed.

Use the `tsplink.writebit()` and `tsplink.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `tsplink.TRIG_BYPASS`.

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.readbit\(\)](#) (on page 15-379)
[tsplink.readport\(\)](#) (on page 15-379)
[tsplink.writebit\(\)](#) (on page 15-390)

tsplink.writeprotect

This attribute contains the write-protect mask that protects bits from changes by the `tsplink.writebit()` and `tsplink.writeport()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0

Usage

```
mask = tsplink.writeprotect
tsplink.writeprotect = mask
```

<i>mask</i>	An integer that specifies the value of the bit pattern for write-protect; set bits to 1 to write-protect the corresponding TSP-Link trigger line

Details

The binary equivalent of *mask* indicates the mask to be set for the TSP-Link trigger line. For example, a *mask* value of 5 has a binary equivalent of 101. This *mask* write-protects TSP-Link trigger lines 1 and 3.

Example

```
tsplink.writeprotect = 5
Write-protects TSP-Link trigger lines 1 and 3.
```

Also see

[Controlling digital I/O lines](#) (on page 7-38)
[tsplink.readbit\(\)](#) (on page 15-379)
[tsplink.readport\(\)](#) (on page 15-379)
[tsplink.writebit\(\)](#) (on page 15-390)
[tsplink.writeport\(\)](#) (on page 15-390)

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)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

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 15-393)

[tspnet.readavailable\(\)](#) (on page 15-398)

[tspnet.write\(\)](#) (on page 15-404)

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")
```

<i>connectionID</i>	The connection ID to be used as a handle in all other <code>tspnet</code> function calls
<i>ipAddress</i>	IP address to which to connect in a string; accepts IP address or host name when trying to connect
<i>portNumber</i>	Port number (default 5025)
<i>initString</i>	Initialization string to send to <i>ipAddress</i>

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 2606B 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 2606B sends an `abort` command to the remote device on connection.

The 2606B also enables TSP prompts on the remote device and error management. The 2606B places remote errors from the TSP-enabled device in its own error queue and prefaces these errors with `Remote Error`, followed by an error description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show errors by changing `localnode.showerrors` 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 15-158)

[localnode.showerrors](#) (on page 15-161)

[tspnet.tsp.abortonconnect](#) (on page 15-401)

[tspnet.disconnect\(\)](#) (on page 15-394)

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)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

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 15-393)

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)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>commandString</i>	The command to send to the remote device
<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valueN</i>	The <i>N</i> th value decoded from the response message; there is one return value for each format specifier in the format string
<i>...</i>	One or more values separated with commas
<i>formatString</i>	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 2606B 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 2606B 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 2606B places any remotely generated errors into its error 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 15-393)
[tspnet.read\(\)](#) (on page 15-397)
[tspnet.termination\(\)](#) (on page 15-399)
[tspnet.write\(\)](#) (on page 15-404)

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)
```

<i>idnString</i>	The returned *IDN? string
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

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 2606B, 1398687,  
1.0.0
```

Also see

[tspnet.connect\(\)](#) (on page 15-393)

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)
```

<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valueN</i>	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
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>formatString</i>	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 2606B 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 2606B reads from a TSP-enabled remote instrument, the 2606B removes Test Script Processor (TSP®) prompts and places any errors it receives from the remote instrument into its own error queue. The 2606B prefaces errors from the remote device with "Remote Error," followed by the error number and error 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 15-393)
[tspnet.readavailable\(\)](#) (on page 15-398)
[tspnet.timeout](#) (on page 15-400)
[tspnet.write\(\)](#) (on page 15-404)

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)
```

<i>bytesAvailable</i>	The number of bytes available to be read from the connection
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

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 create data.
Wait for data to be available.

Also see

[tspnet.connect\(\)](#) (on page 15-393)
[tspnet.read\(\)](#) (on page 15-397)

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)
```

<i>type</i>	An enumerated value indicating the termination type: 1 or <code>tspnet.TERM_LF</code> 4 or <code>tspnet.TERM_CR</code> 2 or <code>tspnet.TERM_CRLF</code> 3 or <code>tspnet.TERM_LFCR</code>
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>termSequence</i>	The termination sequence

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 15-393)

[tspnet.disconnect\(\)](#) (on page 15-394)

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	Instrument reset Recall setup	Not saved	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 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 15-393)

[tspnet.execute\(\)](#) (on page 15-395)

[tspnet.read\(\)](#) (on page 15-397)

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)
```

<i>connectionID</i>	Integer value used as a handle for other <code>tspnet</code> 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	Instrument reset Recall setup	Not saved	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

<i>value</i>	■ 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 does 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 15-393)

tspnet.tsp.rhtablecopy()

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.rhtablecopy(connectionID, "name")
table = tspnet.tsp.rhtablecopy(connectionID, "name", startIndex, endIndex)
```

<i>table</i>	A copy of the synchronous table or a string
<i>connectionID</i>	Integer value used as a handle for other <code>tspnet</code> commands
<i>name</i>	The full name of the reading buffer name and synchronous table to copy
<i>startIndex</i>	Integer start value
<i>endIndex</i>	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

```
t = tspnet.tsp.rhtablecopy(testConnection, "testRemotebuffername.readings", 1, 3)
print(t[1], t[2], t[3])
```

Copy the specified readings table for buffer items 1 through 3, then display the first three readings.

Example output:

```
4.56534e-01
4.52675e-01
4.57535e-01
```

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	Yes			

Usage

```
tspnet.tsp.runscript(connectionID, "script")
tspnet.tsp.runscript(connectionID, "name", "script")
```

<i>connectionID</i>	Integer value used as an identifier for other <code>tspnet</code> commands
<i>name</i>	The name that is assigned to the script
<i>script</i>	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.

If no name is specified, the script is loaded as the anonymous script.

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 15-395)

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")
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>inputString</i>	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 2606B 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 15-393)

[tspnet.read\(\)](#) (on page 15-397)

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")
```

<i>name</i>	The name of the string; the key of the key-value pair
<i>value</i>	The string to associate with <i>name</i> ; 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 15-405)

[userstring.delete\(\)](#) (on page 15-406)

[userstring.get\(\)](#) (on page 15-407)

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
```

<i>name</i>	The name of the string; the key of the key-value pair
<i>body</i>	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 15-404)

[userstring.delete\(\)](#) (on page 15-406)

[userstring.get\(\)](#) (on page 15-407)

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 15-404)

[userstring.catalog\(\)](#) (on page 15-405)

[userstring.get\(\)](#) (on page 15-407)

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")
```

<i>value</i>	The value of the user-defined string key-value pair
<i>name</i>	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 15-404)

[userstring.catalog\(\)](#) (on page 15-405)

[userstring.delete\(\)](#) (on page 15-406)

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)
```

<i>group</i>	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 group.

Example 1

<code>waitcomplete()</code>
Waits for all nodes in the local group.

Example 2

<code>waitcomplete(G)</code>
Waits for all nodes in group G.

Example 3

<code>waitcomplete(0)</code>
Waits for all nodes on the TSP-Link network.

Also see

None

Common commands

In this section:

Common command summary	16-1
Script command equivalents	16-2
Command reference	16-3

Common command summary

The IEEE Std 488.2 common commands that are supported by the 2606B are summarized in the following table. Although commands are shown in uppercase, common commands are not case sensitive, so you can use either uppercase or lowercase. Although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the 2606B does not strictly conform to that standard.

NOTE

Unlike other commands, like those listed in [TSP commands](#) (on page 15-7), each common command must be sent in a separate message.

The common commands cannot be used in scripts.

Command	Name	Description
*CLS	Clear status	Clears all event registers and Error Queue. For detailed information including status commands, see the Status model (on page 10-1).
*ESE <i>mask</i>	Event enable command	Program the Standard Event Status Enable Register. For detailed information including status commands, see the Status model (on page 10-1).
*ESE?	Event enable query	Read the Standard Event Status Enable Register. For detailed information including status commands, see the Status model (on page 10-1).
*ESR?	Event status register query	Read/clear the Standard Event Enable Register. For detailed information including status commands, see the Status model (on page 10-1).
*IDN?	Identification query	Returns the manufacturer, model number, serial number, and firmware revision levels of the unit. For detailed information, see Identification query: *IDN? (on page 16-3).

Command	Name	Description
*OPC	Operation complete command	Set the Operation Complete bit in the Standard Event Register after all pending commands, including overlapped commands, have completed. For detailed information, see Operation complete and query: *OPC and *OPC? (on page 16-3).
*OPC?	Operation complete query	Places an ASCII "1" into the output queue when all selected device operations have completed. For detailed information, see Operation complete and query: *OPC and *OPC? (on page 16-3).
*RST	Reset command	Returns the 2606B to default conditions. For detailed information, see Reset: *RST (on page 16-3).
*SRE <i>mask</i>	Service request enable command	Programs the Service Request Enable Register. For detailed information including status commands, see the Status model (on page 10-1).
*SRE?	Service request enable query	Reads the Service Request Enable Register. For detailed information including status commands, see the Status model (on page 10-1).
*STB?	Status byte query	Reads the status byte register. For detailed information including status commands, see the Status model (on page 10-1).
*TRG	Trigger command	Generates the <code>trigger.EVENT_ID</code> trigger event for use with the trigger model. For detailed information, see Trigger: *TRG (on page 16-4).
*TST?	Self-test query	Returns a 0. For detailed information, see Self-test query: *TST? (on page 16-4).
*WAI	Wait-to-continue command	Waits until all previous commands have completed. For detailed information, see Wait-to-continue: *WAI (on page 16-4).

