Model 2651A High Power System SourceMeter® Instrument

Reference Manual

2651A-901-01 Rev. A / March 2011

Model 2651A
High Power System SourceMeter® Instrument
Reference Manual

© 2011, Keithley Instruments, Inc.

Cleveland, Ohio, U.S.A.

All rights reserved.

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

All Keithley Instruments product names are trademarks or registered trademarks of Keithley Instruments, Inc. Other brand names are trademarks or registered trademarks of their respective holders.

The Lua 5.0 software and associated documentation files are copyright © 1994-2008, Tecgraf, PUC-Rio. Terms of license for the Lua software and associated documentation can be accessed at the Lua licensing site (http://www.lua.org/license.html).

Document number: 2651A-901-01 Rev. A / March 2011

Safety Precautions

KEITHLEY

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 qualified 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 Instruments products are designed for use with electrical signals that are rated Measurement Category I and Measurement Category II, as described in the International Electrotechnical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Measurement Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Measurement Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the user documentation.

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 30V RMS, 42.4V peak, or 60VDC 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 1000V, 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.

The instrument and accessories must be used in accordance with its specifications and operating instructions, or the safety of the equipment may be impaired.

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

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 safety earth 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 safety earth ground using the wire recommended in the user documentation.

The ⚠ symbol on an instrument means caution, risk of danger. The user should 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 caution, 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 Ⓗg 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 dangers 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.

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 Instruments. Standard fuses with applicable national safety approvals may be used if the rating and type are the same. 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 Instruments to maintain accuracy and functionality of the product). If you are unsure about the applicability of a replacement component, call a Keithley Instruments office for information.

To clean an instrument, use a damp cloth 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.

Table of Contents

Introduction 1-1

Welcome 1-1

Extended warranty 1-1

Contact information 1-1

CD-ROM contents 1-2
- Organization of manual sections 1-2

Capabilities and features 1-2

General information 1-3
- Displaying the instrument's serial number 1-3

General operation 2-1

General ratings 2-1

Front and rear panel operation 2-1
- Front panel 2-1
- Rear panel 2-5

Cooling vents 2-7

Turning your instrument on and off 2-8
- Procedure 2-8

System identification 2-9

Beeper 2-10

Display mode 2-10

Basic operation 2-11
- Operation overview 2-11
- Menu overview 2-15
- Operation considerations 2-22
- Basic source-measure procedure 2-24
- Triggering in local mode 2-27
- Configuring trigger attributes in local mode 2-28
- Sink operation and interface 2-29
- Configuring for measure only tests using the MODE key 2-30
- Contact check measurements 2-37
- User setup 2-39

DUT test connections 2-42
- Input/output connectors 2-42
- 2-wire local sensing 2-43
- 4-wire remote sensing 2-44
- Sense mode selection 2-44
- Contact check connections 2-45
- Multiple SMU connections 2-46
- Combining SMU outputs 2-48
- Guarding and shielding 2-53
- Test fixture 2-57
- Floating a SMU 2-58

Output-off states ... 2-59

USB storage overview ... 2-63
Connecting the USB flash drive ... 2-63
File system navigation ... 2-64

Error and status messages ... 2-65

Range ... 2-65
Available ranges ... 2-65
Maximum source values and readings ... 2-66
Measure auto delay ... 2-66
Ranging limitations ... 2-66
Manual ranging ... 2-67
Autoranging ... 2-67
Low range limits ... 2-67
Range considerations ... 2-68
Range programming ... 2-69

Digits ... 2-71
Setting display resolution ... 2-71
Remote digits programming ... 2-71

Speed ... 2-71
Setting speed ... 2-72

Communications interfaces ... 2-73
Selecting an interface ... 2-73
Output queue ... 2-73
LAN communications ... 2-74
GPIB operation ... 2-76
General bus commands ... 2-78
Front-panel GPIB operation ... 2-80
RS-232 interface operation ... 2-81

Functions and features ... 3-1

Rel ... 3-1
Front panel rel ... 3-1
Remote rel programming ... 3-2

Filters ... 3-3
Filter types ... 3-3
Front panel filter control ... 3-4
Remote filter programming ... 3-4

Reading buffers ... 3-6
Front-panel reading buffer control ... 3-6
Remote reading buffer programming ... 3-10

Sweep operation ... 3-19
Overview ... 3-19
Sweep characteristics ... 3-20
Configuring and running sweeps ... 3-29
Sweeping using factory scripts ... 3-30
Sweep programming examples ... 3-30

Triggering ... 3-32
Remote triggering overview ... 3-32
Using the remote trigger model ... 3-34
SMU event detectors ... 3-39
Using trigger events to start actions on trigger objects ... 3-40
Digital I/O port and TSP-Link synchronization lines ... 3-41

Timers ... 3-43
Event blenders ... 3-49
LAN triggering overview ... 3-50
Command interface triggering ... 3-52
Manual triggering ... 3-52
Interactive triggering ... 3-52
Hardware trigger modes ... 3-56
Understanding synchronous triggering modes ... 3-60

High-capacitance mode ... 3-65
Overview ... 3-65
Understanding high-capacitance mode ... 3-65
Enabling high-capacitance mode ... 3-68

Display operations ... 3-70
Display functions and attributes ... 3-70
Display features ... 3-71
Display messages ... 3-72
Input prompting ... 3-76
Indicators ... 3-78
Local lockout ... 3-79
Load test menu ... 3-79
Key-press codes ... 3-81

Digital I/O ... 3-83
Digital I/O port ... 3-83
Using output enable ... 3-88
TSP-Link synchronization lines ... 3-89

Theory of operation ... 4-1

Analog to digital converter ... 4-1

Source-measure concepts ... 4-2
Overview ... 4-2
Compliance principles ... 4-2
Overheating protection ... 4-3
Operating boundaries ... 4-5
Basic circuit configurations ... 4-14
Guard ... 4-17
Cable considerations ... 4-20

Reduction in gain-bandwidth ... 4-20

Pulse rise times ... 4-21
Cable specifications and connection configuration ... 4-22
Range and pulse setting ... 4-22
Load and operating mode ... 4-22

Pulse width ... 4-23

Remote commands ... 5-1

Introduction to remote operation ... 5-1
Controlling the instrument by sending individual command messages ... 5-1
Queries ... 5-3
Information on scripting and programming ... 5-3

About remote commands ... 5-3
Beeper control ... 5-3
Bit manipulation and logical operations ... 5-4

Data queue ... 5-5
Digital I/O ... 5-5
Display ... 5-6
Error queue ... 5-6
Event log ... 5-6
File I/O ... 5-7
GPIB ... 5-8
Instrument identification ... 5-8
LAN and LXI ... 5-9
Miscellaneous ... 5-10
Parallel script execution ... 5-10
Queries and response messages ... 5-10
Reading buffer ... 5-11
Reset ... 5-11
RS-232 ... 5-11
Saved setups ... 5-11
Scripting ... 5-12
SMU ... 5-13
SMU calibration ... 5-14
Status model ... 5-15
Time ... 5-16
Triggering ... 5-17
TSP-Link ... 5-19
TSP-Net ... 5-19
Userstrings ... 5-20

Factory scripts ... 5-20
Introduction ... 5-20
Running a factory script ... 5-21
Retrieving and modifying a factory script listing ... 5-21
KISweep factory script ... 5-22
KIPulse factory script ... 5-23
KIHighC factory script ... 5-24
KIParlib factory script ... 5-24
KISavebuffer factory script ... 5-24

Instrument programming ... 6-1

Fundamentals of scripting for TSP ... 6-1
What is a script? ... 6-2
Runtime and nonvolatile memory storage of scripts ... 6-2
What can be included in scripts? ... 6-2
Commands that cannot be used in scripts ... 6-2
Manage scripts ... 6-3
Working with scripts in nonvolatile memory ... 6-10
Programming example ... 6-12

Fundamentals of programming for TSP ... 6-14
Introduction ... 6-14
What is Lua? ... 6-14
Lua basics ... 6-14
Standard libraries ... 6-30
Programming example ... 6-36

Using Test Script Builder (TSB) ... 6-37
Installing the TSB software ... 6-37
Project Navigator ... 6-37
Script Editor ... 6-37
Programming interaction ... 6-37

Working with TSB Embedded ... 6-37

Using instrument commands 6-37

Advanced scripting for TSP 6-39
Global variables and the script.user.scripts table 6-39
Create a script using the script.new() command 6-40
Restore a script to the runtime environment 6-43
Rename a script 6-43
Delete user scripts from the instrument 6-45
Memory considerations for the runtime environment 6-46

TSP-Link system expansion interface 6-47
Overview 6-47
Connections 6-49
Initialization 6-50
Resetting the TSP-Link network 6-51
Using the expanded system 6-52
TSP advanced features 6-53
Using groups to manage nodes on TSP-Link network 6-56
Running parallel test scripts 6-57
Using the data queue for real-time communication 6-59
Copying test scripts across the TSP-Link network 6-59
Removing stale values from the reading buffer 6-60

TSP-Net 6-60
Overview 6-60
TSP-Net capabilities 6-61
Using TSP-Net with any Ethernet-enabled device 6-61
TSP-Net versus TSP-Link to communicate with TSP-enabled devices 6-62
Instrument commands: General device control 6-63
Instrument commands: TSP-enabled device control 6-63
Example: Using tspnet commands 6-64

Command reference 7-1

Command programming notes 7-1
Placeholder text 7-1
Syntax rules 7-2
Logical instruments 7-3
Time and date values 7-4

Using the command reference 7-4
Command name and standard parameters summary 7-5
Command usage 7-6
Command details 7-7
Example section 7-7
Related commands and information 7-7

Commands 7-8
beeper.beep() 7-8
beeper.enable 7-8
bit.bitand() 7-9
bit.bitor() 7-9
bit.bitxor() 7-10
bit.clear() 7-11
bit.get() 7-11
bit.getfield() 7-12
bit.set() 7-13
bit.setfield() 7-14
bit.test() 7-15
bit.toggle() 7-15
bufferVar.appendmode 7-16

bufferVar.basetimestamp 7-17
bufferVar.cachemode 7-18
bufferVar.capacity 7-18
bufferVar.clear() 7-19
bufferVar.clearcache() 7-20
bufferVar.collectsourcevalues 7-21
bufferVar.collecttimestamps 7-21
bufferVar.fillcount 7-22
bufferVar.fillmode 7-23
bufferVar.n 7-24
bufferVar.timestampresolution 7-24
ConfigPulseIMeasureV() 7-25
ConfigPulseIMeasureVSweepLin() 7-27
ConfigPulseIMeasureVSweepLog() 7-29
ConfigPulseVMeasureI() 7-31
ConfigPulseVMeasureISweepLin() 7-34
ConfigPulseVMeasureISweepLog() 7-36
dataqueue.add() 7-38
dataqueue.CAPACITY 7-39
dataqueue.clear() 7-39
dataqueue.count 7-40
dataqueue.next() 7-41
delay() 7-42
digio.readbit() 7-43
digio.readport() 7-43
digio.trigger[N].assert() 7-44
digio.trigger[N].clear() 7-44
digio.trigger[N].EVENT_ID 7-45
digio.trigger[N].mode 7-46
digio.trigger[N].overrun 7-47
digio.trigger[N].pulsewidth 7-48
digio.trigger[N].release() 7-48
digio.trigger[N].reset() 7-49
digio.trigger[N].stimulus 7-49
digio.trigger[N].wait() 7-51
digio.writebit() 7-51
digio.writeport() 7-52
digio.writeprotect 7-53
display.clear() 7-54
display.getannunciators() 7-54
display.getcursor() 7-56
display.getlastkey() 7-56
display.gettext() 7-58
display.inputvalue() 7-59
display.loadmenu.add() 7-61
display.loadmenu.catalog() 7-62
display.loadmenu.delete() 7-63
display.locallockout 7-64
display.menu() 7-64
display.numpad 7-65
display.prompt() 7-66
display.screen 7-67
display.sendkey() 7-68
display.setcursor() 7-69
display.settext() 7-70
display.smuX.digits 7-71
display.smuX.measure.func 7-71
display.trigger.clear() 7-72
display.trigger.EVENT_ID 7-72
display.trigger.overrun 7-73
display.trigger.wait() 7-73

display.waitkey() 7-74
errorqueue.clear() 7-76
errorqueue.count 7-76
errorqueue.next() 7-77
eventlog.all() 7-78
eventlog.clear() 7-78
eventlog.count 7-79
eventlog.enable 7-80
eventlog.next() 7-80
eventlog.overwritemethod 7-81
exit() 7-82
fileVar:close() 7-82
fileVar:flush() 7-83
fileVar:read() 7-83
fileVar:seek() 7-84
fileVar:write() 7-85
format.asciiprecision 7-86
format.byteorder 7-87
format.data 7-87
fs.chdir() 7-88
fs.cwd() 7-89
fs.is_dir() 7-89
fs.is_file() 7-90
fs.mkdir() 7-90
fs.readdir() 7-91
fs.rmdir() 7-91
gettimezone() 7-92
gm_isweep() 7-92
gm_vsweep() 7-93
gpib.address 7-94
i_leakage_measure() 7-95
i_leakage_threshold() 7-96
InitiatePulseTest() 7-97
io.close() 7-99
io.flush() 7-99
io.input() 7-100
io.open() 7-100
io.output() 7-101
io.read() 7-101
io.type() 7-102
io.write() 7-103
lan.applysettings() 7-103
lan.autoconnect 7-104
lan.config.dns.address[N] 7-104
lan.config.dns.domain 7-105
lan.config.dns.dynamic 7-106
lan.config.dns.hostname 7-107
lan.config.dns.verify 7-107
lan.config.duplex 7-108
lan.config.gateway 7-108
lan.config.ipaddress 7-109
lan.config.method 7-110
lan.config.speed 7-110
lan.config.subnetmask 7-111
lan.linktimeout 7-112
lan.lxidomain 7-113
lan.nagle 7-113
lan.reset() 7-114
lan.restoredefaults() 7-114
lan.status.dns.address[N] 7-115
lan.status.dns.name 7-116

lan.status.duplex 7-117
lan.status.gateway 7-117
lan.status.ipaddress 7-118
lan.status.macaddress 7-118
lan.status.port.dst 7-119
lan.status.port.rawsocket 7-119
lan.status.port.telnet 7-120
lan.status.port.vxi11 7-120
lan.status.speed 7-121
lan.status.subnetmask 7-121
lan.timedwait 7-122
lan.trigger[N].assert() 7-122
lan.trigger[N].clear() 7-123
lan.trigger[N].connect() 7-124
lan.trigger[N].connected 7-124
lan.trigger[N].disconnect() 7-125
lan.trigger[N].EVENT_ID 7-126
lan.trigger[N].ipaddress 7-126
lan.trigger[N].mode 7-127
lan.trigger[N].overrun 7-128
lan.trigger[N].protocol 7-129
lan.trigger[N].pseudostate 7-129
lan.trigger[N].stimulus 7-130
lan.trigger[N].wait() 7-131
localnode.autolinefreq 7-132
localnode.description 7-133
localnode.linefreq 7-133
localnode.model 7-134
localnode.password 7-134
localnode.passwordmode 7-135
localnode.prompts 7-136
localnode.prompts4882 7-137
localnode.reset() 7-137
localnode.revision 7-138
localnode.serialno 7-138
localnode.showerrors 7-139
makegetter() 7-140
makesetter() 7-140
meminfo() 7-141
node[N].execute() 7-142
node[N].getglobal() 7-142
node[N].setglobal() 7-143
opc() 7-144
print() 7-144
printbuffer() 7-145
printnumber() 7-146
PulseIMeasureV() 7-147
PulseVMeasureI() 7-148
QueryPulseConfig() 7-149
reset() 7-151
savebuffer() 7-152
script.anonymous 7-153
script.delete() 7-154
script.load() 7-154
script.new() 7-155
script.newautorun() 7-157
script.restore() 7-157
script.run() 7-158
script.user.catalog() 7-158
scriptVar.autorun 7-159
scriptVar.list() 7-160