Script command equivalents

The TSP commands that can be included in scripts that are equivalent to the common commands are defined in the table below.

Common command	Script command equivalent
*CLS	<code>status.reset()</code>
*ESE?	<code>print(tostring(status.standard.enable))</code>
*ESE <i>mask</i>	<code>status.standard.enable = mask</code>
*ESR?	<code>print(tostring(status.standard.event))</code>
*IDN?	<code>print([[Keithley Instruments, Model]]..localnode.model..[[,]].localnode.serialno.. [[,]].localnode.revision)</code>
*OPC?	<code>waitcomplete() print([[1]])</code>
*OPC	<code>opc()</code>
*RST	<code>reset()</code>
*SRE?	<code>print(tostring(status.request_enable))</code>
*SRE <i>mask</i>	<code>status.request_enable = mask</code>
*STB?	<code>print(tostring(status.condition))</code>
*TRG	Not available
*TST?	<code>print([[0]])</code>
*WAI	<code>waitcomplete()</code>

Command reference

Details of all common commands (except those associated with the status model) are described below.

NOTE

Status command usage is in the [Status model](#) (on page 10-1).

Identification query: *IDN?

Retrieves the identification string.

*IDN? Command that reads ID information

The identification string includes the manufacturer, model number, serial number, and firmware revision levels. This string is sent in the following format:

```
Keithley Instruments, Model 2606B, xxxxxxxx, yyyy
```

Where:

xxxxxxx is the serial number

yyyy is the firmware revision level

Operation complete and query: *OPC and *OPC?

Wait for pending overlapped commands to complete.

*OPC Operation complete command that sets the OPC bit

*OPC? Operation complete query that places a "1" in the output queue

When *OPC is sent, the OPC bit in the Standard Event Register (see [Status model](#) (on page 10-1)) is set when all overlapped commands complete. The *OPC? command places an ASCII "1" in the output queue when all previous overlapped commands complete.

Reset: *RST

Returns the instrument to default conditions.

*RST Command that returns the instrument to default conditions

When the *RST command is sent, the instrument returns to the default conditions. This performs the same actions as [reset\(\)](#) (on page 15-176).

Self-test query: *TST?

Requests self-test results.

*TST? Places a zero (0) in the output queue

This command always places a zero (0) in the output queue. This command is included for common command compatibility only; the 2606B does not actually perform a self-test.

Trigger: *TRG

Generates a command interface trigger event for the trigger model.

*TRG This command generates the `trigger.EVENT_ID` trigger event for the trigger model

The `trigger.EVENT_ID` is a constant that contains the command interface trigger event number. You can set the stimulus of any trigger object to the value of this constant to have the trigger object respond to the trigger events generated by this command. See [trigger.EVENT_ID](#) (on page 15-367) and [Using the remote trigger model](#) (on page 7-3).

Wait-to-continue: *WAI

Suspends the execution of subsequent commands until all previous overlapped commands are finished.

*WAI This pauses until overlapped commands are complete

Two types of device commands exist:

- **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 finish before the next command is executed.

The *WAI command suspends the execution of subsequent commands until the instrument operations of all previous overlapped commands are finished. The *WAI command is not needed for sequential commands.

In this section:

Verification	17-1
Adjustment	17-12

Verification

WARNING

The information in this section is intended for qualified service personnel only, as described by the types of product users in the Safety precautions pages, provided at the beginning of this document. Do not attempt these procedures unless you are qualified to do so.

Some of these procedures may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

Use the procedures in this section to verify that the 2606B accuracy is within the limits stated in the instrument's one-year accuracy specifications. Perform the verification procedures:

- When you first receive the instrument to make sure that it was not damaged during shipment.
- To verify that the instrument meets factory specifications.
- To determine if calibration is required.
- After performing a calibration adjustment to make sure the instrument was adjusted properly.

The 2606B contains two independent SMU modules. Each module requires separate verification and adjustment. The instructions in this section use the front panel. You can also use remote programming commands. Refer to [Instrument programming](#) (on page 14-1) for information on using commands to control the instrument.

The instructions in this section use the web interface (refer to [Virtual Front Panel](#) (on page 3-24)). You can also use remote programming commands.

NOTE

If the instrument is still under warranty and its performance is outside specified limits, contact your Keithley Instruments representative or the factory to determine the correct course of action.

Calibration test requirements

Be sure that you perform the calibration tests:

- Under the proper environmental conditions.
- After the specified warmup period.
- Using the correct line voltage.
- Using the proper test equipment.
- Using the specified output signal and reading limits.

NOTE

Product specifications are subject to change. Listed uncertainties and test limits are provided only as examples. Always verify values against the most recent product specifications.

Environmental conditions

Conduct your performance calibration procedures in a test environment with:

- An ambient temperature of 18 °C to 28 °C.
- A relative humidity of less than 70 percent unless otherwise noted.

NOTE

Product specifications that are listed as 18 °C to 28 °C assume adjustment has been done at 23 °C. If the 2606B is adjusted at a different temperature, the specifications apply to ± 5 °C of that adjustment temperature.

Line power

The 2606B requires a line voltage of 100 V to 240 V and a line frequency of 50 Hz or 60 Hz. Perform calibration tests in this range.

Warmup period

Allow the 2606B to warm up for at least two hours before conducting the calibration procedures.

If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the internal temperature of the instrument to stabilize. Typically, allow one extra hour to stabilize an instrument that is 10 °C outside the specified temperature range.

Also allow test equipment to warm up for the minimum time specified by the manufacturer.

Recommended verification equipment

The following table summarizes recommended maximum allowable test equipment uncertainty for verification points. Total test equipment measurement uncertainty should meet or be less than the listed values at each test point. Generally, test equipment uncertainty should be at least four times better than corresponding 2606B specifications.

Recommended verification equipment				
Description	Manufacturer and model	Accuracy		
Digital multimeter	Keithley Instruments Model 2002 or Keysight 3458A	DC voltage ¹	90 mV:	±8 ppm
			0.9 V:	±5 ppm
			5.4 V:	±4 ppm
			18 V:	±6 ppm
		DC current ²	90 nA:	±430 ppm
			0.9 mA:	±45 ppm
			9 µA:	±25 ppm
			90 µA:	±23 ppm
			0.9 mA:	±20 ppm
			9 mA:	±20 ppm
			90 mA:	±35 ppm
			0.9 A:	±110 ppm
0.5 Ω, 250 W, 0.1% precision resistor	Isotek RUG-Z-R500-0.1-TK3	Resistance ³	0.5 Ω:	±125 ppm
1 GΩ, 200 V, 1% standard	Keithley Instruments 2600-STD-RES	Resistance ⁴	1 GΩ:	±250 ppm
¹ Ninety-day specifications show full-range accuracy of recommended model used for specified measurement point.				
² Id.				
³ Resistor used to test 3 A range should be characterized to uncertainty shown using resistance function of digital multimeter before use.				
⁴ Standard is a guarded and characterized 1 GΩ resistor that is used to test the 100 nA current range.				

Calibration limits

The calibration limits stated in this section have been calculated using only the 2606B one-year accuracy specifications. They do not include test equipment uncertainty. If a particular measurement falls outside the allowable range, recalculate new limits based both on the 2606B specifications and corresponding test equipment specifications.

Source limit calculations

As an example of how verification limits are calculated, assume you are testing the 6 V dc output range using a 5.4 V output value. Using the one-year accuracy specification for 5.4 V dc output of ± (0.02% of output + 1.8 mV offset), the calculated output limits are:

$$\text{Output limits} = 5.4 \text{ V} \pm [(5.4 \text{ V} \times 0.02\%) + 1.8 \text{ mV}]$$

$$\text{Output limits} = 5.4 \text{ V} \pm (0.00108 \text{ V} + 0.0018 \text{ V})$$

$$\text{Output limits} = 5.4 \text{ V} \pm 0.00288 \text{ V}$$

$$\text{Output limits} = 5.39712 \text{ V to } 5.40288 \text{ V}$$

Measurement limit calculations

Measurement limits are calculated in the same way as the source limits, except that the limits are calculated with respect to the measurement of the external reference instrument.

Restoring factory defaults

Before performing the calibration procedures, restore the instrument to its factory defaults.

To restore the factory defaults:

1. Press the **MENU** key.
2. Scroll to the **SETUP** menu item, and then press the **ENTER** key.
3. Scroll to the **RECALL** menu item, and then press the **ENTER** key.
4. Scroll to the **INTERNAL** menu item, and then press the **ENTER** key.
5. Scroll to the **FACTORY** menu item.
6. Press the **ENTER** key to restore defaults.

Performing the calibration test procedures

Perform the following calibration tests to make sure the instrument is operating within specifications:

- [Current source accuracy](#) (on page 17-6)
- [Current measurement accuracy](#) (on page 17-8)
- [Voltage source accuracy](#) (on page 17-9)
- [Voltage measurement accuracy](#) (on page 17-11)

If the 2606B is not within specifications and not under warranty, see the procedures in [Adjustment](#) (on page 17-12) for information on adjusting the instrument.