scriptVar.name 7-160
scriptVar.run() 7-161
scriptVar.save() 7-162
scriptVar.source 7-162
serial.baud 7-163
serial.databits 7-164
serial.flowcontrol 7-164
serial.parity 7-165
serial.read() 7-166
serial.write() 7-167
settime() 7-167
settimezone() 7-168
setup.poweron 7-169
setup.recall() 7-170
setup.save() 7-171
smuX.abort() 7-171
smuX.buffer.getstats() 7-172
smuX.buffer.recalculatestats() 7-173
smuX.cal.adjustdate 7-174
smuX.cal.date 7-175
smuX.cal.due 7-175
smuX.cal.fastadc() 7-176
smuX.cal.lock() 7-177
smuX.cal.password 7-177
smuX.cal.polarity 7-178
smuX.cal.restore() 7-179
smuX.cal.save() 7-180
smuX.cal.state 7-180
smuX.cal.unlock() 7-181
smuX.contact.calibratehi() 7-182
smuX.contact.calibratelo() 7-183
smuX.contact.check() 7-186
smuX.contact.r() 7-187
smuX.contact.speed 7-188
smuX.contact.threshold 7-189
smuX.makebuffer() 7-189
smuX.measure.adc 7-190
smuX.measure.autorangeY 7-191
smuX.measure.autozero 7-192
smuX.measure.calibrateY() 7-193
smuX.measure.count 7-194
smuX.measure.delay 7-195
smuX.measure.delayfactor 7-196
smuX.measure.filter.count 7-196
smuX.measure.filter.enable 7-197
smuX.measure.filter.type 7-198
smuX.measure.highcrangedelayfactor 7-199
smuX.measure.interval 7-199
smuX.measure.lowrangeY 7-200
smuX.measure.nplc 7-201
smuX.measure.overlappedY() 7-201
smuX.measure.rangeY 7-202
smuX.measure.rel.enableY 7-204
smuX.measure.rel.levelY 7-205
smuX.measure.Y() 7-205
smuX.measureYandstep() 7-207
smuX.nvbufferY 7-208
smuX.reset() 7-209
smuX.savebuffer() 7-209
smuX.sense 7-210
smuX.source.autorangeY 7-211

smuX.source.calibrateY() 7-211
smuX.source.compliance 7-213
smuX.source.delay 7-213
smuX.source.func 7-214
smuX.source.highc 7-215
smuX.source.levelY 7-216
smuX.source.limitY 7-217
smuX.source.lowrangeY 7-218
smuX.source.offfunc 7-218
smuX.source.offlimitY 7-219
smuX.source.offmode 7-220
smuX.source.output 7-221
smuX.source.outputenableaction 7-222
smuX.source.rangeY 7-223
smuX.source.settling 7-224
smuX.source.sink 7-225
smuX.trigger.arm.count 7-226
smuX.trigger.arm.set() 7-226
smuX.trigger.arm.stimulus 7-227
smuX.trigger.ARMED_EVENT_ID 7-228
smuX.trigger.autoclear 7-229
smuX.trigger.count 7-230
smuX.trigger.endpulse.action 7-230
smuX.trigger.endpulse.set() 7-231
smuX.trigger.endpulse.stimulus 7-231
smuX.trigger.endsweep.action 7-233
smuX.trigger.IDLE_EVENT_ID 7-233
smuX.trigger.initiate() 7-234
smuX.trigger.measure.action 7-235
smuX.trigger.measure.set() 7-236
smuX.trigger.measure.stimulus 7-237
smuX.trigger.measure.Y() 7-238
smuX.trigger.MEASURE_COMPLETE_EVENT_ID 7-239
smuX.trigger.PULSE_COMPLETE_EVENT_ID 7-240
smuX.trigger.source.action 7-241
smuX.trigger.source.limitY 7-242
smuX.trigger.source.linearY() 7-243
smuX.trigger.source.listY() 7-244
smuX.trigger.source.logY() 7-245
smuX.trigger.source.set() 7-246
smuX.trigger.source.stimulus 7-247
smuX.trigger.SOURCE_COMPLETE_EVENT_ID 7-248
smuX.trigger.SWEEP_COMPLETE_EVENT_ID 7-249
smuX.trigger.SWEEPING_EVENT_ID 7-249
status.condition 7-250
status.measurement.* 7-252
status.measurement.buffer_available.* 7-254
status.measurement.current_limit.* 7-255
status.measurement.instrument.* 7-256
status.measurement.instrument.smuX.* 7-257
status.measurement.reading_overflow.* 7-260
status.measurement.voltage_limit.* 7-261
status.node_enable 7-262
status.node_event 7-264
status.operation.* 7-266
status.operation.calibrating.* 7-268
status.operation.instrument.* 7-269
status.operation.instrument.digio.* 7-271
status.operation.instrument.digio.trigger_overrun.* 7-273
status.operation.instrument.lan.* 7-275
status.operation.instrument.lan.trigger_overrun.* 7-277

status.operation.instrument.smuX.* 7-279
status.operation.instrument.smuX.trigger_overrrun.* 7-281
status.operation.instrument.trigger_blender.* 7-283
status.operation.instrument.trigger_blender.trigger_overrun.* 7-284
status.operation.instrument.trigger_timer.* 7-286
status.operation.instrument.trigger_timer.trigger_overrun.* 7-287
status.operation.instrument.tsplink.* 7-289
status.operation.instrument.tsplink.trigger_overrun.* 7-290
status.operation.measuring.* 7-292
status.operation.remote.* 7-293
status.operation.sweeping.* 7-295
status.operation.trigger_overrun.* 7-296
status.operation.user.* 7-298
status.questionable.* 7-300
status.questionable.calibration.* 7-301
status.questionable.instrument.* 7-303
status.questionable.instrument.smuX.* 7-304
status.questionable.over_temperature.* 7-306
status.questionable.unstable_output.* 7-307
status.request_enable 7-308
status.request_event 7-310
status.reset() 7-312
status.standard.* 7-312
status.system.* 7-314
status.system2.* 7-316
status.system3.* 7-318
status.system4.* 7-320
status.system5.* 7-322
SweepILinMeasureV() 7-324
SweepIListMeasureV() 7-325
SweepILogMeasureV() 7-326
SweepVLinMeasureI() 7-328
SweepVListMeasureI() 7-329
SweepVLogMeasureI() 7-330
timer.measure.t() 7-332
timer.reset() 7-333
trigger.blender[N].clear() 7-333
trigger.blender[N].EVENT_ID 7-334
trigger.blender[N].orenable 7-334
trigger.blender[N].overrun 7-335
trigger.blender[N].reset() 7-335
trigger.blender[N].stimulus[M] 7-336
trigger.blender[N].wait() 7-337
trigger.clear() 7-338
trigger.EVENT_ID 7-339
trigger.timer[N].clear() 7-339
trigger.timer[N].count 7-339
trigger.timer[N].delay 7-340
trigger.timer[N].delaylist 7-341
trigger.timer[N].EVENT_ID 7-341
trigger.timer[N].overrun 7-342
trigger.timer[N].passthrough 7-343
trigger.timer[N].reset() 7-343
trigger.timer[N].stimulus 7-344
trigger.timer[N].wait() 7-345
trigger.wait() 7-346
tsplink.group 7-347
tsplink.master 7-347
tsplink.node 7-347
tsplink.readbit() 7-348
tsplink.readport() 7-349

tsplink.reset() 7-349
tsplink.state 7-350
tsplink.trigger[N].assert() 7-350
tsplink.trigger[N].clear() 7-351
tsplink.trigger[N].EVENT_ID 7-352
tsplink.trigger[N].mode 7-352
tsplink.trigger[N].overrun 7-354
tsplink.trigger[N].pulsewidth 7-355
tsplink.trigger[N].release() 7-355
tsplink.trigger[N].reset() 7-356
tsplink.trigger[N].stimulus 7-357
tsplink.trigger[N].wait() 7-358
tsplink.writebit() 7-359
tsplink.writeport() 7-359
tsplink.writeprotect 7-360
tspnet.clear() 7-361
tspnet.connect() 7-361
tspnet.disconnect() 7-363
tspnet.execute() 7-363
tspnet.idn() 7-364
tspnet.read() 7-365
tspnet.readavailable() 7-366
tspnet.reset() 7-366
tspnet.termination() 7-367
tspnet.timeout 7-368
tspnet.tsp.abort() 7-368
tspnet.tsp.abortonconnect 7-369
tspnet.tsp.rbtablecopy() 7-369
tspnet.tsp.runscript() 7-370
tspnet.write() 7-371
userstring.add() 7-372
userstring.catalog() 7-372
userstring.delete() 7-373
userstring.get() 7-374
waitcomplete() 7-374

Troubleshooting guide 8-1

Introduction 8-1
Error summary 8-1
Error effects on scripts 8-1
Reading errors 8-2
Error summary list 8-2
LAN troubleshooting suggestions 8-6

Frequently asked questions (FAQs) 9-1

How do I optimize performance? 9-1
Setting speed 9-1
Disabling autozero to increase speed 9-1
How do I upgrade the firmware? 9-2
How do I use the Digital I/O port? 9-2
How do I trigger other instruments? 9-2
Triggering a scanner 9-2

Programming triggering 9-3
More information on triggering 9-3
How do I generate a service request? 9-3
Setting up a service request 9-3
Service request programming example 9-4
Polling for SRQs 9-4
How do I store measurements in nonvolatile memory? 9-4
When should I change the output off state? 9-5
How do I make contact check measurements? 9-5

Next steps 10-1

Next steps 10-1
Additional Model 2651A information 10-1

Maintenance A-1

Line fuse replacement A-1
Front panel tests A-2
Keys test A-3
Display patterns test A-4
Upgrading the firmware A-4
Firmware upgrade using the instrument web interface A-5

Calibration B-1

Verification B-1
Verification test requirements B-1
Restoring factory defaults B-4
Performing the verification test procedures B-4
Current source accuracy B-6
Current measurement accuracy B-8
Voltage source accuracy B-13
Voltage measurement accuracy B-14
Adjustment B-16
Introduction B-16
Environmental conditions B-16
Calibration considerations B-17
Calibration overview B-19
Calibration commands quick reference B-21
Calibration adjustment procedure B-22

LAN concepts and settings C-1

Establishing a point-to-point connection C-1
Step 1: Identify and record the existing IP configuration C-2
Step 2: Disable DHCP to use the computer's existing IP address C-4
Step 3: Configure the instrument's LAN settings C-6
Step 4: Install the crossover cable C-7
Step 5: Access the instrument's internal web page C-7
Connecting to the LAN C-8

Setting the LAN configuration method....C-8
Setting the IP address....C-8
Setting the gateway....C-9
Setting the subnet mask....C-9
Configuring the domain name system (DNS)....C-10
LAN speeds....C-11
Duplex mode....C-11
Viewing LAN status messages....C-12
Viewing the network settings....C-13
Confirming the active speed and duplex negotiation....C-13
Confirming port numbers....C-14
Selecting a remote command interface....C-14
VXI - 11 connection....C-14
Raw socket connection....C-14
Dead socket connection....C-15
Telnet connection....C-15
Logging LAN trigger events in the event log....C-19
Accessing the event log from the command interface....C-20

Common commands....D-1

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

Status model....E-1

Status Byte Register....E-1
Status register sets....E-2
Queues....E-2
Service requests and connections....E-2
Status function summary....E-3
Status model diagrams....E-4
Clearing registers and queues....E-12
Programming and reading registers....E-13
Programming enable and transition registers....E-13
Reading registers....E-14
Status byte and service request (SRQ)....E-14
Status Byte Register....E-14
Service Request Enable Register....E-16
Serial polling and SRQ....E-17
SPE, SPD (serial polling)....E-17
Status byte and service request commands....E-17
Enable and transition registers....E-18
Controlling node and SRQ enable registers....E-18

Status register sets E-18
System Summary Registers E-19
Standard Event Register E-19
Operation Status Registers E-20
Measurement Event Registers E-23
Register programming example E-24

Queues E-25
Output queue E-25
Error queue E-25

TSP-Link system status E-26
Status model configuration example E-26

Display character codes F-1

Display character dot patterns F-5

Index I-1

Section 1

Introduction

In this section:

Welcome 1-1
Extended warranty 1-1
Contact information 1-1
CD-ROM contents 1-2
Capabilities and features 1-2
General information 1-3

Welcome

Thank you for choosing a Keithley Instruments product. The Model 2651A High Power System SourceMeter® Instrument provides manufacturers of electronic components and semiconductor devices a high-power source-measure unit (SMU). A SMU is an instrument that combines source and measurement capabilities in a single instrument. This combination simplifies test processes by eliminating synchronization and connection issues associated with multiple instrument solutions. A Model 2651A 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 with the Series 2600A.

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 representative for details.

Contact information

If you have any questions after reviewing this information, please contact your local Keithley Instruments representative or call Keithley Instruments corporate headquarters (toll-free inside the U.S. and Canada only) at 1-888-KEITHLEY (1-888-534-8453), or from outside the U.S. at +1-440-248-0400. For worldwide contact numbers, visit the Keithley Instruments website (http://www.keithley.com).

CD-ROM contents

Two CD-ROMs are shipped with each Model 2651A order. The Model 2651A Quick Start Guide, User's Manual, and Reference Manual are provided in PDF format on the Model 2651A Product Information CD-ROM.

- **Quick Start Guide:** Provides unpacking instructions, describes basic connections, and reviews basic operation information. If you are new to Keithley Instruments equipment, refer to the Quick Start Guide to take the steps needed to unpack, set up, and verify operation.
- **User's Manual:** Provides application examples. If you need a starting point to begin creation of applications, refer to the User's Manual for a variety of specific examples.
- **Reference Manual:** Includes advanced operation topics and maintenance information. Programmers looking for a command reference, and users looking for an in-depth description of the way the instrument works (including troubleshooting and optimization), should refer to the Reference Manual.

A second CD-ROM contains the Test Script Builder script development software (Keithley Instruments part number KTS-850). Use this CD-ROM to install the Test Script Builder Integrated Development Environment. This software not only provides an environment to develop a test program, but also 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.

Organization of manual sections

Bookmarks for each manual section have been provided in the PDF (see Adobe® Acrobat® or Reader® help for information about bookmarks). The manual sections are also listed in the Table of Contents located at the beginning of this manual.

Capabilities and features

All Model 2651A High Power System SourceMeter® Instruments have the following features:

- Resistance and power measurement functions
- Four-quadrant sink or source operation
- Contact check function
- High capacitance mode for load impedance up to 50 μF (microfarads)
- Linear, logarithmic, and custom sweeping and pulsing
- Filtering to reduce reading noise
- High-speed sampling fast analog to digital converter (ADC); the fast ADC uses a hardware buffer that can store up to 5,000 readings in a single measure action
- 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 Model 2651A to control other devices
- Web-based characterization tool that provides easy access to data gathering, sweeping, and pulsing features
- LXI® Class C 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 IEEE-488 (GPIB), RS-232, and local area network (LAN) connections

Additional source and measure features:

- Source ± DC voltage from 1 μV to 40.4 V
- Source ± DC current from 10 pA to 20.2 A
- Source ± pulse current up to 50 A
- Measure ± pulse current up to 50 A
- Measure ± DC voltage from 1 μV to 40.4 V
- Measure ± DC current from 10 pA to 20.2 A

General information

Displaying the instrument's serial number

To display the serial number on the front panel:

1. If the Model 2651A is in remote operation, press the **EXIT (LOCAL)** key once to place the instrument in local operation.
2. Press the **MENU** key.
3. Use the navigation wheel ◷ to scroll to the **SYSTEM-INFO** menu.
4. Press the **ENTER** key.
5. On the SYSTEM INFORMATION menu, scroll to the **SERIAL#** menu item and press the **ENTER** key. The Model 2651A serial number is displayed.

Section 2

General operation

In this section:

General ratings 2-1
Front and rear panel operation 2-1
Cooling vents 2-7
Turning your instrument on and off 2-8
System identification 2-9
Beeper 2-10
Display mode 2-10
Basic operation 2-11
DUT test connections 2-42
USB storage overview 2-63
Error and status messages 2-65
Range 2-65
Digits 2-71
Speed 2-71
Communications interfaces 2-73

General ratings

The Model 2651A instrument's general ratings and connections are contained in the following table.

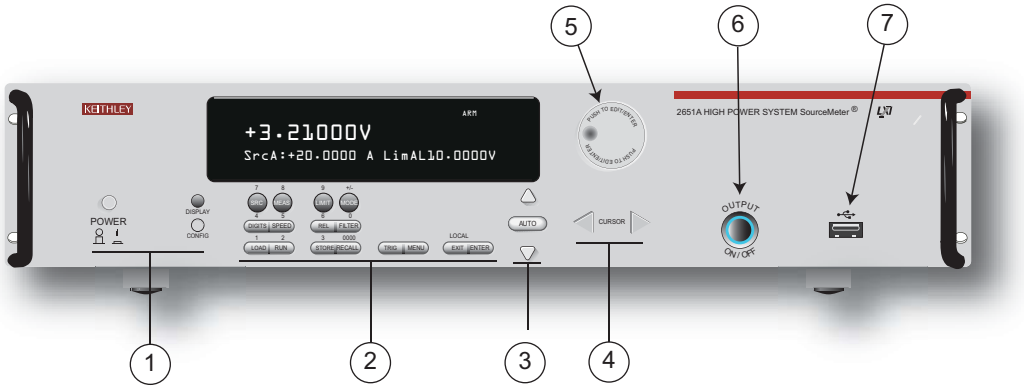
Category	Specification
Supply voltage range	100 V AC to 240 V AC, 50 Hz to 60 Hz (autosensing), 550 VA maximum
Input and output connections	See Front and rear panel operation (on page 2-1)
Environmental conditions	**For indoor use only:** Altitude: Maximum 2000 meters above sea level Operating: 0° C 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

Front and rear panel operation

Front panel

The front panel of the Model 2651A is shown below. The descriptions of the front-panel controls follow the figure.

Figure 1: Front panel Model 2651A



1. Special keys and power switch

POWER	Power switch. The in position turns the Model 2651A on (I); the out position turns it off (O).
DISPLAY	Toggles between the various source-measure displays and the user message mode.
CONFIG	Use to configure a function or operation.

2. Setup, performance control, special operation, and numbers

Source-measure setup

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

Performance control

DIGITS	Cycles through display resolution (4-1/2, 5-1/2, or 6-1/2 digits).
SPEED	Sets the measurement speed (FAST, MEDium, NORMAL, HI-ACCURACY, or OTHER). Setting the speed selects either the fast or integrating A/D converter. When FAST is selected, the fast A/D converter is used. When the integrating A/D converter is selected, the measurement speed and accuracy are set by controlling the measurement aperture. Also see Speed (on page 2-71).
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 test for execution (FACTORY, USER, or SCRIPTS).
RUN	Runs the last selected factory or user-defined tests.
STORE	Accesses reading buffers and takes readings. TAKE_READINGS: Sets number of readings to make and also takes and stores readings. SAVE: Stores readings in one of two internal buffers (SMUA_BUFFER1 or SMUA_BUFFER2), or to a user-installed flash drive (USB1) in CSV or XML format. Readings include measurements, source values, and time stamp values, if configured.
RECALL	Recalls information (DATA or STATISTICS) stored in either CHANA-BUFF1 or CHANA-BUFF2: 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 2-16). The main menu can be used to configure many facets of operation.
EXIT	Cancels selection and backs out of the menu structure. Also used as a **LOCAL** key to take the unit out of remote operation.
ENTER	Accepts selection and moves to the next choice or exits the menu.

Numbers

Number keys	When enabled, the number keys (0-9, +/-, 0000) allow direct numeric entry in the EDIT mode. Press the navigation wheel ⟲ to enter EDIT mode. For more information, see Setting a value (on page 2-20).

3. 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 non-range numbers (as an example, when editing the limit value).

4. Cursor keys

Use the **CURSOR** keys to move the cursor left or right. When the cursor is on the desired source or compliance value digit, push the navigation wheel ◯ to enter edit mode, and turn the navigation wheel to edit the value. Push the navigation wheel again when finished editing.

Use the **CURSOR** keys (or navigation wheel) to navigate 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 sub-menu item.

5. Navigation wheel

(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

6. Output control

Turns the source output on or off.

7. USB port

(USB port)

Use the USB port to connect a USB flash drive. The USB flash drive can be used to store reading buffer data, scripts, and user setup options.

8. Display indicators (not shown)

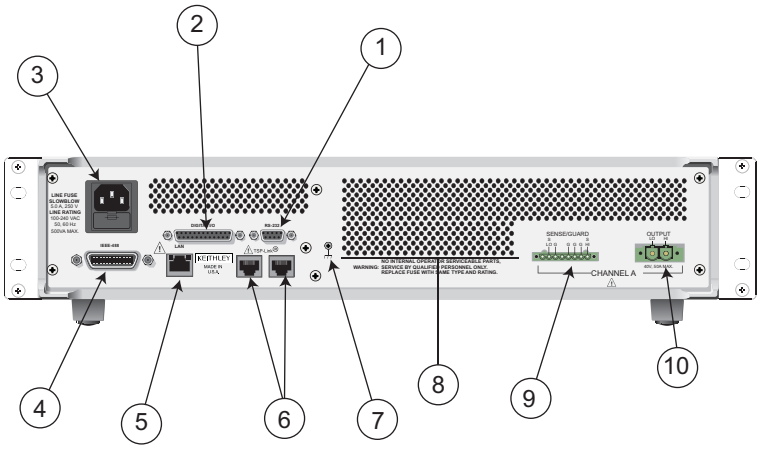
The items listed below represent the possible display indicators and their meaning.

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

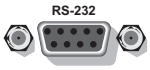
Rear panel

The rear panel of Model 2651A is shown below. The descriptions of the rear-panel components follow the figure.

Figure 2: Rear panel Model 2651A

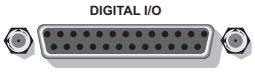


1. RS-232



Female DB9 connector. For RS-232 operation, use a straight-through (not null modem) DB9 shielded cable (Keithley Instruments Model 7009-5) for connection to the PC.

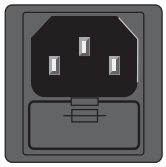
2. DIGITAL I/O



Female DB-25 connector. Pins provided: Fourteen digital input or output pins, seven GND pins, three +5 V pins, and one pin for output enable.

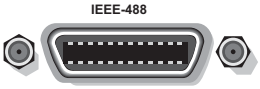
Use a cable equipped with a male DB-25 connector (Keithley Instruments part number CA-126-1).

3. Power module



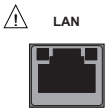
Contains the AC line receptacle and power line fuse. The instrument can operate on line voltages of 100 V to 240 V AC at line frequencies of 50 Hz or 60 Hz.

4. IEEE-488



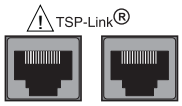
Connector for IEEE-488 (GPIB) operation. Use a shielded cable, such as the Keithley Instruments Model 7007-1 or Model 7007-2.

5. LAN



Use this RJ-45 connector to connect the instrument to the local area network. This interface supports Auto-MDIX, so either a CAT-5 cross-over cable (provided), or a normal CAT-5 straight-through cable (not provided) can be used.

6. TSP-link

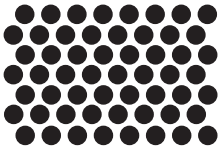


Expansion interface that allows a Model 2651A and other TSP-enabled instruments to trigger and communicate with each other. Use a category 5e or higher LAN crossover cable (Keithley Instruments part number CA-180-3A).

7. Chassis ground

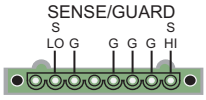
Ground screw for connections to chassis ground.

8. Cooling exhaust vent



Exhaust vent for the internal cooling fan. Keep the vent free of obstructions to prevent overheating. Also see Cooling vents (on page 2-7).

9. Terminal connector



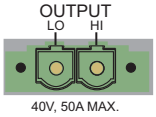
This connector provides input/output connections for sense (S HI/S LO) and guard (G). Connections are as follows:

S LO = Sense LO

G = Guard

S HI = Sense HI

10. OUTPUT connector (40 V, 50 A maximum)



This connector provides HI/LO connection points for currents. Connections are as follows:

LO = Input/Output LO

HI = Input/Output HI

Cooling vents

The Model 2651A has top and side intake vents and a rear exhaust vent. Either the top or both side intake vents, as well as the rear exhaust vent, must be unobstructed to properly dissipate heat. Do not place a container of liquid (water or coffee for instance) on the top cover. If it spills, the liquid may enter the case through the vents and cause severe damage.

Excessive heat could damage the Model 2651A and degrade its performance. The Model 2651A must be operating in an environment where the ambient temperature does not exceed 50° C.

⚠ CAUTION

To prevent damaging heat build-up and ensure specified performance:

Either the top or both side intake vents, as well as the rear exhaust vent, must be kept free of any obstructions. Even partial blockage could impair proper cooling.

DO NOT position any devices adjacent to the Model 2651A that force air (heated or unheated) toward its cooling vents or surfaces. This additional airflow could compromise accuracy.

When rack mounting the Model 2651A, make sure there is adequate airflow around both sides to ensure proper cooling. Adequate airflow enables air temperatures within approximately one inch of the Model 2651A surfaces to remain within specified limits under all operating conditions.

Rack mounting high power dissipation equipment adjacent to the Model 2651A could cause excessive heating to occur. To produce specified Model 2651A accuracies, maintain the specified ambient temperature around the surfaces of the Model 2651A. Proper cooling practice, in rack configurations with convection cooling only, 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 Model 2651A). Mount precision equipment as low as possible in the rack, where temperatures are coolest. Adding space panels below the Model 2651A will help provide adequate airflow.

Turning your instrument on and off

Procedure

Follow the procedure below to connect the Model 2651A to line power and turn on the instrument. The Model 2651A 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.

⚠ CAUTION

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

To turn a Model 2651A 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 female 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 Model 2651A contains a separate 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. Failure to use a grounded outlet may result in personal injury or death due to electric shock.

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.

Warm-up

The Model 2651A 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 Model 2651A to automatically detect the line frequency (either 50 Hz or 60 Hz) at each power-up. This detected line frequency will be used for aperture (NPLC) calculations. In noisy environments, you can manually configure the unit to match the actual line frequency.

To configure the line frequency from the 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 back out of the menu structure.

To configure the line frequency from a remote interface:

Set the `localnode.linefreq` or the `localnode.autolinefreq` attribute. The following programming example illustrates how to set the line frequency to 60 Hz:

```
localnode.linefreq = 60
```

The following programming example illustrates how to remotely configure the instrument to automatically detect line frequency at each power-up:

```
localnode.autolinefreq = true
```

Fuse replacement

A rear panel fuse drawer is located below the AC receptacle (refer to 3. Power Module (on page 2-6)). This fuse protects the power line input of the instrument. If the line fuse needs to be replaced, refer to Line fuse replacement (on page A-1).

System identification

Serial number, firmware revision, and calibration dates can be displayed by selecting **SYSTEM-INFO** from the main menu.

To view the system information:

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

For remote programming, use the `*IDN?` query to read system information.

Beeper

With the beeper enabled, a beep will be issued to acknowledge the following actions:

- A short beep, emulating a key click, is issued when a front-panel key is pressed.
- A short beep is also issued when the navigation wheel ◌ is turned or pressed.
- A longer beep is issued when the source output changes state (turned on or off).

To turn the beeper on or off from the 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 a remote interface:

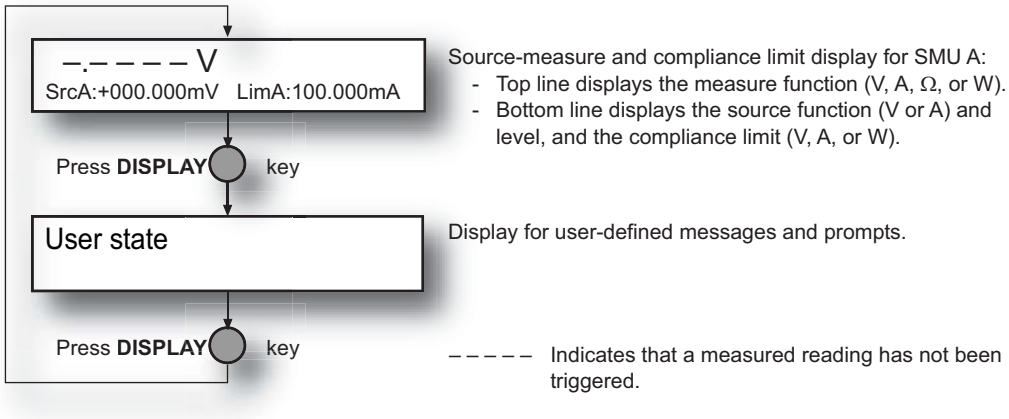
Set the `beeper.enable` attribute. The following programming example illustrates how to enable the beeper:

```
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 3-70) for information about display messaging.

Figure 3: Display modes



Basic operation

Operation overview

Quick Tip

Before you begin any of the following front panel procedures, make sure that you exit out of the menu structure. Press the **EXIT (LOCAL)** key as needed to back out of a menu.

Source-measure capabilities

From the front panel, the instrument can be configured 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

Voltage and current

The following table lists the source and measure limits for the voltage and current functions. The full range of operation is explained in Operating boundaries (on page 4-5) in the Theory of Operations.

Model 2651A source-measure capabilities

Range	Source	Measure
100 mV	± 101 mV	± 102 mV
1 V	± 1.01 V	± 1.02 V
10 V	± 10.1 V	± 10.2 V
20 V	± 20.2 V	± 20.4 V
40 V	± 40.4 V	± 40.8 V
1 µA	± 1.01 µA	± 1.02 µA
10 µA	± 10.1 µA	± 10.2 µA
100 µA	± 101 µA	± 102 µA
1 mA	± 1.01 mA	± 1.02 mA
10 mA	± 10.1 mA	± 10.2 mA
100 mA	± 101 mA	± 102 mA
1 A	± 1.01 A	± 1.02 A
5 A	± 5.05 A	± 5.1 A
10 A	± 10.1 A	± 10.2 A
20 A	± 20.2 A	± 20.4 A
50 A	± 50.5 A	± 51 A
Maximum power = 202 W		

Compliance limit

When sourcing voltage, the Model 2651A can be set to limit current or power. Conversely, when sourcing current, the Model 2651A can be set to limit voltage or power. The Model 2651A output will not exceed the compliance limit. The maximum compliance limit is the same as the maximum values listed in the following table. Note that the compliance value will limit 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 unless the instrument is in sink mode. The maximum compliance limits are based on source range. For more information, see the Compliance principles (on page 4-2) topic.

The instrument's compliance limit operation changes dependent on the source mode (current or voltage), load, and the configured limits (current, voltage, and power). It is important to distinguish both the current and voltage limits from the power limit. 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 compliance 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 the Operating boundaries (on page 4-5) topic.

NOTE

The only exception to the compliance limit not being exceeded is the VLIMIT when operating as an ISOURCE. To avoid excessive (and potentially destructive) currents from flowing, the VLIMIT will source or sink up to 160 mA for ISOURCE ranges on or below 100 mA. For the ranges 1 A and 5 A, the maximum current is 5.4 A. For ranges 10 A and above, the maximum current allowed is the current source setting.

Maximum compliance values for Model 2651A

Source range	Maximum compliance value
100 mV	20 A (50 A pulse)
1 V	20 A (50 A pulse)
10 V	20 A (50 A pulse)
20 V	10 A (50 A pulse)
40 V	5 A (50 A pulse)
100 nA	40 V
1 µA	40 V
10 µA	40 V
100 µA	40 V
1 mA	40 V
10 mA	40 V
100 mA	40 V
1 A	40 V
5 A	40 V
10 A	20 V
20 A	10 V
50 A (pulse)	40 V

Setting the compliance limit

Front-panel compliance limit

Set the compliance limit from the front panel as follows:

1. Press the **LIMIT** key to directly access compliance editing. Pressing the LIMIT button while in limit edit mode will toggle the display between the complimentary function limit and the power limit display.
2. Press the navigation wheel ⟳, set the compliance limit to the desired value, and then press the **ENTER** key or the navigation wheel ⟳ to complete editing.
3. Press the **EXIT (LOCAL)** key to return to the normal display.

Remote compliance limit

The table below summarizes basic commands to program the compliance limit. See Programming commands for more details on these commands. To program the compliance, simply send the command using the desired parameter. The following programming example illustrates how to set the current and voltage compliance 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 compliance state:

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

A returned value of 1 indicates that the voltage limit has been reached if the unit is configured as a current source, or that the current limit has been reached if the unit is configured as a voltage source.

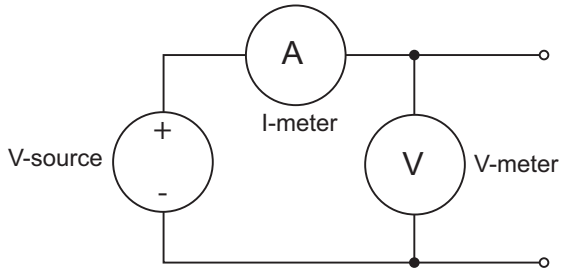
Compliance commands

Command	Description
`smua.source.limiti = limit`	Set current compliance limit.
`smua.source.limitv = limit`	Set voltage compliance limit.
`smua.source.limitp = limit`	Set power compliance limit.
`compliance = smua.source.compliance`	Test if in compliance (`true` = in compliance; `false` = not in compliance).

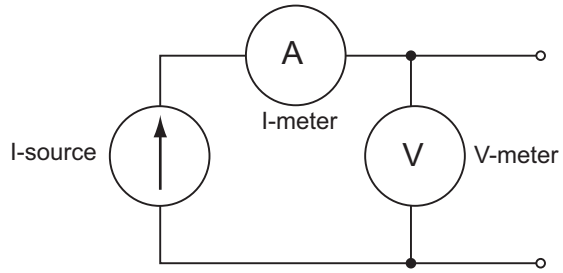
Basic circuit configurations

The fundamental source-measure configurations for the Model 2651A are shown in the figure below. When sourcing voltage, you can measure current or voltage (configuration A). When sourcing current, you can measure voltage or current (configuration B). See Basic circuit configurations (on page 4-14) detailed information on these circuit configurations.

Figure 4: Fundamental source-measure configuration



A: Source V



B: Source I

Menu overview

Menu navigation

To navigate through the menus and submenus, the Model 2651A must not be in edit mode (the EDIT indicator is not illuminated).

Selecting menu items

To navigate the Main and Configuration menus, use the editing keys as follows:

- Use the **CURSOR** arrow keys to select a menu or an option.
- Press the **ENTER** key to select an item or menu option.
- Rotate the navigation wheel ⟳ (clockwise or counter-clockwise) to select a value.
- Use the **EXIT (LOCAL)** key to cancel changes or to back out of the menu structure.

NOTE

You can use the navigation wheel ⟳ to select items from the menu or submenus.

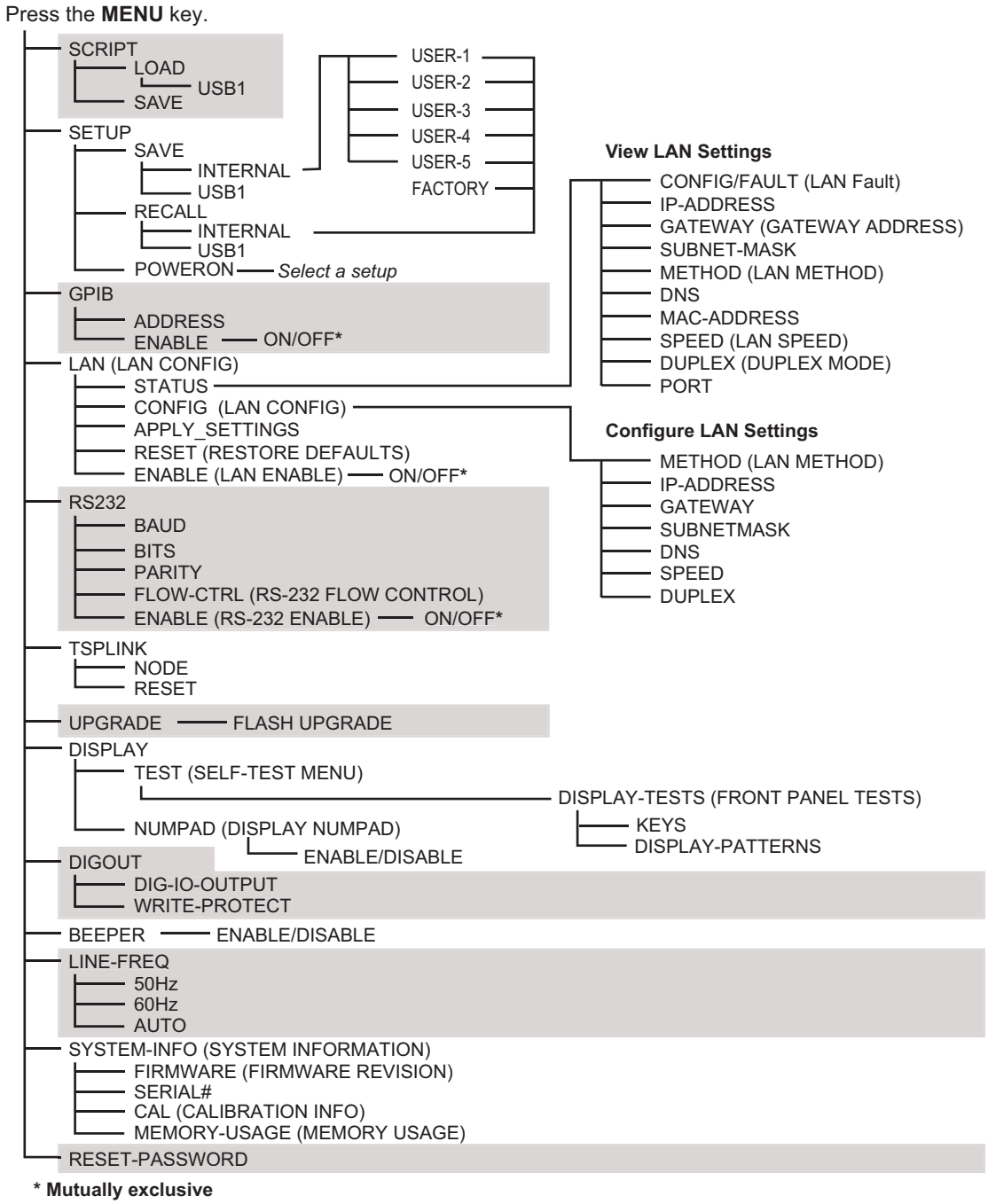
Menu trees

Many facets of operation are configured through menus. Refer to Menu navigation (on page 2-15) for more details about using menus.

Main menu

The main menu's structure is summarized in the following figure. For directions on navigating the menu, see Menu navigation (on page 2-15). For other menu items, see Configuration menus (on page 2-19).

Figure 5: Main menu tree



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

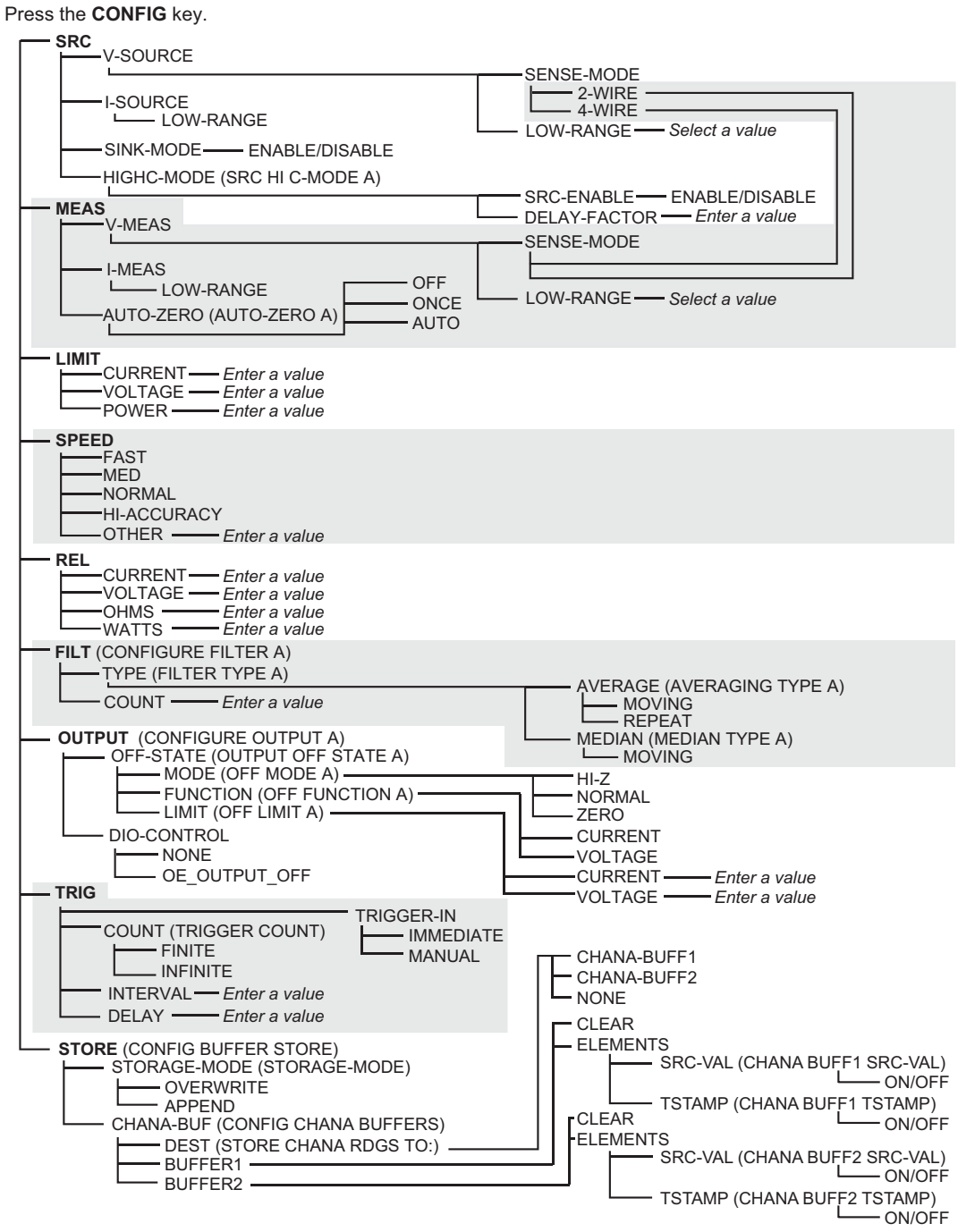
Menu selection	Description	For more information, see:
SCRIPT - LOAD - SAVE	Saves and recalls users scripts Loads scripts into nonvolatile memory Saves scripts	Manage scripts (on page 6-3)
SETUP - SAVE - RECALL - POWERON	Saves and recalls user and factory setup options Saves user setup options Recalls user setup options Sets the configuration used during startup	User setup (on page 2-39)
GPIB - ADDRESS - ENABLE	Configures the GPIB interface options Configures the address for the GPIB interface Enables and disables the GPIB interface	Communications Interfaces (on page 2-73)
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 C-1)
RS232 - BAUD - BITS - PARITY - FLOW-CTRL - ENABLE	Controls the options for the RS-232 interface Sets the baud rate Configures the number of bits Sets the parity Configures the flow control Enables and disables the RS-232 interface	Communications Interfaces (on page 2-73)
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 6-47)
UPGRADE	Upgrades the firmware from a USB flash drive	Upgrading the firmware (on page A-4)
DISPLAY - TEST - NUMPAD	Accesses display functions Runs the display test Enables and disables the numeric keypad	Front panel tests (on page A-2) See Numeric entry method in Setting a value (on page 2-20)
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 3-83)
BEEPER - ENABLE - DISABLE	Controls the key beeps Enables the key beeps Disables the key beeps	General operation (on page 2-1)

Menu selection	Description	For more information, see:
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	General operation (on page 2-1)
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 kilobytes	General operation (on page 2-1)
RESET-PASSWORD	Resets the system password	Password management

Configuration menus

The configuration menu structure is summarized in the following figure. For directions on navigating the menu, see Menu navigation (on page 2-15). For other menu items, see Main menu (on page 2-16).

Figure 6: CONFIG menu tree



The following table contains descriptions of the configuration menus, as well as cross-references to related information. To select a menu, press the **CONFIG** key and then the front-panel key associated with the desired menu (see the description column in the following table).

Quick Tip

Press the **EXIT (LOCAL)** key to return to a previous menu.

Configuration menu	To access, press the CONFIG key and then:	Description	For more information, see:
SRC	SRC	V-source sense, low range; I-source low range; and highC mode	Range (on page 2-65)
MEAS	MEAS	V and I-measure sense, low range; autozero	Range (on page 2-65)
LIMIT	LIMIT	V-source and I-source compliance limits	Compliance limit (on page 2-12)
SPEED	SPEED	Measurement speed (NPLC)	Speed (on page 2-71)
REL	REL	Set relative values	Rel (on page 3-1)
FILT	FILTER	Control digital filter	Filters (on page 3-3)
OUTPUT	OUTPUT ON/OFF	Set off-state, control digital I/O	Output-off states (on page 2-59)
TRIG	TRIG	Set trigger in, count, interval, and delay	Triggering (on page 3-32)
STORE	STORE	Set buffer count and destination	Source-measure concepts (on page 4-2)

Setting values

Setting a value

There are two ways to adjust a value: the **Navigation wheel method** or **Numeric entry method** (using the keypad). To use the keypad, the keypad feature must be enabled. When using either method, use the following editing techniques:

Quick Tip

To set a value to zero, press the **0000** numeric entry key. To toggle the polarity of a value, press the **+/–** numeric entry key.

Navigation wheel method:

1. Use the **CURSOR** arrow keys to move the cursor to the value that you want to edit.
2. Press the navigation wheel ◌ or the **ENTER** key to enter edit mode. The EDIT indicator is illuminated.
3. Rotate the navigation wheel ◌ to set the appropriate value.
4. Press the **ENTER** key to select the value or press the **EXIT (LOCAL)** key to cancel the change.
5. (Optional) Press the **EXIT (LOCAL)** key to return to the main menu.

Numeric entry method:

NOTE

The numeric entry method may only be used if the numeric keypad is enabled.

1. If the keypad is disabled, press the **MENU** key, then select **DISPLAY > NUMPAD > ENABLE**.
2. Use the **CURSOR** arrow keys to move the cursor to the value that you want to edit.
3. Press any of the desired number keys (0-9, +/-, 0000) (see 1. Special keys and power switch (on page 2-2)). The cursor moves to the next value on the right.
4. Repeat the above steps as required to set the desired values.
5. Press the **ENTER** key to select the value or press the **EXIT (LOCAL)** key to cancel change.
6. (Optional) Press the **EXIT (LOCAL)** key to return to the main menu.

Setting source and compliance values

When the Model 2651A is in the edit mode (EDIT indicator is on), the editing controls are used to set source and compliance values. Note that source autoranging will turn off when editing the source value.

To edit the source value:

1. Press the **SRC** key. The cursor flashes in the source value field.
2. Use the **CURSOR** keys or the navigation wheel ◌ to move the cursor to the desired digit.
3. Press the navigation wheel ◌ or the **ENTER** key to edit the source value. The EDIT indicator is illuminated.
4. Do one of the following to change the source value:
   - Rotate the navigation wheel ◌ to adjust the digit. The digit automatically increments or decrements the next significant digit when changing from 9 to 0 or from 0 to 9.
   - If the keypad feature is enabled, use the number keys (see "1. Special keys and power switch" on page 2-2) (0-9, +/-, 0000) to enter the source value.

NOTE

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

5. Once the desired value displays, press the **ENTER** key (the EDIT indicator is not illuminated).
6. (Optional) Press the **EXIT (LOCAL)** key to cancel source editing.

To edit compliance values:

1. Press the **LIMIT** key.
2. Use the **CURSOR** keys or the navigation wheel to move the cursor to the desired digit.
3. Press the navigation wheel or the **ENTER** key to enter edit mode. The EDIT indicator is illuminated.
4. Do one of the following to modify the compliance limit value:
   - Rotate the navigation wheel to adjust the value. The digit automatically increments or decrements the next significant digit when changing from 9 to 0 or from 0 to 9.
   - If the keypad feature is enabled, use the number keys (see 1. Special keys and power switch (on page 2-2)) to enter the value.
5. Once the desired value displays, press the **ENTER** key (the EDIT indicator is not illuminated).
6. (Optional) Press the **EXIT (LOCAL)** key to back out of the compliance menu.

NOTE

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

Operation considerations

The following paragraphs discuss autozero and NPLC caching.

Autozero

The integrating ADC of the Model 2651A uses a ratiometric A/D conversion technique. To ensure accuracy of readings, the instrument must periodically obtain fresh measurements of its internal ground and voltage reference. The time interval between needing to update these reference measurements is determined by the integration aperture being used for measurements. Separate reference and zero measurements are used for each aperture. When using the fast ADC, autozero is not needed.

There are three different settings for autozero as summarized in the table below. By default, the instrument automatically checks these reference measurements whenever a signal measurement is made (AUTO). If the reference measurements are out of date when a signal measurement is made, the instrument will automatically take 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, the OFF selection can be used to disable the automatic reference measurements. Note that with automatic reference measurements disabled, the instrument may gradually drift out of specification.

To minimize the drift, a reference and zero measurement should be made just prior to the critical test sequence. The ONCE setting can be used 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	Turns automatic reference measurements off, but immediately taking one reference and one zero measurement.
AUTO	Automatically takes new acquisitions when processor determines reference and zero values are out-of-date.

Front-panel autozero

Set the autozero from the front panel as follows:

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

Remote command autozero

Use the autozero command with the appropriate option shown in the following table to set autozero through a remote (see smuX.measure.autozero (on page 7-192) for more details). For example, send the following command to activate channel A automatic reference measurements:

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

Autozero command and options

Command	Description
`smua.measure.autozero = smua.AUTOZERO_OFF`	Disable autozero**
`smua.measure.autozero = smua.AUTOZERO_ONCE`	Force one ref and zero
`smua.measure.autozero = smua.AUTOZERO_AUTO`	Reacquire ref and zero with measurement when needed

**Old NPLC cache values will be used when autozero is disabled (see NPLC caching (on page 2-24)).

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 are acquired and stored in the cache. If there are already ten NPLC values stored, the oldest one will be overwritten by the newest one. When autozero is off, NPLC values stored in the cache will be used regardless of how old they are. If there are no entries in the cache for the aperture being used, the unit will acquire them when the first measurement is made. NPLC caching applies when using the integrating ADC. NPLC caching does not apply when the fast ADC is used.

Basic source-measure procedure

Front-panel source-measure procedure

Use the following procedure to perform the basic source-measure operations of the Model 2651A. The following procedure assumes that the Model 2651A is already connected to the DUT as explained in DUT test connections (on page 2-42).

⚠ 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 Model 2651A while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A 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.

Step 1: Select and set source level

Perform the following steps to select the source and edit the source value:

1. Press the **SRC** key as needed to select the **V-Source** or **I-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, as indicated by the EDIT indicator.
3. Use the **RANGE** keys to select a range that will accommodate the value you want to set. See Range (on page 2-65) for more information. For best accuracy, use the lowest possible source range.
4. Enter the desired source value, then press the **ENTER** key or the navigation wheel ⟳ to complete editing.

Step 2: Set compliance limit

Perform the following steps to edit the compliance limit value:

1. Press the **LIMIT** key.
2. Move the cursor to the digit to change, then press the navigation wheel ⟳ to enter the EDIT mode, as indicated by the EDIT indicator.
3. Enter the desired limit value, then press the **ENTER** key or the navigation wheel ⟳ to complete editing.

Step 3: Select measurement function and range

Select measurement function and range as follows:

1. Select the desired measurement function by pressing **MEAS** or **MODE**.
2. Set the desired measurement range with the **RANGE** keys, or enable **AUTO RANGE**, keeping the following points in mind:
   - 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), measurement range selection can be done manually or automatically. When using manual ranging, use the lowest possible range for best accuracy. When autorange is enabled, the Model 2651A automatically goes to the most sensitive range to make the measurement.

Step 4: Turn output on

Turn the output on by pressing the **OUTPUT ON/OFF** control. The OUTPUT indicator light will turn on.

Step 5: Observe readings on the display.

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

Step 6: Turn output off

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

Remote source-measure procedure

Basic source-measurement procedures can also be performed by remote. To do this, send the appropriate commands in the right sequence. The following table summarizes basic source-measure commands. See Remote Commands (on page 5-1) for more information on using these commands.

Basic source-measure commands

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