Test considerations

When performing the calibration procedures:

- Be sure to restore factory front panel defaults as described in [Restoring factory defaults](#) (on page 17-4).
- Make sure that the test equipment is properly warmed up and connected to the 2606B output terminals. Use 4-wire sensing for voltage.
- Make sure the 2606B SMU is set to the correct source range.
- Be sure the 2606B SMU output is turned on before making measurements.
- Be sure the test equipment is set up for the proper function and range.
- Allow the 2606B SMU output signal to settle before making a measurement.
- Do not connect test equipment to the 2606B through a scanner, multiplexer, or other switching equipment.

WARNING

The maximum common-mode voltage (voltage between LO and chassis ground) is 250 V dc. Exceeding this value may cause a breakdown in insulation, creating a shock hazard that could result in personal injury or death.

WARNING

The input/output terminals of the 2606B 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 2606B terminals to CAT II, CAT III, or CAT IV circuits. Connection of the 2606B terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

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 2606B 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 2606B before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Setting the source range and output value

Before testing each calibration point, you must properly set the source range and output value.

To set the source range and output value:

1. Press the **SRC** key to select the appropriate source function.
2. Press the navigation wheel to enable the edit mode (EDIT indicator on).
3. When the cursor in the source display field is flashing, set the source range to the range being calibrated. Use the up or down **RANGE** keys to select the range.
4. Use the navigation wheel and **CURSOR** keys to set the source value to the required value.
5. Press the navigation wheel to complete editing.

Setting the measurement range

When simultaneously sourcing and measuring either voltage or current, the measure range is coupled to the source range, and you cannot independently control the measure range. Thus, it is not necessary for you to set the range when testing voltage or [current measurement accuracy](#) (on page 17-8).

Current source accuracy

To verify that the 2606B output current accuracy is within specified limits:

1. With the power off, connect the digital multimeter to the 2606B terminals as shown in the figure below.

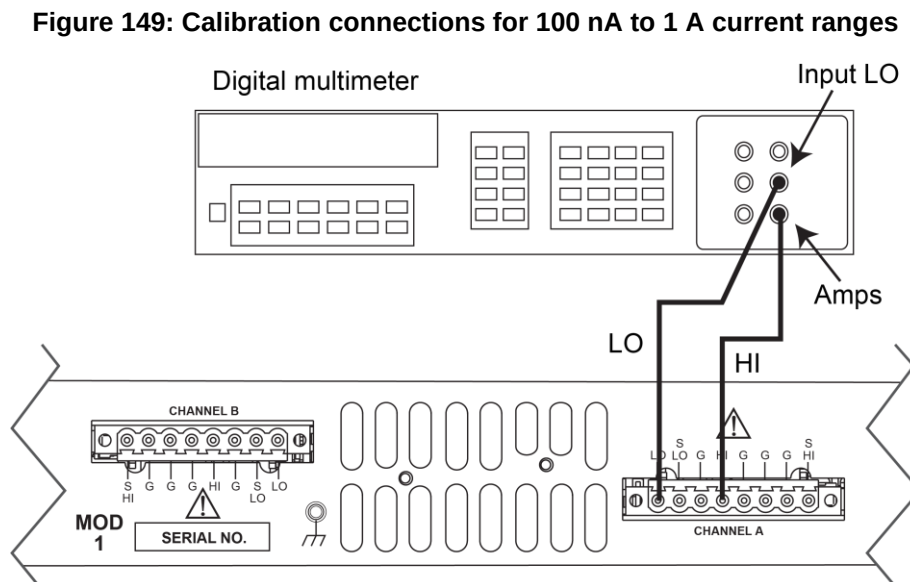
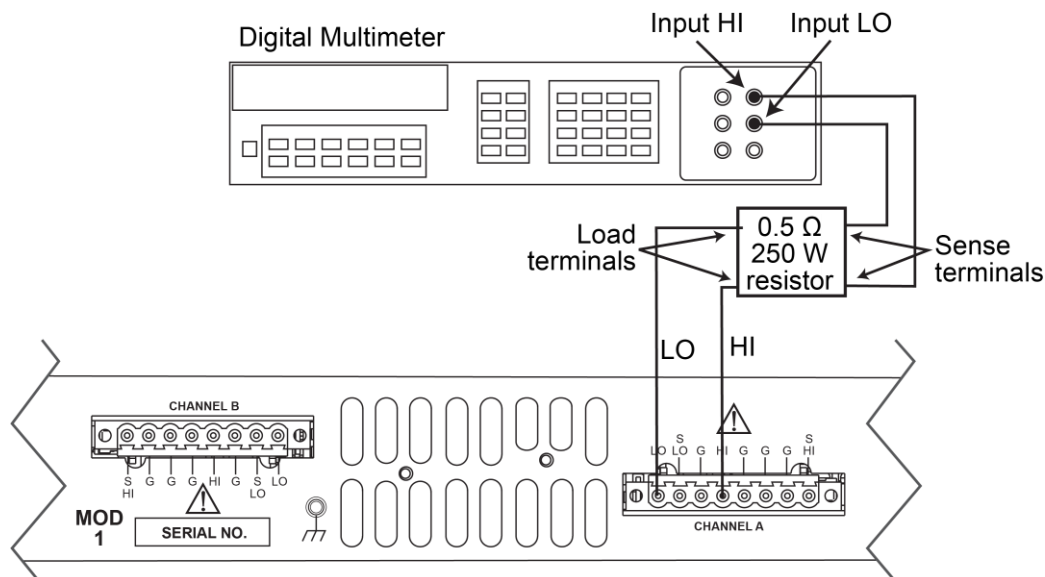


Figure 150: Calibration connections for 3 A current range

2. Select the multimeter dc current measuring function.
3. Select the single-channel display mode.
4. Press the **SRC** key to source current and make sure the source output is turned on.
5. Verify output current accuracy for each of the currents for the 100 nA to 1 A ranges using the values listed in the following table. For each test point:
 - Select the correct source range.
 - Set the 2606B output current to the correct value.
 - Verify that the multimeter reading is within the limits given in the table below.

Output current accuracy limits

Source range	Output current setting	Output current limits (1 year, 18 °C to 28 °C)
100 nA	90.000 nA	89.846 nA to 90.154 nA
1 μA	0.90000 μA	0.89893 μA to 0.90107 μA
10 μA	9.0000 μA	8.9923 μA to 9.0077 μA
100 μA	90.000 μA	89.913 μA to 90.087 μA
1 mA	0.90000 mA	0.89943 mA to 0.90057 mA
10 mA	9.0000 mA	8.9913 mA to 9.0087 mA
100 mA	90.000 mA	89.943 mA to 90.057 mA
1 A	0.90000 A	0.89775 A to 0.90225 A
3 A	2.40000 A	2.39456 A to 2.40544 A

6. Repeat the procedure for negative output currents with the same magnitudes as those listed.
7. Turn the output off, and change connections as shown in the figure titled "Connections for 3 A current range," above.
8. Select the DMM dc volts function.
9. Repeat steps 4 through 6 for the 3 A range. Calculate the current from the DMM voltage reading and the characterized $0.5\ \Omega$ resistance value: $I=V/R$.
10. Repeat the above procedure for the other channel.

Current measurement accuracy

The procedure involves applying accurate currents from the 2606B current source and then verifying that 2606B current measurements are within required limits.

To verify that 2606B current measurement accuracy is within specified limits:

1. With the power off, connect the digital multimeter to the 2606B terminals as shown in the figure titled "Calibration connections for 100 nA to 1 A current ranges" in [Current source accuracy](#) (on page 17-6).
2. Select the multimeter dc current function.
3. Select the single-channel display mode.
4. Set the 2606B SMU to both source and measure current by pressing the **SRC** and then the **MEAS** keys. Make sure the source output is turned on.
5. Verify measure current accuracy for each of the currents listed using the values listed in the following table for your model number. For each measurement:
 - Select the correct source range.
 - Set the 2606B output current so that the digital multimeter reading is the value indicated in the source current column of the table below. It may not be possible to set the current source to get exactly the required reading on the digital multimeter. Use the closest possible setting and modify the reading limits accordingly.
 - If necessary, press the **TRIG** key to display readings.
 - Verify that the 2606B current reading is within the limits given in the table below.

- Repeat the procedure for negative calibrator currents with the same magnitudes as those listed.

Current measurement accuracy limits

Source and measure range ¹	Source current ²	Current reading limits (1 year, 18 °C to 28 °C)
100 nA	90.000 nA	89.855 nA to 90.145 nA
1 µA	0.9000 µA	0.89928 µA to 0.90073 µA
10 µA	9.0000 µA	8.9963 µA to 9.0038 µA
100 µA	90.000 µA	89.957 µA to 90.043 µA
1 mA	0.9000 mA	0.89962 mA to 0.90038 mA
10 mA	9.0000 mA	8.9957 mA to 9.0043 mA
100 mA	90.000 mA	89.962 mA to 90.038 mA
1 A	0.90000 A	0.89823 A to 0.90177 A
3 A	2.4000 A	2.3953 A to 2.4047 A

¹ Measure range coupled to source range when simultaneously sourcing and measuring current.

² As measured by precision digital multimeter. Use closest possible value and modify reading limits accordingly if necessary. See [Measurement limit calculations](#) (on page 17-4).