```
print(smua.measure.i())
```

Source-measure programming example

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

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

```
-- Restore Model 2651A defaults.
smua.reset()
-- Select voltage source function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set source range to auto.
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 current reading in buffer.
print(smua.measure.i(smua.nvbuffer1))
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
-- Beep.
beeper.enable = beeper.ON
beeper.beep(1, 1200)
beeper.enable = beeper.OFF
```

Triggering in local mode

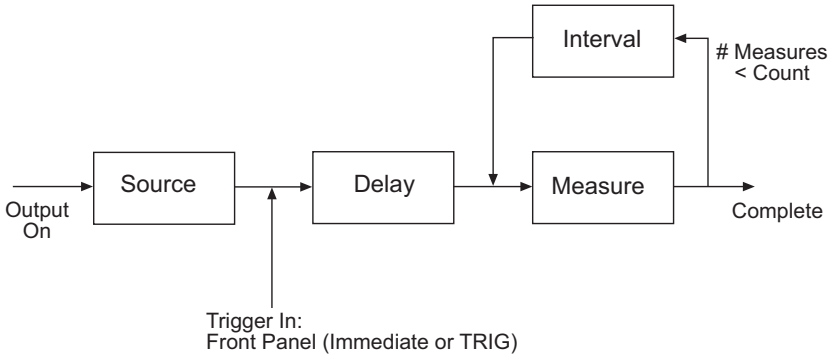
It is not necessary to change any trigger settings to use the basic source and measurement procedures covered in this section, however it is important to reset the instrument before triggering in local mode.

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 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).
- (Front panel operation only) If the immediate trigger source is selected, a measurement will be triggered immediately. However, if the manual trigger source is selected, the front panel TRIG key must be pressed.
- The unit waits for the programmed delay period (if any).
- The instrument takes one measurement.
- If the number of measurements is less than the programmed trigger count, it cycles back to take another measurement (the measurement cycle will be repeated indefinitely if the infinite trigger count is selected).
- For multiple measurements, the unit waits for the programmed trigger interval (if any) before taking the next measurement.

Figure 7: Local triggering



Configuring trigger attributes in local mode

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

TRIGGER-IN: Use these options to select the trigger-in source:

- **IMMEDIATE**: Triggering occurs immediately and the unit starts once it is ready to take measurements (for example, after the source output is turned on).
- **MANUAL**: The front panel TRIG key must be pressed to trigger the instrument to take readings.

COUNT: Sets the trigger count (number of measurements) as follows:

- **FINITE**: The unit will cycle through measurement cycles for the programmed trigger count (1 to 99999).
- **INFINITE**: The unit will cycle 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 configures 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.
7. Press **EXIT (LOCAL)** to return to normal display.
8. Press the **OUTPUT ON/OFF** control to turn the output on, and then press **TRIG**. A 2 s delay occurs before the first measurement. The unit cycles through measurements indefinitely with a 1 s interval between measurements.
9. Press the **OUTPUT ON/OFF** control again to stop taking readings.

Sink operation and interface

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and compliance levels before connecting the Model 2651A High Power System SourceMeter® Instrument to a device that can deliver energy such as other voltage sources, batteries, capacitors, solar cells, or other System SourceMeter instruments. Program recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance levels may result in damage to the instrument or to the device under test.

When operating as a sink (V and I have opposite polarity), the SourceMeter instrument is dissipating power rather than sourcing it. An external source (for example, a battery) or an energy storage device (for example, a capacitor) can force operation into the sink region.

For example, if a 12 V battery is connected to the V-Source (In/Out HI to battery high) that is programmed for +10 V, sink operation will occur in the second quadrant (Source +V and measure - I).

⚠ CAUTION

When using the I-Source as a sink, ALWAYS set V-Compliance to a level that is higher than the external voltage level. Failure to do so could result in excessive current flow into the Model 2651A (<102 mA) and incorrect measurements. See Compliance limit for details.

NOTE

The only exception to the compliance limit not being exceeded is the VLIMIT when operating as an ISOURCE. To avoid excessive (and potentially destructive) currents from flowing, the VLIMIT will source or sink up to 102 mA for ISOURCE ranges on or below 100 mA. For the ranges 1 A and above, the maximum current allowed is the current source setting.

The sink operating limits are shown in General High Power System SourceMeter® Instrument power equation in the General power equation (on page 4-3).

Configuring for measure only tests using the MODE key

In addition to being used for conventional source-measure operations, the Model 2651A can also be used like a meter to measure current, voltage, resistance, or power.

To configure the Model 2651A as a V-meter, I-meter, ohm-meter, or watt-meter:

1. Press the **MODE** key.
2. Turn the navigation wheel ◌ to select the desired meter from the menu (**I-METER**, **V-METER**, **OHM-METER**, or **WATT-METER**) and then press the **ENTER** key.
3. The Model 2651A has now been configured to the selected meter.

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

- V-meter and I-meter measurements (on page 2-30)
- Ohms measurements (on page 2-31)
- Power measurements (on page 2-36)

V-meter and I-meter measurements

To make V-meter and I-meter measurements without using the MODE key (such as when configuring measure-only tests over the remote interface), follow the basic procedure below.

Perform the following steps to use the Model 2651A to measure voltage or current:

1. Select source-measure functions.
   Voltmeter: Press the **SRC** key to select the I-source, and press the **MEAS** key to select the voltage measurement function.
   I-meter (ammeter): Press the **SRC** key to select the V-source, and press the **MEAS** key to select the current measurement function.
2. Set source and compliance levels. Use the editing procedure provided in steps 1 and 2 of the Front-panel source-measure procedure (on page 2-24) to edit the source and compliance levels:
   a. Select the lowest source range and set the source level to zero.
   b. Set compliance to a level that is higher than the expected measurement.

⚠ CAUTION

When using the Model 2651A as a voltmeter, V-compliance must be set higher than the voltage that is being measured. Failure to do this could result in excessive current flow into the Model 2651A (<100 A) and incorrect measurements.

3. Select range: Use the **RANGE** keys to select a fixed measurement range that will accommodate the expected reading. Use the lowest possible range for best accuracy. When measuring the function opposite from the source function, autorange can be used instead. The Model 2651A automatically goes to the most sensitive range.
4. Connect voltage or current to be measured. Connect the device-under-test (DUT) to the Model 2651A using 2-wire connections (see DUT test connections (on page 2-42)).
5. Press the **OUTPUT ON/OFF** control to turn the output on.
6. View the displayed reading (press the **TRIG** key if necessary). When finished, press the **OUTPUT ON/OFF** control again to turn the output off.

Ohms measurements

Ohms calculations

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 sourced current

Ohms ranging

The front panel ohms function does not use ranging. The unit formats a calculated V/I reading to best fit the display. There may be leading zeros if the ohms reading is very small (<1 mΩ).

Basic ohms measurement procedure

When using the MODE key to select ohms measurement, the Model 2651A is automatically configured as a current source with a level of 1 mA. If you wish to change the source function, source value, or compliance value (in other words, if you wish to customize the MODE key's standard ohm-meter's configuration), then perform the following steps to perform ohms measurements. The following procedure assumes that the Model 2651A is already connected to the device-under-test (DUT) as explained in DUT test connections (on page 2-42).

⚠ 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 Model 2651A while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A 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 take an ohms measurement:

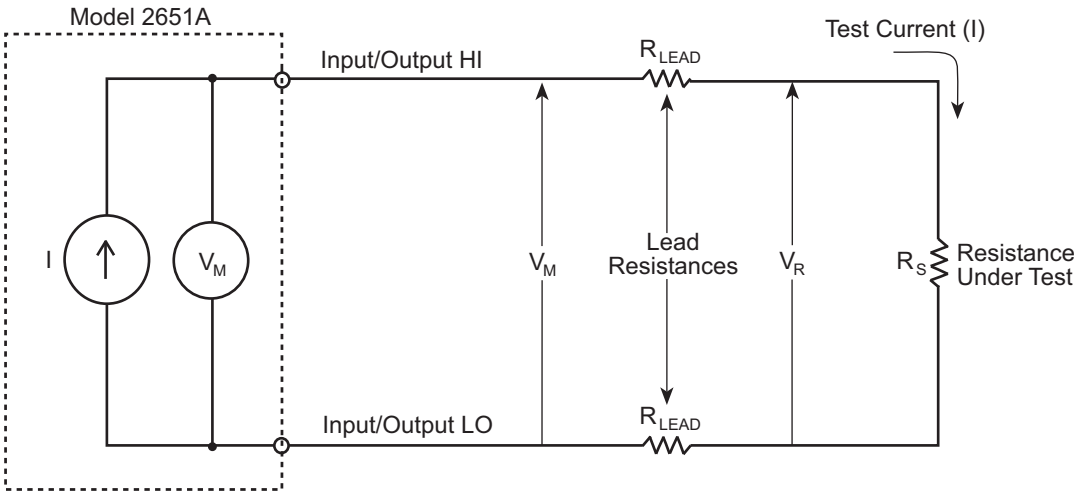
1. Press the **SRC** key to select the desired source function, and then set the output source (current or voltage, dependant on which function is selected) to the desired value based on the expected resistance. See Step 1 (see "Step 1: Select and set source level" on page 2-24) of Front-panel source-measure procedure (see "Front-panel source-measure procedure" on page 2-24) earlier in this section.
2. 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 (see "Step 2: Set compliance limit" on page 2-25) of Front panel source-measure procedure (see "Front-panel source-measure procedure" on page 2-24) earlier in this section.
3. Press the **MEAS** key to display voltage or current, then make sure that AUTO measurement range is on.
4. Press the **MEAS** key to display ohms.
5. Turn on the output, then note the reading on the display. If necessary, press the **TRIG** key to display continuous readings. Turn off the output when finished.
6. Press the **OUTPUT ON/OFF** control to turn the output on.
7. View the displayed reading (press the **TRIG** key if necessary). When finished, press the **OUTPUT ON/OFF** control again to turn the output off.

Ohms sensing

Ohms measurements can be made using either 2-wire or 4-wire sensing. See DUT test connections (on page 2-42) 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 2-wire resistance sensing, test lead resistance can seriously affect the accuracy of 2-wire resistance measurements, particularly with lower resistance values. The 4-wire sensing method shown in 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 8: 2-wire resistance sensing

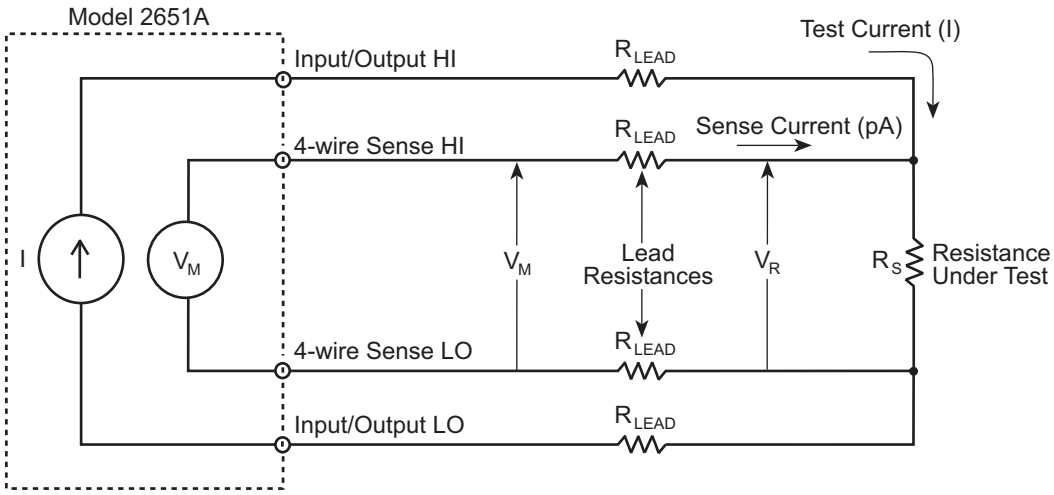


I = Current sourced
V_M = Voltage measured
V_R = Voltage across resistor

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

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

Figure 9: 4-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 = $\frac{V_M}{I} = \frac{V_R}{I} = R_S$

Sense selection

Front panel sense selection

To select sensing mode:

1. Press the **CONFIG** key, and then press the **MEAS** key. Choose **V-MEAS**, and then press the **ENTER** key or the navigation wheel ⊙.
2. Select **SENSE-MODE**, then press the **ENTER** key.
3. Select **2-WIRE** or **4-WIRE**, as desired, and then press the **ENTER** key or the navigation wheel ⊙.

Remote sense selection

Use the `smua.sense` command to control sense selection by remote. The programming example below illustrates how to enable 4-wire sensing:

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

See Remote source-measure procedure (on page 2-25) and Remote Commands (on page 5-1) for details.

Remote ohms programming

The following paragraphs summarize basic commands necessary for remote ohms programming and also give a programming example for a typical ohms measurement situation.

Remote ohms command

Use the `smua.measure.r()` function to obtain a resistance reading. The programming example below illustrates how to obtain a resistance reading from SMU A:

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

See Remote source-measure procedure (on page 2-25) for more commands necessary to set up source and measure functions, and Remote Commands (on page 5-1) 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: auto
- Voltage compliance: 10 V
- Sense mode: 4-wire