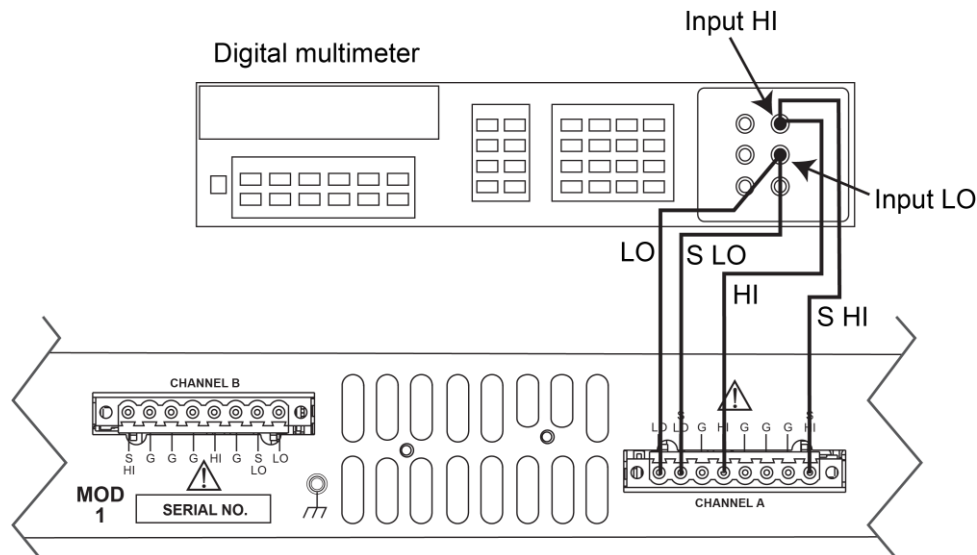
- Turn the output off and change connections as shown adding the 0.5 Ω 250 W resistor (see the figure titled “Connections for 3 A current range” in [Current source accuracy](#) (on page 17-6)).
- Select the DMM volts function.
Repeat steps 4 through 6 for the 3 A range. Calculate the current from the DMM voltage reading and characterized 0.5 Ω resistance value.
- Repeat the above procedure for the other channel.

Voltage source accuracy

To perform this test, you will set the output voltage to each full-range value and measure the voltages with a precision digital multimeter.

To verify that the 2606B output voltage accuracy is within specified limits:

- With the power off, connect the digital multimeter (DMM) to the 2606B output terminals using 4-wire connections, as shown below.

Figure 151: Connections for voltage verification and adjustment

2. Set the multimeter measuring function to dc volts.
3. Select the single-channel display mode.
4. Press the **SRC** key to source voltage and make sure the source output is turned on.
5. Enable the 2606B 4-wire (remote sense) mode:
 - a. Press the **CONFIG** key and then the **SRC** key.
 - b. Select **V-SOURCE > SENSE-MODE > 4-WIRE**.
6. Verify output voltage accuracy for each of the voltages listed in the following table. For each test point:
 - Select the correct source range.
 - Set the 2606B output voltage to the indicated value.
 - Verify that the multimeter reading is within the limits given in the table.

Output voltage accuracy limits

Source range	Output voltage setting	Output voltage limits (1 year, 18 °C to 28 °C)
100 mV	90.000 mV	89.732 mV to 90.268 mV
1 V	0.90000 V	0.89942 V to 0.90058 V
6 V	5.4000 V	5.39712 V to 5.40288 V
20 V	18.000 V	17.9844 V to 18.0156 V

7. Repeat the procedure for negative output voltages with the same magnitudes as those listed in the previous table, as applicable.
8. Repeat the procedure for the other channel.

Voltage measurement accuracy

To perform this test, you set the source voltage, as measured by a precision digital multimeter, and then verify that the 2606B voltage readings are within required limits.

To verify that the 2606B voltage measurement accuracy is within specified limits:

1. With the power off, connect the digital multimeter to the 2606B output terminals using 4-wire connections (use the same connections as in the figure titled "Connections for voltage verification" in [Voltage source accuracy](#) (on page 17-9)).
 2. Select the multimeter dc volts function.
 3. Select the single-channel display mode.
 4. Enable the 2606B 4-wire (remote sense) mode:
 - a. Press the **CONFIG** key and then the **MEAS** key.
 - b. Select **V-MEAS > SENSE-MODE > 4-WIRE**.
 5. Set the 2606B SMU to both source and measure voltage by pressing the **SRC** and then the **MEAS** keys.
 6. Make sure the source output is turned on (if off, press the **OUTPUT ON/OFF** control).
 7. Verify voltage measurement accuracy for each of the voltages listed in the table (see below). For each test point:
 - Select the correct source range.
 - Set the 2606B output voltage such that the digital multimeter reading is the value indicated in the source voltage column of the table below. It may not be possible to set the voltage source to get exactly the required reading on the digital multimeter. Use the closest possible setting and modify the reading limits accordingly.
 - Verify that the 2606B voltage reading is within the limits given in the table.
- Repeat the procedure for negative source voltages with the same magnitudes as those listed in the table (see below).
8. Repeat the above procedure for the other channel.

Voltage measurement accuracy limits

Source and measure range ¹	Source voltage ²	Voltage reading limits (1 year, 18 °C to 28 °C)
100 mV	90.000 mV	89.8365 V to 90.1635 mV
1 V	0.90000 V	0.899665 V to 0.900335 V
6 V	5.4000 V	5.39819 V to 5.40181 V
20 V	18.000 V	17.9893 V to 18.0107 V

¹ Measure range coupled to source range when simultaneously sourcing and measuring voltage.
² As measured by precision digital multimeter. Use closest possible value and modify reading limits if necessary.

Adjustment

WARNING

The information in this topic is intended only for qualified service personnel. Some of the procedures may expose you to hazardous voltages that could result in personal injury or death. Do not attempt to perform these procedures unless you are qualified to do so.

Use the procedures in this section to calibrate the 2606B.

These procedures require accurate test equipment to measure precise dc voltages and currents.

The 2606B contains two independent SMU modules. Each module requires separate adjustment.

The instructions in this section use the web interface (refer to [Virtual Front Panel](#) (on page 3-24)). You can also use remote programming commands. Refer to [Instrument programming](#) (on page 14-1) for information on using commands to control the instrument.

NOTE

Product specifications are subject to change. Listed uncertainties and test limits are provided only as an example. Always verify values against actual product specifications.

Environmental conditions

Temperature and relative humidity

Conduct the adjustment procedures at an ambient temperature of 18 °C to 28 °C, with relative humidity of less than 70 percent (unless otherwise noted).

NOTE

Product specifications that are listed as 18 °C to 28 °C assume adjustment has been done at 23 °C. If the 2606B is adjusted at a different temperature, the specifications apply to ± 5 °C of that temperature.

Line power

The 2606B requires a line voltage of 100 V to 240 V at a line frequency of 50 Hz or 60 Hz. The instrument must be adjusted within this range.

Warmup period

Allow the 2606B to warm up for at least two hours before adjusting the instrument.

If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the internal temperature of the instrument to stabilize. Typically, allow one extra hour to stabilize an instrument that is 10 °C outside the specified temperature range.

Allow the test equipment to warm up for the minimum time specified by the manufacturer.

Adjustment considerations

When performing the adjustment procedures:

- Make sure that the test equipment is properly warmed up and connected to the correct 2606B terminals.
- Always allow the source signal to settle before adjusting each point.
- Do not connect test equipment to the 2606B through a scanner or other switching equipment.
- If an error occurs during adjustment, the 2606B generates an appropriate error message. See the [Error summary list](#) (on page 12-2) for more information.

WARNING

The maximum common-mode voltage (voltage between LO and chassis ground) is 250 V dc. Exceeding this value may cause a breakdown in insulation, creating a shock hazard that could result in personal injury or death.

WARNING

The input/output terminals of the 2606B 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 2606B terminals to CAT II, CAT III, or CAT IV circuits. Connection of the 2606B terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

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 2606B 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 2606B before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Adjustment cycle

Perform an adjustment at least once a year to make sure the instrument meets or exceeds its specifications.

Recommended adjustment equipment

The table below contains the recommended equipment for the calibration adjustment procedures. You can use alternate equipment as long as that equipment has specifications equal to or greater than those listed in the table. When possible, test equipment specifications should be at least four times better than corresponding 2606B specifications.

Recommended adjustment equipment

Description	Manufacturer/Model	Accuracy		
Digital Multimeter	Keithley Instruments Model 2002 or Agilent 3458A	DC voltage ¹	90 mV: 0.9 V: 5.4 V: 18 V:	±8 ppm ±5 ppm ±4 ppm ±6 ppm
		DC current ²	90 nA: 0.9 A: 9 A: 90 A: 0.9 mA: 9 mA: 90 mA: 0.9 A:	±430 ppm ±45 ppm ±25 ppm ±23 ppm ±20 ppm ±20 ppm ±35 ppm ±110 ppm
0.5 Ω, 250 W, 0.1% Precision Resistor	Isotek RUG-Z-R500-0.1-TK3	Resistance ³	0.5 Ω:	±125 ppm
50 Ω Resistors (2)	Any suitable ⁴			
1 GΩ, 200 V, 1% standard	Keithley Instruments Model 2600-STD-RES	Resistance ⁵	1 GΩ:	±250 ppm
¹ 90-day specifications show full-range accuracy of recommended model used for specified calibration point.				
² Id.				
³ Resistor used to calibrate 3 A ranges should be characterized to uncertainty shown using resistance function of a digital multimeter before use.				
⁴ Used for contact check calibration. Characterize resistors using ohms function of digital multimeter before use.				
⁵ Standard is a guarded and characterized 1 GΩ resistor that is used to test the 100 nA current range.				

Calibration adjustment overview

The following topics contain an overview of the entire calibration adjustment procedure.