```
-- Restore Model 2651A defaults.
smua.reset()
-- Select current source function.
smua.source.func = smua.OUTPUT_DCAMPS
-- Set source range to 10 mA.
smua.source.rangei = 10e-3
-- Set current source to 10 mA.
smua.source.leveli = 10e-3
-- Set voltage limit to 10 V.
smua.source.limitv = 10
-- Enable 4-wire ohms.
smua.sense = smua.SENSE_REMOTE
-- Set voltage range to auto.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Get resistance reading.
print(smua.measure.r())
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
```

Power measurements

Power calculations

Power readings are calculated from the sourced and measured current or voltage as follows:

$P = V \times I$

Where:

P is the calculated power

V is the sourced or measured voltage

I is the measured or sourced current

Basic power measurement procedure

If you need to customize the MODE key's standard watt-meter configuration, perform the following steps to perform power measurements. The following procedure assumes that the System SourceMeter® instrument is already connected to the device-under-test (DUT) as explained in Remote Commands (on page 5-1).

⚠ 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 Model 2651A while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A 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 perform power measurements:

1. Set source function and value. Press the **SRC** key to select the voltage or current source function as required, then set the output voltage or current to the desired value. See Step 1 of Front-panel source-measure procedure (on page 2-24) earlier in this section.
2. 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 2-24) earlier in this section.
3. Press the **MEAS** key to display power.
4. Turn on the output, then note the reading on the display. If necessary, press the **TRIG** key to display continuous readings.
5. Turn off the output when finished.

Remote power programming

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

Remote power command

The programming example below illustrates how to obtain a power *reading* from smua:

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

See for more commands necessary to set up source and measure functions and also Remote Commands (on page 5-1) for more details.

Power 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, auto
- Current compliance: 50 mA

```
-- Restore Model 2651A defaults.
smua.reset()
-- Select voltage source function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set source
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 auto.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Get power reading.
print(smua.measure.p())
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
```

Contact check measurements

Overview

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

⚠ 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 Model 2651A while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A 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.

Contact check commands

The following table summarizes basic contact check commands. See Remote Commands (on page 5-1) for more information on using these commands.

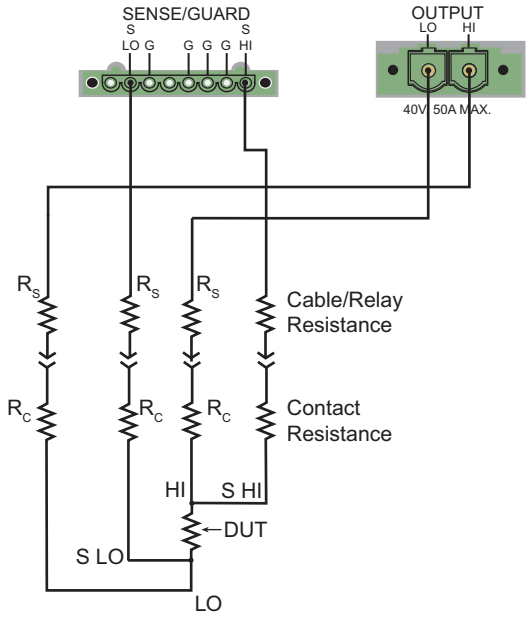
Basic contact check commands

Command	Description
`flag = smua.contact.check()`	Determine if contact resistance is lower than threshold.
`rhi, rlo = smua.contact.r()`	Measure the contact resistance.

`smua.contact.speed = speed_opt`	Set speed_opt to one of the following:
	`0` or `smua.CONTACT_FAST`
	`1` or `smua.CONTACT_MEDIUM`
	`2` or `smua.CONTACT_SLOW`

`smua.contact.threshold = rvalue`	Resistance threshold for the contact check function.

Figure 10: Contact check measurement connections



Contact check programming example

The following programming example illustrates the setup and command sequence for a typical contact measurement. These commands set the contact check speed to fast and the threshold to 10 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 speed to fast.
smua.contact.speed = smua.CONTACT_FAST
-- Set threshold to 10 ohms.
smua.contact.threshold = 10
-- Check contacts against threshold.
if not smua.contact.check() then
    -- Set speed to slow.
    smua.contact.speed = smua.CONTACT_SLOW
    -- Get 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
```

User setup

The Model 2651A can be restored 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 Model 2651A powers-up to the original default settings. The default settings are also contained in the five user setup locations but may be overwritten. The original default settings are listed in the Instrument Command Library found in Remote commands (on page 5-1). The instrument will always start-up loading the power-on setup.

Saving user setups

You can save the present Model 2651A setup to internal nonvolatile memory or a USB flash drive.

To save a user setup to nonvolatile memory:

1. Configure the Model 2651A for the desired operating modes to be saved.
2. Press the **MENU** key, select **SETUP**, and then press the **ENTER** key.
3. Select the **SAVE** menu item, and then press the **ENTER** key.
4. Select **INTERNAL**, then press the **ENTER** key.
5. Select the user number (1 through 5), and press the **ENTER** key.

To save a user setup to an external USB flash drive:

1. Configure the Model 2651A for the desired operating modes to be saved.
2. Insert the USB flash drive into the USB port on the front panel of the Model 2651A.
3. Press the **MENU** key, then select **SETUP**, then press the **ENTER** key.
4. Select **SAVE** menu item, then press the **ENTER** key.
5. Select **USB1**. The file `setup000.set` is displayed.
6. Turn the navigation wheel ⟳ to change the last three digits of the file name and then press the **ENTER** key.

Recalling a saved setup

You can recall setups from internal nonvolatile memory or a USB flash drive at any time. To recall a saved setup:

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. (INTERNAL only) Do one of the following:
   - Select the user number (1 through 5), then press the **ENTER** key.
   - Select **FACTORY** to restore factory defaults, then press the **ENTER** key.
6. (USB1 only) Select the appropriate file and then press the **ENTER** key.

Start-up configuration

You can specify the Model 2651A start-up (power-on) configuration from the front panel. Set the start-up configuration to a previously stored setup (recalled from internal nonvolatile memory) or reset to the factory default setup. To select the power-on setup:

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. Do one of the following:
   - Select **FACTORY** to load the original defaults.
   - Select **USER NUMBER** (1 through 5) to load a user preference.
5. Press the **ENTER** key.
6. Press the **EXIT (LOCAL)** key to return to the main menu.

Saving user setups from a command interface

Saving and recalling user setups

Use the `setup.save()` and `setup.recall()` functions to save and recall user setups. The following programming example illustrates how to save the present setup as setup 1, and then recall setup 1:

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

Restoring the factory default setups

Use one of the reset functions to return the Model 2651A to the original factory defaults:

```
-- 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 just the local node.
localnode.reset()
```

Start-up (power-on) configuration

You can specify the Model 2651A start-up (power-on) configuration. Use the `setup.poweron` attribute to select which setup to return to upon power-up. To set the `poweron` attribute:

```
setup.poweron = n       -- Select power-on setup.
```

Where:

n = `0` (`*RST`/`reset()` factory defaults)

n = `1` to `5` (user setups 1-5)

DUT test connections

Input/output connectors

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument uses screw terminal connectors for input and output connections to devices under test (DUTs) as shown below.

A connector can be removed from the rear panel by loosening the two captive retaining screws and pulling it off the rear panel (on page 2-5). Each screw in the terminal connector cable assembly (used with the terminal connector) can accommodate from 24 AWG (0.2 mm^2) to 12 AWG (2.5 mm^2) conductors. The screws in the output connector cable assembly (used with the output connector) requires use of 6 AWG (16 mm^2) conductors.

Basic connection sequence:

1. With the connector uninstalled from the Model 2651A rear panel, make the wire connections from a connector to the DUT.
2. Reinstall the connector onto the rear panel.
3. Tighten the two captive screws.

⚠ WARNING

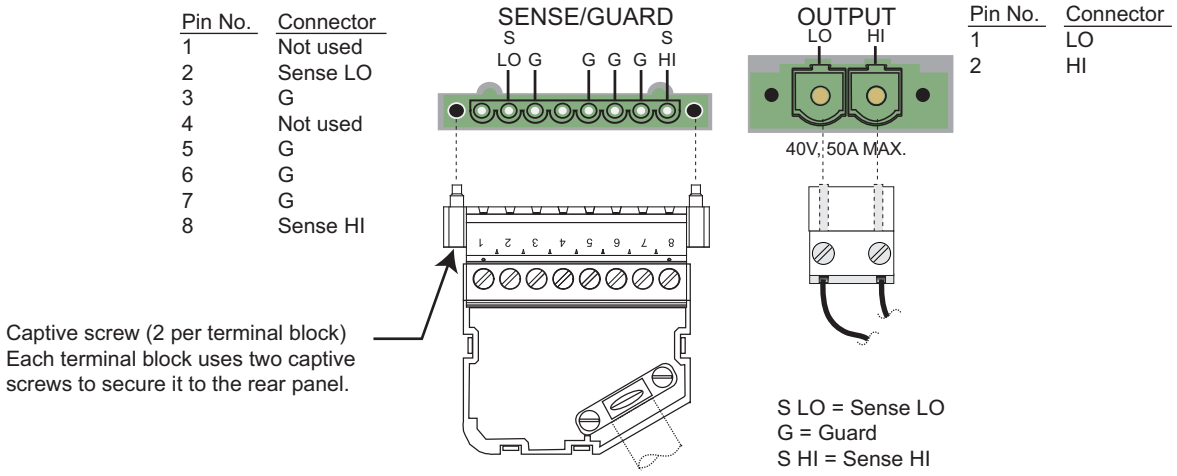
Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, NEVER make or break connections to the Model 2651A while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of a High Power 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 2-58) later in this section for details on floating the SMUs.

The input/output connectors of the High Power System SourceMeter® Instrument are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500 V peak. Do not connect the Model 2651A 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 and/or damage to the High Power System SourceMeter® Instrument, when connecting to a source with a greater current capability than the Model 2651A, a user-supplied fuse, rated at no more than 20 A SLO-BLO, should be installed in-line with the Model 2651A input/output connectors.

Figure 11: Model 2651A input/output connectors



Input/output LO and chassis ground

SMU input/output LO is available at the rear panel terminal block (labeled OUTPUT). Input/output LO is electrically isolated from chassis ground. See 7. Chassis ground (on page 2-6) for location of chassis ground connection.

NOTE

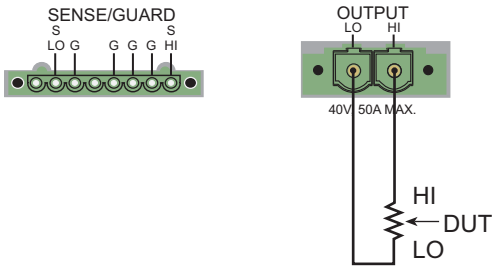
Using the chassis as a ground point for signal connections to the Model 2651A chassis may result in higher or lower noise. The tie point to the chassis can help quiet measurements if the Model 2651A common-mode current is channeled to the chassis instead of the device. However, if other equipment is also connected to the chassis, higher noise (due to the other connected equipment) may result when using the chassis as a ground point. If you choose to use the chassis as a ground point for signal connections, use the Model 2651A chassis screw as a connection point.

2-wire local sensing

Two-wire local sensing measurements (connected as shown in the following figure) can be used for the following source-measure conditions:

- Sourcing and measuring current.
- Sourcing and/or measuring voltage in high impedance (>1 kΩ) test circuits.

Figure 12: Two-wire resistance connections



4-wire remote sensing

When sourcing and/or measuring voltage in a low-impedance test circuit (see the figure below), there can be errors associated with IR drops in the test leads. 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.

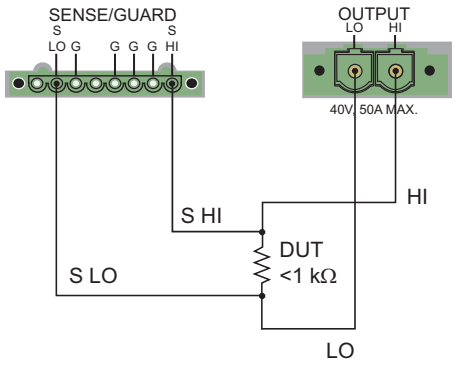
Use 4-wire remote sensing for the following source-measure conditions:

- Sourcing and/or measuring voltage in low impedance (<1 kΩ) test circuits.
- Enforce voltage compliance limit directly at the DUT.

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 Model 2651A will 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 2-37) for more details.

Figure 13: Model 2651A four-wire connections (remote sensing)



Sense mode selection

The sense mode can be set for 2-wire local or 4-wire remote connections.

NOTE

The default sense setting is 2-wire local.