Parameter values

The full-scale parameters are 90 percent of full-scale as indicated (see the table in [Step sequence](#) (on page 17-15)). Note that you cannot send a value of 0 for the two zero parameters. Instead, you must send a very small value, such as 1e-30 or -1e-30.

Sense modes

The preceding table for your specific model lists the sense modes for the calibration steps. Note that each source and measure range is calibrated using the LOCAL sense mode. In addition, the 100 mV source and measure range is also calibrated using the REMOTE sense mode, and the 1 V and 1 mA source ranges are also calibrated using the CALA sense mode.

Step sequence

Adjustment steps must be performed in a specific sequence. See the following table that is specific table to your model. Note that all steps are performed using 2-wire (local sensing) except as noted. Adjustment of each range is performed as a four-point calibration:

- + ZERO
- + FULL SCALE
- - ZERO
- - FULL SCALE

NOTE

Before performing the adjustment steps, refer to [Parameter values](#) (on page 17-15) and [Sense modes](#) (on page 17-15).

Adjustment steps

Function ¹	Adjustment steps ²	Calibration points ³	Sense mode ⁴
Voltage Source and Measure	100 mV	$\pm 1\text{e-}30, \pm 90 \text{ mV}$	smuX.SENSE_LOCAL
	100 mV	$\pm 1\text{e-}30, \pm 90 \text{ mV}$	smuX.SENSE_REMOTE
	1 V	$\pm 1\text{e-}30, \pm 0.9 \text{ V}$	smuX.SENSE_LOCAL
	1 V	$\pm 1\text{e-}30, \pm 0.9 \text{ V}$	smuX.SENSE_CALA
	6 V	$\pm 1\text{e-}30, \pm 5.4 \text{ V}$	smuX.SENSE_LOCAL
	20 V	$\pm 1\text{e-}30, \pm 18 \text{ V}$	smuX.SENSE_LOCAL
Current Source and Measure	100 nA	$\pm 1\text{e-}30, \pm 90 \text{ nA}$	smuX.SENSE_LOCAL
	1 μA	$\pm 1\text{e-}30, \pm 0.9 \mu\text{A}$	smuX.SENSE_LOCAL
	10 μA	$\pm 1\text{e-}30, \pm 9 \mu\text{A}$	smuX.SENSE_LOCAL
	100 μA	$\pm 1\text{e-}30, \pm 90 \mu\text{A}$	smuX.SENSE_LOCAL
	1 mA	$\pm 1\text{e-}30, \pm 0.9 \text{ mA}$	smuX.SENSE_LOCAL
	1 mA	$\pm 1\text{e-}30, \pm 0.9 \text{ mA}$	smuX.SENSE_CALA
	10 mA	$\pm 1\text{e-}30, \pm 9 \text{ mA}$	smuX.SENSE_LOCAL
	100 mA	$\pm 1\text{e-}30, \pm 90 \text{ mA}$	smuX.SENSE_LOCAL
	1 A	$\pm 1\text{e-}30, \pm 0.9 \text{ A}$	smuX.SENSE_LOCAL
	3 A	$\pm 1\text{e-}30, \pm 2.4 \text{ A}$	smuX.SENSE_LOCAL

¹ Calibrate only the source for the CALA sense steps.

² Steps must be performed in the order shown.

³ Do not use actual 0 values for zero calibration points. Send very small values such as $\pm 1\text{e-}30$. Calibration polarities must also be set as shown in the procedures.

⁴ Output must be off before changing to the CALA sense mode.

Calibration commands quick reference

The following table summarizes remote calibration commands. For a more complete description of these commands, refer to the [TSP command reference](#) (on page 15-1).

Calibration commands	
Command	Description
<code>smuX.cal.adjustdate = <i>adjustDate</i></code>	Set date when the adjustment was done.
<code>smuX.cal.date = <i>calDate</i></code>	Set calibration date (<i>calDate</i> of 0 indicates date not set).
<code>smuX.cal.due = <i>calDue</i></code>	Set date when calibration should be performed (<i>calDue</i> of 0 indicates date not set).
<code>smuX.cal.password = "<i>newPassword</i>"</code>	Change password to <i>newPassword</i> .
<code>smuX.cal.polarity = <i>calPolarity</i></code>	Set polarity: smuX.CAL_AUTO (automatic polarity). smuX.CAL_NEGATIVE (negative polarity). smuX.CAL_POSITIVE (positive polarity).
<code>smuX.cal.restore(<i>calset</i>)</code>	Load set of calibration constants: smuX.CALSET_NOMINAL (nominal constants). smuX.CALSET_FACTORY (factory constants). smuX.CALSET_DEFAULT (normal constants). smuX.CALSET_PREVIOUS (previous constants).
<code>smuX.cal.save()</code>	Store constants in nonvolatile memory as DEFAULT calibration set.

Calibration commands	
Command	Description
<code>calstate = smuX.cal.state</code>	Request calibration state: smuX.CALSTATE_CALIBRATING smuX.CALSTATE_LOCKED smuX.CALSTATE_UNLOCKED
<code>smuX.cal.unlock("password")</code>	Unlock calibration (default password: KI0026XX)
<code>smuX.measure.calibratei(range, cp1Measured, cp1Reference, cp2Measured, cp2Reference)</code>	Adjust current measurement range calibration*: $\pm range$ (measurement range to adjust). <i>cp1Measured</i> (2606B measured value for calibration point 1). <i>cp1Reference</i> (reference measurement for calibration point 1). <i>cp2Measured</i> (2606B measured value for calibration point 2). <i>cp2Reference</i> (reference measurement for calibration point 2).
<code>smuX.measure.calibratev(range, cp1Measured, cp1Reference, cp2Measured, cp2Reference)</code>	Adjust voltage measurement range calibration*: $\pm range$ (measurement range to adjust). <i>cp1Measured</i> (2606B measured value for calibration point 1). <i>cp1Reference</i> (reference measurement for calibration point 1). <i>cp2Measured</i> (2606B measured value for calibration point 2). <i>cp2Reference</i> (reference measurement for calibration point 2).
<code>smuX.source.calibratei(range, cp1Expected, cp1Reference, cp2Expected, cp2Reference)</code>	Adjust current source range calibration*: $\pm range$ (source range to adjust). <i>cp1Expected</i> (source value programmed for calibration point 1). <i>cp1Reference</i> (reference measurement for calibration point 1). <i>cp2Expected</i> (source value programmed for calibration point 2). <i>cp2Reference</i> (reference measurement for calibration point 2).
<code>smuX.source.calibratev(range, cp1Expected, cp1Reference, cp2Expected, cp2Reference)</code>	Adjust voltage source range calibration*: $\pm range$ (source range to adjust). <i>cp1Expected</i> (source value programmed for calibration point 1). <i>cp1Reference</i> (reference measurement for calibration point 1). <i>cp2Expected</i> (source value programmed for calibration point 2). <i>cp2Reference</i> (reference measurement for calibration point 2).

Calibration commands	
Command	Description
<code>smuX.contact.calibratelo(<i>cp1Measured</i>, <i>cp1Reference</i>, <i>cp2Measured</i>, <i>cp2Reference</i>)</code>	Adjust the low/sense low contact check measurement calibration. <i>cp1Measured</i> (value measured by SMU for calibration point 1). <i>cp1Reference</i> (reference measurement for calibration point 1). <i>cp2Measured</i> (value measured by SMU for calibration point 2). <i>cp2Reference</i> (reference measurement for calibration point 2).
<code>smuX.contact.calibratehi(<i>cp1Measured</i>, <i>cp1Reference</i>, <i>cp2Measured</i>, <i>cp2Reference</i>)</code>	Adjust the high/sense high contact check measurement calibration. <i>cp1Measured</i> (value measured by SMU for calibration point 1). <i>cp1Reference</i> (reference measurement for calibration point 1). <i>cp2Measured</i> (value measured by SMU for calibration point 2). <i>cp2Reference</i> (reference measurement for calibration point 2).

* Calibration point 1 should be performed at approximately 0% of range; calibration point 2 should be performed at approximately 90% of range. See [Step sequence](#) (on page 17-15) for calibration points.

Adjustment procedure

Use the following procedure to perform remote calibration adjustment by sending commands over a communications interface. The remote commands and appropriate parameters are separately summarized for each step.

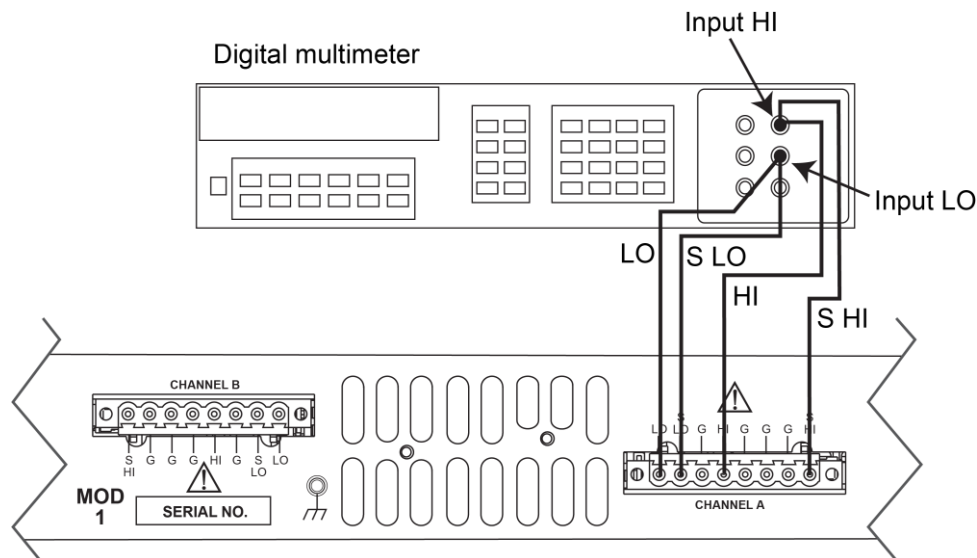
Prepare the 2606B for calibration adjustment

1. Connect the 2606B to the controller using an appropriate interface cable.
2. Turn on the 2606B and the test equipment and allow them to warm up for at least two hours before performing calibration adjustment.
3. Make sure the communication interface parameters are set up properly (to configure the interface, press the **MENU** key, and then select **LAN**; configuration of the USB interface is not necessary).

Voltage calibration adjustment

1. Connect the 2606B SMU to the digital multimeter using the 4-wire connections shown in the figure below.

Figure 152: Connections for voltage verification and adjustment



2. Select the multimeter dc volts function.
3. To initialize voltage calibration, send the following commands:

```
smua.cal.unlock("KI0026XX")
smua.reset()
smua.source.func = smua.OUTPUT_DCVOLTS
```

NOTE

You do not need to set the measure range when following this procedure for calibration because the measure range is locked to the source range when measuring the source function.

Perform each calibration adjustment for the voltage source and measure function step listed in [Step sequence](#) (on page 17-15) as follows. Be sure to complete each of the steps for all six voltage steps in [Step sequence](#) (on page 17-15) before performing current calibration.

1. Select the range being calibrated with this command:

```
smua.source.rangev = range
```

2. Select the correct sense mode based on the calibration step for the voltage source and measure function from the [Step sequence](#) (on page 17-15), for example:

```
smua.sense = smua.SENSE_LOCAL
```

3. Select positive polarity, and then set the source output to the positive zero value. For example:

```
smua.cal.polarity = smua.CAL_POSITIVE
smua.source.levelv = 1e-30
```

4. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

5. Allow the readings to settle, then get both the multimeter and 2606B voltage readings at the positive zero value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
Z_rdg = smua.measure.v()
```

6. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

7. Set the source output to the positive full-scale value for the present range, for example:

```
smua.source.levelv = 0.9
```

8. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

9. Allow the readings to settle, then get both the multimeter and 2606B voltage readings at the positive full-scale output value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
FS_rdg = smua.measure.v()
```

10. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

11. Send the source calibration command using the range, +zero and +FS multimeter readings, and +zero and +FS source values for the parameters:

```
smua.source.calibratev(range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
```

Where:

<i>range</i>	= The present calibration range
<i>src_Z</i>	= The +zero 2606B programmed source output value
<i>DMM_Z_rdg</i>	= The +zero DMM measurement
<i>src_FS</i>	= The +FS 2606B programmed source output value
<i>DMM_FS_rdg</i>	= The +FS DMM measurement

Typical values for the 1 V range:

```
smua.source.calibratev(1, 1e-30, 1e-5, 0.9, 0.903)
```

12. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and 2606B readings, and the range setting for the parameters. For example:

```
smua.measure.calibratev(range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
```

Where:

range = The present calibration range
Z_rdg = The +zero 2606B measurement
DMM_Z_rdg = The +zero DMM measurement
FS_rdg = The +FS 2606B measurement
DMM_FS_rdg = The +FS DMM measurement

Typical 1 V range values:

```
smua.measure.calibratev(1, 1e-4, 1e-5, 0.92, 0.903)
```

13. Select negative polarity, then set the source output to the negative zero value, for example:

```
smua.cal.polarity = smua.CAL_NEGATIVE  
smua.source.levelv = -1e-30
```

14. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

15. Allow the readings to settle, then get both the multimeter and 2606B voltage readings at the negative zero value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
Z_rdg = smua.measure.v()
```

16. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

17. Set the source output to the negative full-scale value, for example:

```
smua.source.levelv = -0.9
```

18. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

19. Allow the readings to settle, then get both the multimeter and 2606B voltage readings at the negative full-scale output value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
FS_rdg = smua.measure.v()
```

20. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

21. Send the source calibration command using the range, -zero and -FS multimeter readings, and -zero and -FS source values for the parameters:

```
smua.source.calibratev(-range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
```

Where:

-range = The negative of the present calibration range
src_Z = The -zero 2606B programmed source output value
DMM_Z_rdg = The -zero DMM measurement
src_FS = The -FS 2606B programmed source output value
DMM_FS_rdg = The -FS DMM measurement

Typical values for the 1 V range:

```
smua.source.calibratev(-1, -1e-30, -1e-4, -0.9, -0.896)
```

22. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and 2606B readings and range setting for the parameters:

```
smua.measure.calibratev(-range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
```

Where:

-range = The negative of the present calibration range
Z_rdg = The -zero 2606B measurement
DMM_Z_rdg = The -zero DMM measurement
FS_rdg = The -FS 2606B measurement
DMM_FS_rdg = The -FS DMM measurement

Typical 1 V range values:

```
smua.measure.calibratev(-1, -1e-4, -1e-6, -0.89, -0.896)
```

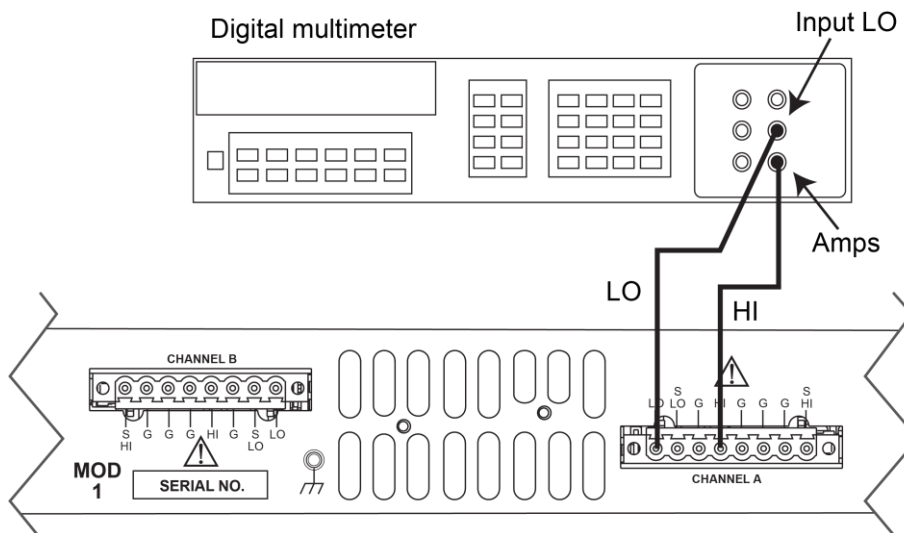
23. Select automatic polarity mode:

```
smua.cal.polarity = smua.CAL_AUTO
```

Current calibration adjustment

1. Connect the 2606B SMU to the digital multimeter (see the following figure), and then select the multimeter dc current function.

Figure 153: Calibration connections for 100 nA to 1 A current ranges



2. Send this command to initialize current calibration:

```
smua.source.func = smua.OUTPUT_DCAMPS
```

NOTE

It is not necessary to set the measure range when following this procedure for calibration because the measure range is locked to the source range when measuring the source function.

Be sure to complete the steps for the 100 nA to 1 A ranges before continuing with 3 A calibration.

Perform each calibration step listed in step sequence for the 100 nA through 1 A ranges:

1. Select the range being calibrated:

```
smua.source.rangei = range
```

2. Select the correct sense mode:

```
smua.sense = smua.SENSE_LOCAL
```

3. Select positive polarity, then set the source output to the positive zero value:

```
smua.cal.polarity = smua.CAL_POSITIVE
```

```
smua.source.level1 = 1e-30
```

4. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

5. Allow the readings to settle, then get both the multimeter and 2606B current readings at the positive zero value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
Z_rdg = smua.measure.i()
```

6. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

7. Set the source output to the positive full-scale value for the present range, for example:

```
smua.source.level1 = 90e-3
```

8. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

9. Allow the readings to settle, then get both the multimeter and 2606B current readings at the positive full-scale output value (the 2606B measurement is not necessary if calibration is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
FS_rdg = smua.measure.i()
```

10. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

11. Send the source calibration command using the range, zero and +FS multimeter readings, and zero and +FS source values for the parameters:

```
smua.source.calibratei(range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
```

Where:

<i>range</i>	= The present calibration range
<i>src_Z</i>	= The +zero 2606B source output value
<i>DMM_Z_rdg</i>	= The +zero DMM measurement
<i>src_FS</i>	= The +FS 2606B source output value
<i>DMM_FS_rdg</i>	= The +FS DMM measurement

Typical values for the 100 mA range:

```
smua.source.calibratei(100e-3, 1e-30, 1e-5, 90e-3, 88e-3)
```

12. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and 2606B readings, and range setting for the parameters:

```
smua.measure.calibratei(range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
```

Where:

range = The present calibration range
Z_rdg = +zero 2606B measurement
DMM_Z_rdg = The +zero DMM measurement
FS_rdg = +FS 2606B measurement
DMM_FS_rdg = The +FS DMM measurement

Typical 100 mA range values:

```
smua.measure.calibratei(100e-3, 1e-6, 1e-5, 0.089, 0.088)
```