Front-panel sense mode selection

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

1. Press the **CONFIG** key.
2. Press the **SRC** or **MEAS** key.*
3. If you pressed the SRC key:
   - Select **V-SOURCE** > **SENSE-MODE**.

   If you pressed the MEAS key:
   - Select **V-MEAS** > **SENSE-MODE**.
4. Select **2-WIRE** or **4-WIRE** as desired.

* The Model 2651A sense mode can be accessed and set from either the V-SOURCE or the V-MEAS menu items.

Remote programming sense selection

The following table summarizes the commands to select the sense mode. See Programming commands for details on using these commands.

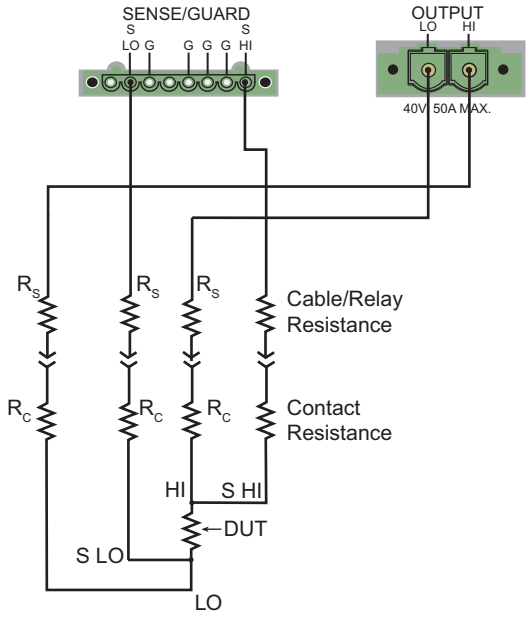
Commands to select sense mode

Command	Description
`smua.source.output = smua.OUTPUT_OFF`	Turns off the source-measure unit (SMU) output.
`smua.sense = smua.SENSE_LOCAL`	Selects local (2-wire) sense.
`smua.sense = smua.SENSE_REMOTE`	Selects remote (4-wire) sense.

Contact check connections

The contact check function prevents measurement errors due to excessive resistance in the force or sense leads. The figure below shows connections for contact check measurements. See Contact check measurements (on page 2-37) for operation and Programming commands for details on contact check commands.

Figure 14: Contact check measurement connections



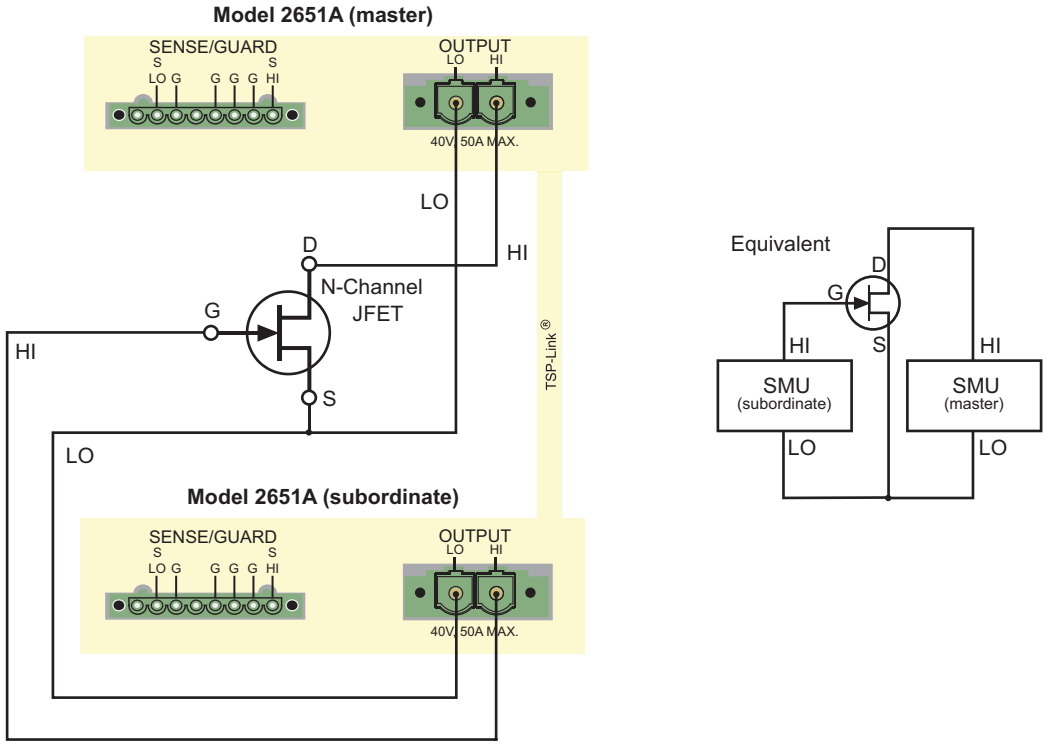
Multiple SMU connections

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and compliance levels before connecting the Model 2651A High Power System SourceMeter® Instrument to a device that can deliver energy such as other voltage sources, batteries, capacitors, solar cells, or other System SourceMeter instruments. Program recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance levels may result in damage to the instrument or to the device under test.

The figure below shows how to use the SMUs of two Model 2651A instruments to test a 3-terminal device, such as an N-channel JFET (see TSP advanced features (on page 6-53) for information on using multiple Model 2651A instruments). A typical application is for the subordinate Model 2651A to source a range of gate voltages, while the master Model 2651A sources voltage to power the device and measures current at each gate voltage.

Figure 15: Two SMUs connected to a 3-terminal device (local sensing)



The outputs of multiple SMUs can also be combined to obtain higher current and voltage levels (pulse only operation). For information including instrument configuration considerations and cautions, refer to Combining SMU outputs (on page 2-48).

Combining SMU outputs

The following information provides cautions and important considerations that need to be observed when combining SMU output channels.

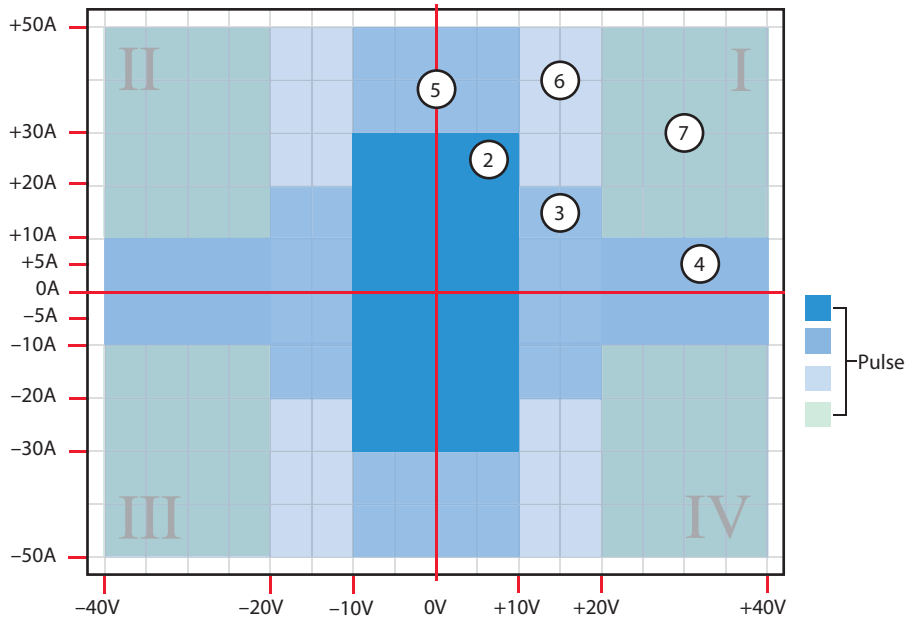
Use care when combining SMU channels. Whenever SMU channels are combined, it is best to use instruments with identical current and voltage envelopes/ranges.

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and compliance levels before connecting the Model 2651A High Power System SourceMeter® Instrument to a device that can deliver energy such as other voltage sources, batteries, capacitors, solar cells, or other System SourceMeter instruments. Program recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance levels may result in damage to the instrument or to the device under test.

When combining the outputs of two Model 2651A High Power System SourceMeter® Instruments, restrict operation to pulse only for all operating areas (both the standard and the extended operating areas). The following figure and table illustrate the pulse regions for each SMU when used in combination with another SMU. Refer to the Model 2651A specifications on the Keithley Instruments website (http://www.keithley.com) for the latest pulse width and duty cycle information. Measurements are given priority over source and display operations, so make sure that the measurement time does not exceed the allowable pulse width and duty cycle in a particular pulse region.

Figure 16: SMU pulse regions when combining SMUs

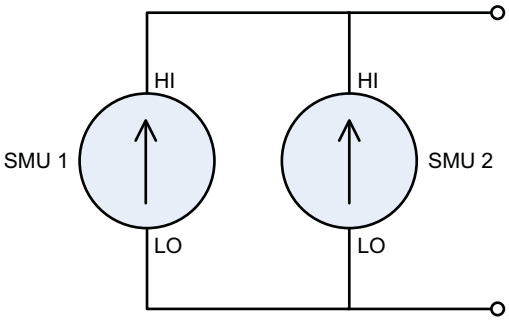


Pulse region specification

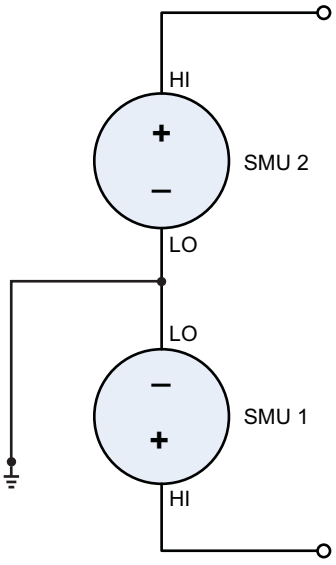
Region (quadrant diagram)	Region maximum	Maximum pulse width	Maximum duty cycle
2	30 A at 10 V	1 ms	50%
3	20 A at 20 V	1.5 ms	40%
4	10 A at 40 V	1.5 ms	40%
5	50 A at 10 V	1 ms	35%
6	50 A at 20 V	330 µs	10%
7	50 A at 40 V	300 µs	1%

Configuration guidelines are presented in the following table. Additional information, including examples on combining SMU channels, can be found in application notes on the Keithley Instruments website (http://www.keithley.com).

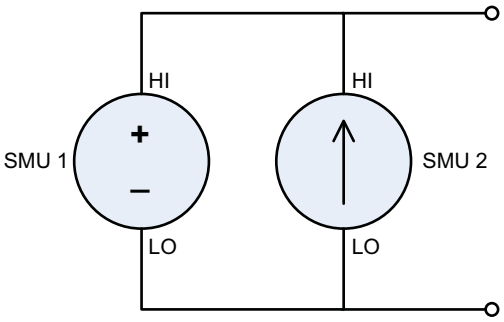
SourceMeter instrument configuration	Guidelines

Source current using parallel SMUs
 
	Maximum Pulsed Signal Levels for Model 2651A:
	100 A with 36 V compliance
	
	SMU 1 configuration:
	Output-off mode:
	`smuX.source.offmode = smuX.OUTPUT_NORMAL`
	Output-off function:
	`smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`
	Current limit for normal output-off mode (this is the maximum current that will flow between the two SMUs when the output is off):
	`smuX.source.offlimiti = 1e-3` (default)
	Voltage compliance must be 10% lower than the voltage compliance of SMU 2. Therefore this SMU will control the maximum voltage across the DUT. Voltage compliance limit (maximum):
	`smuX.trigger.source.limitv = 36`
	
	SMU 2 configuration:
	Output-off mode:
	`smuX.source.offmode = smuX.OUTPUT_NORMAL`
	Output-off function:
	`smuX.source.offfunc = smuX.OUTPUT_DCAMPS`
	Normal output-off voltage limit:
	`smuX.source.offlimitv = 40` (default)
	Voltage compliance must be 10% higher than the voltage compliance of SMU 1. Voltage compliance limit (maximum):
	`smuX.trigger.source.limitv = 40`

SourceMeter instrument configuration	Guidelines

Source voltage using series SMUs
 
	Maximum Pulsed Signal Levels for Model 2651A:
	80 V with 45 A compliance
	
	SMU 1 configuration:
	Output-off mode:
	`smuX.source.offmode = smuX.OUTPUT_NORMAL`
	Output-off function:
	`smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`
	Normal output-off current limit:
	`smuX.source.offlimiti = 1e-3` (default)
	Current compliance must be 10% higher than the current compliance of SMU 2. Current compliance limit (maximum):
	`smuX.trigger.source.limiti = 50`
	The polarity of SMU 1 is generally the opposite of the desired voltage polarity across the device. In order to achieve a positive voltage across the device, program SMU 1 to a negative voltage level. For example, to output 80 V across the device, program SMU 1 to -40 V and SMU 2 to +40 V. In order to achieve a negative voltage across the device, program SMU 1 to a positive voltage level and SMU 2 to a negative voltage level.
	Source polarity changes incur a 100 µs penalty. The number 0 is considered a positive value. For negative-going pulses with a 0 V bias level, avoid the time penalty by programming a negative number very near zero, for example, -1e-12 V.
	
	SMU 2 configuration:
	Output-off mode:
	`smuX.source.offmode = smuX.OUTPUT_NORMAL`
	Output-off function:
	`smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`
	Normal output-off current limit (0.9 mA which is 10% less than the SMU 1 `smuX.source.offlimiti`):
	`smuX.source.offlimiti = 0.9e-3`
	Current compliance must be 10% lower than the current compliance of SMU 1.Therefore this SMU will control the maximum current through the DUT. Current compliance limit (maximum):
	`smuX.trigger.source.limiti = 45`

SourceMeter instrument configuration	Guidelines

Source voltage with extended current using parallel SMUs
 
 	
	Maximum Pulsed Signal Levels for Model 2651A:
	36 V with 95 A compliance
	
	SMU 1 configuration:
	Output-off mode:
	`smuX.source.offmode = smuX.OUTPUT_NORMAL`
	Output-off function:
	`smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`
	Normal output-off current limit:
	`smuX.source.offlimiti = 1e-3` (default)
	The current compliance of SMU 1 (V-source) must be 10% greater than that maximum programmed current of SMU 2 (I-source)
	Current compliance limit (maximum):
	`smuX.trigger.source.limiti = 50`
	
	SMU 2 configuration:
	Output-off mode:
	`smuX.source.offmode = smuX.OUTPUT_NORMAL`
	Output-off function:
	`smuX.source.offfunc = smuX.OUTPUT_DCAMPS`
	Normal output-off voltage limit (40 V maximum):
	`smuX.source.offlimitv = 40`
	The voltage compliance of SMU 2 (I-source) must be 10% greater than the maximum programmed voltage of SMU 1 (V-source).
	Voltage compliance limit (maximum):
	`smuX.trigger.source.limitv = 40`

For all configurations	• When combining Model 2651A SMUs, restrict operation to pulse only • For comparable rise times, the source range and level of SMU 1 must match the source range and level of SMU 2 • The programmed current and voltage levels for both SMUs must fall within the same pulse region. See the figure titled "SMU pulse regions when combining SMUs" for a definition of pulse regions (earlier in this topic) • Placing two voltage sources in parallel is not recommended • Placing two current sources in series is not recommended • Think carefully about the appropriate output-off mode (`smuX.source.offmode`) and output-off function (`smuX.source.offfunc`) whenever changes are made to the source function. The alternative is to always use the high impedance output-off mode (`smuX.source.offmode = smuX.OUTPUT_HIGH_Z`).