13. Select negative polarity, then set the source output to the negative zero value, for example:

```
smua.cal.polarity = smua.CAL_NEGATIVE  
smua.source.level1 = -1e-30
```

14. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

15. Allow the readings to settle, then get both the multimeter and 2606B current readings at the negative zero value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
Z_rdg = smua.measure.i()
```

16. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

17. Set the source output to the negative full-scale value, for example:

```
smua.source.level1 = -90e-3
```

18. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

19. Allow the readings to settle, then get both the multimeter and 2606B current readings at the negative full-scale output value (the 2606B measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the 2606B:

```
FS_rdg = smua.measure.i()
```

20. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

21. Send the source calibration command using the -range, -zero and -FS multimeter readings, and -zero and -FS source values for the parameters:

```
smua.source.calibratei(-range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
```

Where:

-range = The negative of the present calibration range
 src_Z = The zero 2606B source output value
 DMM_Z_rdg = The zero DMM measurement
 src_FS = The FS 2606B source output value
 DMM_FS_rdg = The FS DMM measurement

Typical values for the 100 mA range:

```
smua.source.calibratei(-100e-3, -1e-30, -1e-6, -90e-3, -89.2e-3)
```

22. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and 2606B readings, and range setting for the parameters:

```
smua.measure.calibratei(-range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
```

Where:

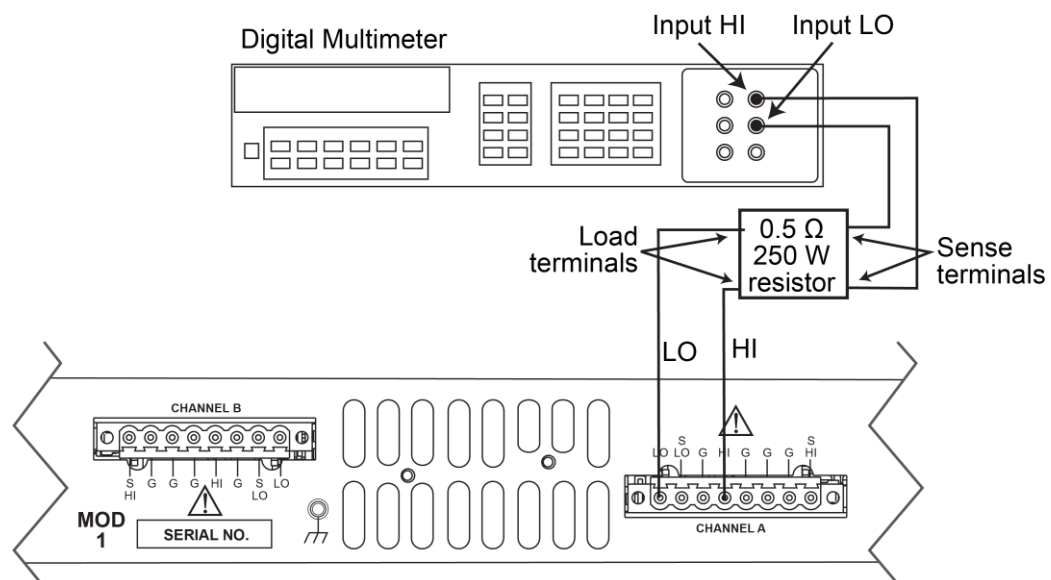
-range = The negative of the present calibration range
 Z_rdg = The zero 2606B measurement
 DMM_Z_rdg = The zero DMM measurement
 FS_rdg = The FS 2606B measurement
 DMM_FS_rdg = The FS DMM measurement

Typical 100 mA range values:

```
smua.measure.calibratei(-100e-3, -1e-5, -1e-6, -91e-3, -89.2e-3)
```

23. Change connections as shown in the following figure.

Figure 154: Calibration connections for 3 A current range



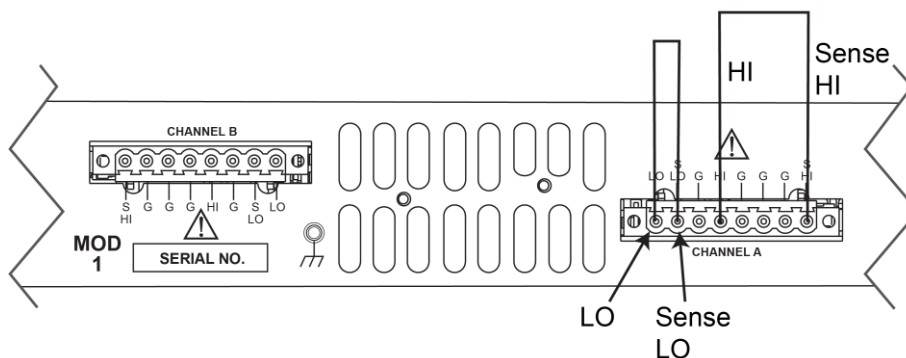
24. Select the DMM dc volts function.
25. Repeat the above steps for the 3 A range. Compute the current reading from the DMM voltage reading and characterized $0.5\ \Omega$ resistance value: $I = V/R$.
26. Select automatic polarity mode:

```
smua.cal.polarity = smua.CAL_AUTO
```

Contact check calibration adjustment

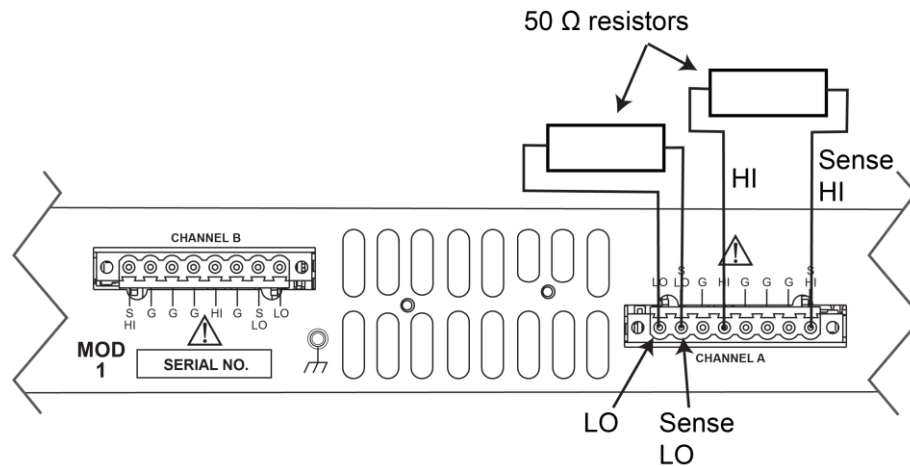
1. As illustrated in the following figure:
 - Short the 2606B SENSE LO and LO terminals together.
 - Short the SENSE HI and HI terminals together.

Figure 155: Connections for contact check 0 ohm calibration



2. Allow the readings to settle, then get the 2606B readings:


```
r0_hi, r0_lo = smua.contact.r()
```
3. Characterize both $50\ \Omega$ resistors using the resistance function of the digital multimeter.
4. As illustrated in the following figure:
 - Characterize both $50\ \Omega$ resistors using the resistance function of the digital multimeter.
 - Connect a $50\ \Omega$ resistor between the SENSE LO and LO terminals.
 - Connect the second $50\ \Omega$ resistor between the SENSE HI and HI terminals.

Figure 156: Connections for contact check 50 ohm calibration

5. Allow the readings to settle, then get the 2606B readings:

```
r50_hi, r50_lo = smua.contact.r()
```

6. Send the contact check low calibration adjustment command:

```
smua.contact.calibratelo(r0_lo, Z_actual, r50_lo, 50_ohm_actual)
```

Where:

r0_lo = 2606B 0 Ω low measurement

Z_actual = Actual zero value; the resistance of the short between the SENSE LO and LO terminals

r50_lo = 2606B 50 Ω low measurement

50_ohm_actual = Actual 50 Ω resistor value; the actual value of the resistor between the SENSE LO and LO terminals

Typical values:

```
smua.contact.calibratelo(r0_lo, 0, r50_lo, 50.15)
```

Where *r0_lo* is the same value as measured in step B, and *r50_lo* is the same value as measured in step 5.

7. Send the contact check high calibration command:

```
smua.contact.calibratehi(r0_hi, Z_actual, r50_hi, 50_ohm_actual)
```

Where:

r0_hi = 2606B 0 Ω high measurement

Z_actual = Actual zero value; the resistance of the short between the SENSE HI and HI terminals

r50_hi = 2606B 50 Ω high measurement

50_ohm_actual = Actual 50 Ω resistor value; the value of the resistor between the SENSE HI and HI terminals

Typical values:

```
smua.contact.calibratehi(r0_hi, 0, r50_hi, 50.15)
```

Where *r0_hi* is the same value as measured in step B, and *r50_hi* is the same value as measured in step 5.

Program calibration dates

Use the following command to set the calibration adjustment date:

```
smua.cal.adjustdate = os.time{year=2017, month=12, day=1}
```

Optionally, it is possible to set the calibration date and calibration due date with the following commands:

```
smua.cal.date = os.time{year=2017, month=12, day=1}
```

```
smua.cal.due = os.time{year=2018, month=12, day=1}
```

If you do not wish to set a calibration date or calibration due date and want to clear the previous values, use the following commands:

```
smua.cal.date = 0
```

```
smua.cal.due = 0
```

The actual year, month, day, and (optional) hour and minute should be used (seconds can be given but are essentially ignored due to the precision of the internal date storage format). The allowable range for the year is from 1970 to 2037, the month is from 1 to 12, and the day is from 1 to 31.

Save constants and lock out calibration

Calibration adjustment is now complete, so you can store the calibration constants in nonvolatile memory by sending the following command:

```
smua.cal.save()
```

NOTE

Unless you send the save command, the calibration adjustment you just performed are temporary.

To lock out further calibration, send the following command after completing the calibration procedure:

```
smua.cal.lock()
```

Repeat calibration procedure for Channel B

Repeat the entire procedure above for Channel B. Be sure to:

- Make test connections to Channel B terminals.
- Substitute `smub` for `smua` in all commands.

Frequently asked questions

In this section:

How do I find the serial number?.....	18-1
How do I optimize performance?.....	18-2
How do I upgrade the firmware?	18-2
How do I use the digital I/O port?	18-3
How do I trigger other instruments?	18-3
How do I store measurements in nonvolatile memory?.....	18-4
When should I change the output-off state?.....	18-5
How do I make contact check measurements?.....	18-5
How can I change the line frequency or voltage?.....	18-5
Where can I get the LabVIEW driver?	18-6
Why is the reading value 9.91e37?	18-6

How do I find the serial number?

Each 2606B has three serial numbers, one for the instrument and one for each module.

The instrument serial number is on a label on the front panel. This serial number is not available through a remote interface.

The serial numbers for the modules are on labels on the rear panel. The serial number for a specific module can be accessed through the virtual front panel or by using the [localnode.serialno](#) (on page 15-161) command over a remote interface.

To access the serial number from the virtual front panel:

1. If the 2606B is in remote operation, press the **EXIT (LOCAL)** key once to place the instrument in local operation.
2. Select the **MENU** key.
3. Use the navigation wheel to scroll to the SYSTEM-INFO menu item.
4. Press the **ENTER** key. The SYSTEM INFORMATION menu is displayed.
5. Scroll to the **SERIAL#** menu item.
6. Press the **ENTER** key. The serial number for the module is displayed.

How do I optimize performance?

The primary factors that affect measurement accuracy and speed are:

- **Warm-up:** For rated measurement accuracy, allow the 2606B to warm up for at least two hours before use.
- **Speed setting:** The speed setting affects both speed and accuracy. For more information, see [Speed](#) (on page 5-33).
- **Autozero:** Autozero can be disabled to increase speed at the expense of accuracy (for more information, see [Disabling autozero to increase speed](#) (on page 18-2)).

Disabling autozero to increase speed

Disabling autozero (setting it to OFF) can increase measurement speed. If autozero is disabled, accuracy drifts with time and temperature.

NOTE

Turning autozero OFF disables the autozero function and possibly increases measurement speed. To minimize drift, setting autozero to ONCE performs an autozero operation one time (when it is selected), and then disables the autozero function. For a more detailed discussion of autozero, see [Autozero](#) (on page 5-10).

To configure autozero from the virtual front panel:

1. Press the **CONFIG** key, and then select **MEAS** from the menu.
2. Select **AUTO-ZERO**, and then press the **ENTER** key or the navigation wheel.
3. Select the mode (**OFF**, **ONCE**, or **AUTO**), and then press the **ENTER** key or the navigation wheel.
4. Press the **EXIT (LOCAL)** key to the normal display.

Refer to [Remote command autozero](#) (on page 5-11) for details about configuring autozero from a remote interface.

How do I upgrade the firmware?

For information on upgrading the firmware, see [Upgrading the firmware](#) (on page 11-2).

How do I use the digital I/O port?

You can use the 2606B 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 I/O lines, send commands to the 2606B over a remote interface.

Use a cable equipped with a DB-25 plug (L-com part number CSMN25MF-5) to connect the digital I/O port to other Keithley Instruments models equipped with a Trigger Link (TLINK).

For more information about the 2606B digital I/O port, see [Digital I/O](#) (on page 7-37).

How do I trigger other instruments?

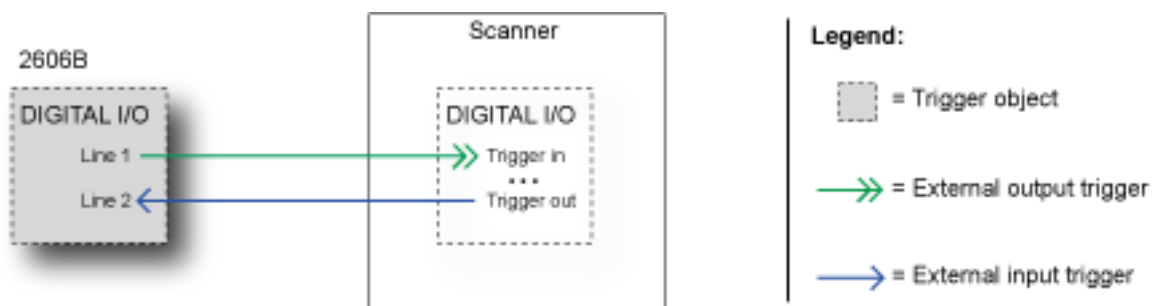
You can use the 2606B digital input/output to control an external digital circuit, such as a device handler used to perform binning operations. For more information about the 2606B digital I/O port, see [Digital I/O](#) (on page 7-37).

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 14-52) for additional information.

Triggering a scanner

A typical test scenario might call for using the 2606B with a scanner to test a number of devices under test (DUTs) in sequence. A basic example of this uses the 2606B digital I/O port to trigger a scanner (shown in the figure below). In this example, line 1 of the digital I/O port is used as a trigger output and connected to the scanner mainframe trigger input, and line 2 of the digital I/O port is used as a trigger input.

Figure 157: Triggering a scanner



Interactive trigger programming

The programming example below illustrates how to set up interactive triggering. The example sets the output trigger pulse width on line 1, then programs both lines 1 and 2 for falling edge triggers. Digital I/O line 1 trigger asserts, and then line 2 waits for the input trigger up to the timeout period specified.

```
-- Set line 1 pulse width to 10 us.
digio.trigger[1].pulsewidth = 10e-6
-- Set line 1 mode to falling edge.
digio.trigger[1].mode = digio.TRIG_FALLING
-- Set line 2 mode to falling edge.
digio.trigger[2].mode = digio.TRIG_FALLING
-- Assert trigger on line 1.
digio.trigger[1].assert()
-- When complete, wait for trigger on line 2.
digio.trigger[2].wait(2)
```

More information about triggering

To obtain precise timing and synchronization between instruments, use the remote trigger model. For more information about the remote trigger model and interactive triggering using other trigger objects, see [Triggering](#) (on page 7-1).

How do I store measurements in nonvolatile memory?

After the measurements are complete, you can save the reading buffer data to the nonvolatile memory in the instrument.

To save the reading buffer data:

1. From the virtual front panel, press the **STORE** key, and then select **SAVE**.
2. Select **INTERNAL** to save to internal nonvolatile memory.
3. Select one of the following:
 - **SMUA_BUFFER1**
 - **SMUA_BUFFER2**
 - **SMUB_BUFFER1**
 - **SMUB_BUFFER2**
4. The virtual front panel displays *Saving...* This may take awhile.
5. Press the **EXIT (LOCAL)** key to return to the main menu.

For additional information, see [Saving reading buffers](#) (on page 6-5).

When should I change the output-off state?

CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the 2606B to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other 2606B instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

The 2606B instrument provides multiple output-off states. The multiple states are required because different types of connected devices (or loads) require different behaviors from the 2606B when its 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. Therefore, careful selection of the proper output-off state is important to prevent damage to devices and instruments. This is especially true when the device can deliver energy to the 2606B, such as a battery or capacitor or when another SourceMeter instrument is connected across the output terminals. In these situations, use an output-off state that isolates the instrument from the device by using one of the following commands:

- `smuX.source.offfunc = smuX.OUTPUT_DCAMPS`
- `smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`

There are other guidelines to follow when connecting the output of multiple 2606B instruments to get a larger current or voltage. Please refer to the following references for more information:

- [Multiple SMU connections](#) (on page 5-24)
- Keithley application notes on tek.com/keithley

How do I make contact check measurements?

For information about making contact check measurements, see [Contact check measurements](#) (on page 5-21) and [Contact check](#) (on page 5-9).

How can I change the line frequency or voltage?

The 2606B requires a line voltage of 100 V ac to 240 V ac ($\pm 10\%$) and a line frequency of 50 Hz or 60 Hz.

Where can I get the LabVIEW driver?

The latest NI™ LabVIEW™ driver is available on tek.com/keithley.

Why is the reading value 9.91e37?

This value indicates that there is a measurement overflow error. This error occurs when:

- A measurement performed on a fixed range has a measured value greater than the specified range
- The measured value is larger than the maximum current or voltage range of the instrument (exceeds the instrument rating)

If the instrument displays the overflow message on a particular range, select a higher range until an on-range reading is displayed. To ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow.

In this section:

[Additional 2606B information](#) 19-1

Additional 2606B information

For additional information about the 2606B, 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*
- *Semiconductor Device Test Applications Guide*
- Application notes
- Updated drivers
- Information about related products, including:
 - Series 2600B System SourceMeter® Instruments
 - The Model 4200A-SCS Semiconductor Characterization System
 - The 2651A High Power System SourceMeter® Instrument
 - The 2657A High Power System SourceMeter® Instrument
 - The 2400 Series SMU SourceMeter Instruments with touchscreen user interface

In addition, your local Field Applications Engineer 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 • 1-800-833-9200 • tek.com/keithley