Combining channels in series to output higher voltage

⚠ WARNING

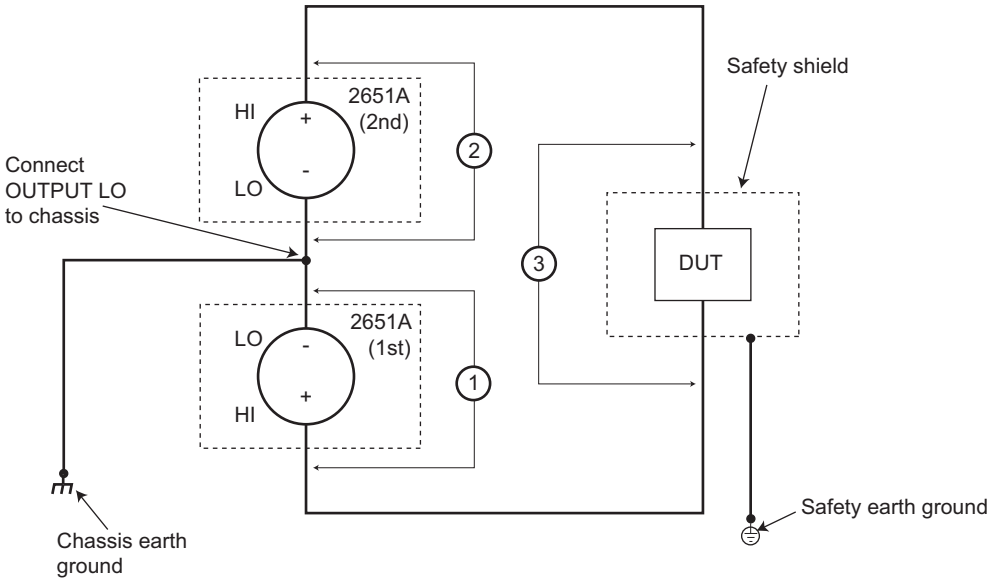
Channels in series cause hazardous voltage (>30 V RMS, 42 V peak) to be present and accessible at the Model 2651A output connector. A safety shield must be used whenever hazardous voltages will be present in the test circuit. To prevent electrical shock that could cause injury or death, NEVER use the Model 2651A in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield.

Higher pulse voltage can be output by connecting two (and only two) Model 2651A instrument's channels in series. When combining two SMU channels, make sure both SMUs have the same model number.

The following figure illustrates two Model 2651As configured with the two channels connected in series to output up to 80 V (40 V per channel).

Whenever hazardous voltage (>30 V RMS, 42 V peak) will be output, a safety shield must completely surround the DUT test circuit. When using a metal safety shield, it must be connected to a known safety earth ground and chassis ground.

Figure 17: Connecting channels in series for higher voltage



(1) First Model 2651A: SMU maximum pulse voltage: -40 V
(2) Second Model 2651A: SMU maximum pulse voltage: +40 V
(3) Series SMU maximum pulse voltage (as shown): 80 V

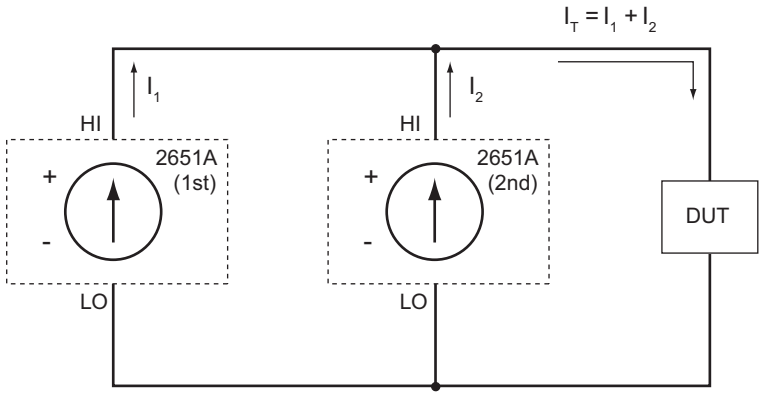
For further information, visit the Keithley Instruments website (http://www.keithley.com) for application notes on combining SMU channels.

Combining channels in parallel to output higher current

Higher pulse current can be output by connecting two (and only two) Model 2651A instrument's channels in parallel. When combining two SMU channels, make sure both SMUs have the same model number.

The figure below illustrates the connection scheme of two Model 2651A instruments connected in parallel. Two Model 2651A can output up to 100 A at 36 V (see Combining SMU outputs (on page 2-48)). The current delivered to the DUT is the sum of currents output by SMU channels (I_T). Combining the two Model 2651A instrument's expands the power envelope. For further information, visit the Keithley Instruments website (http://www.keithley.com) for application notes on combining two Model 2651A instruments' channels.

Figure 18: Connecting channels in parallel for higher current



I_1 Single SMU maximum pulse current: 50 A
I_2 Single SMU maximum pulse current: 50 A
I_T Paralleled SMU channels maximum pulse current (as shown): 100 A

Guarding and shielding

Source-measure performance and safety are optimized with the effective use of guarding and shielding (noise and safety shields).

Guarding

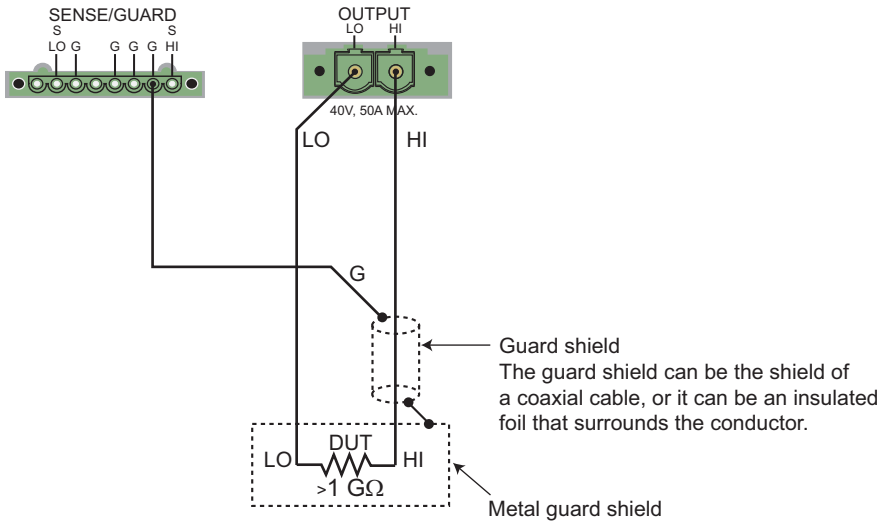
A driven guard is always enabled and provides a buffered voltage that is at the same level as the input/output HI voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between input/output high and low. Without guarding, leakage and capacitance in the external high-impedance test circuit could be high enough to adversely affect the performance of the Model 2651A.

Guarding (shown below) should be used when test circuit impedance is >1 GΩ.

NOTE

See Guard (on page 4-17) for details on the principles of guarding.

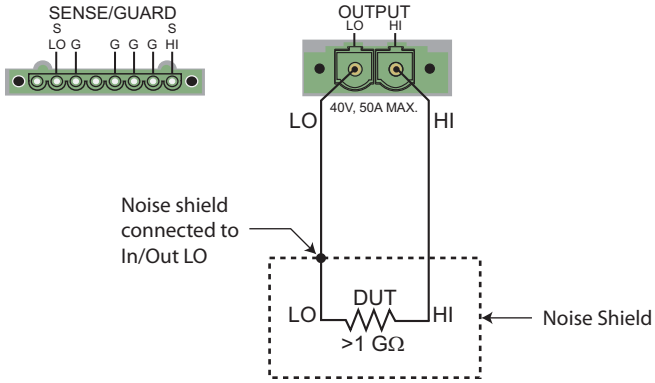
Figure 19: High-impedance guarding



Noise shield

A noise shield (see following figure) is used to prevent unwanted signals from being induced 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 SMU LO as shown.

Figure 20: Noise shield



Safety shield

⚠ WARNING

A safety shield must be used whenever hazardous voltages (>30 V RMS, 42 V peak) will be present in the test circuit. To prevent electrical shock that could cause injury or death, NEVER use the Model 2651A in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield.

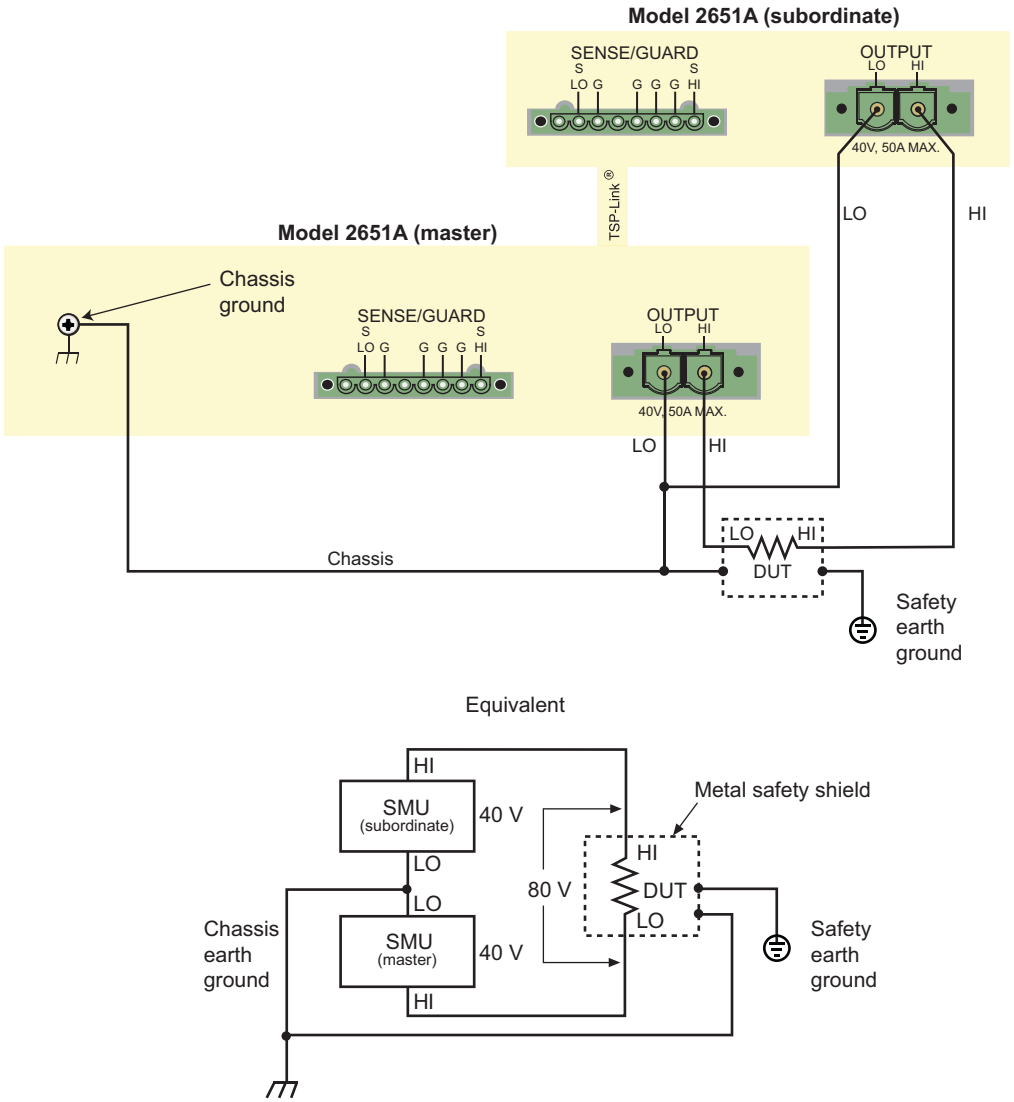
The safety shield can be metallic or nonmetallic, and must completely surround the DUT test circuit. A metal safety shield must be connected to a known safety earth ground and chassis ground. See Test fixture (on page 2-57) later in this section for important safety information on the use of a metal or nonmetallic enclosure.

Model 2651A safety shield

The maximum output voltage for a Model 2651A channel is 40 V, which is considered a nonhazardous level. However, using two Model 2651A voltage sources in a series configuration or floating a SMU (on page 2-58) can cause test circuit voltage to exceed 42 V. In the following figure, the source-measure units (SMUs) of two Model 2651A instruments can be connected in series to apply 80 V to a device under test (DUT) (see TSP advanced features (on page 6-53) for information on using multiple Model 2651A).

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

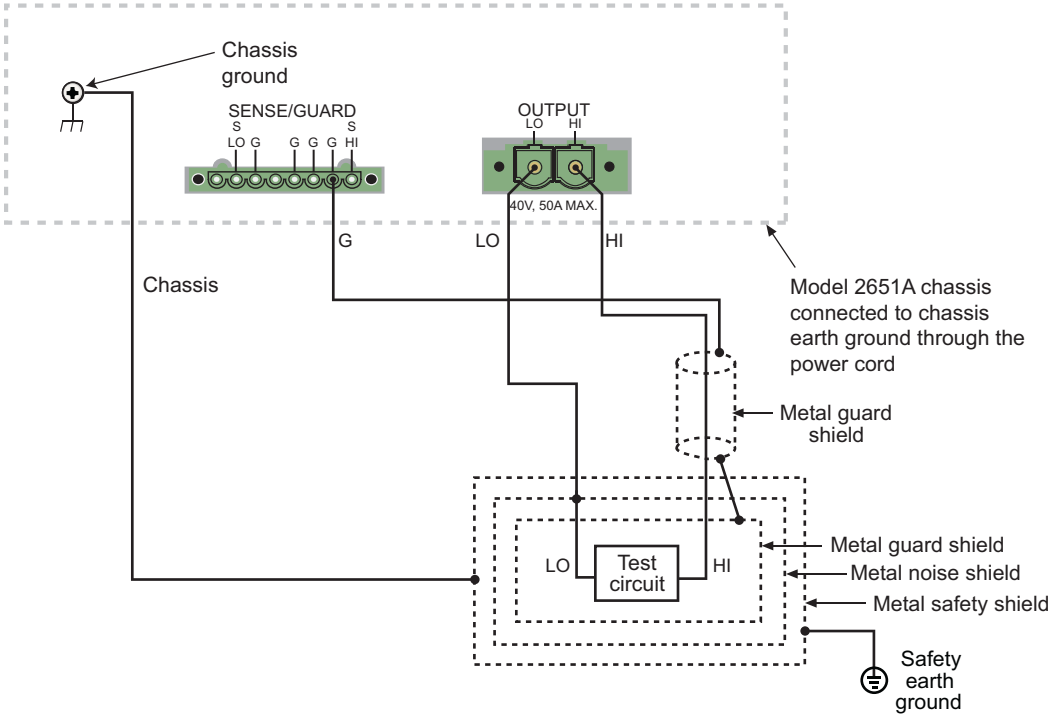
Figure 21: Safety shield for hazardous voltage using two Model 2651A instruments



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 (G) of the SMU. The noise shield is connected to SMU LO. The safety shield is connected to the chassis and to a safety earth ground.

Figure 22: Connections for noise shield, safety shield, and guarding



Test fixture

The Model 2651A digital I/O port provides an output enable line for use with a test fixture switch. When properly used, the output of the Model 2651A will turn off when the lid of the test fixture is opened.

A test fixture can be used for an external test circuit. The test fixture can be a metal or nonmetallic enclosure, and is typically equipped with a lid. The test circuit is mounted inside the test fixture. When hazardous voltages (>30 V RMS, 42 V peak) will be present, the test fixture must have the following safety requirements:

⚠ WARNING

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

Nonmetallic enclosures must be constructed of materials suitably rated for flammability and the voltage and temperature requirements of the test circuit.

For metallic enclosures, the test fixture chassis must be properly connected to safety earth 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 safety earth ground.

Construction material: A metal test fixture must be connected to a known safety Earth Ground as described in the above **WARNING**. A nonmetallic 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 nonmetallic 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 also be isolated from the test fixture. Internally, Teflon standoffs are typically used to insulate the internal pc-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.

⚠ 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 Model 2651A is not suitable for control of safety circuits and should not be used to control a safety interlock.

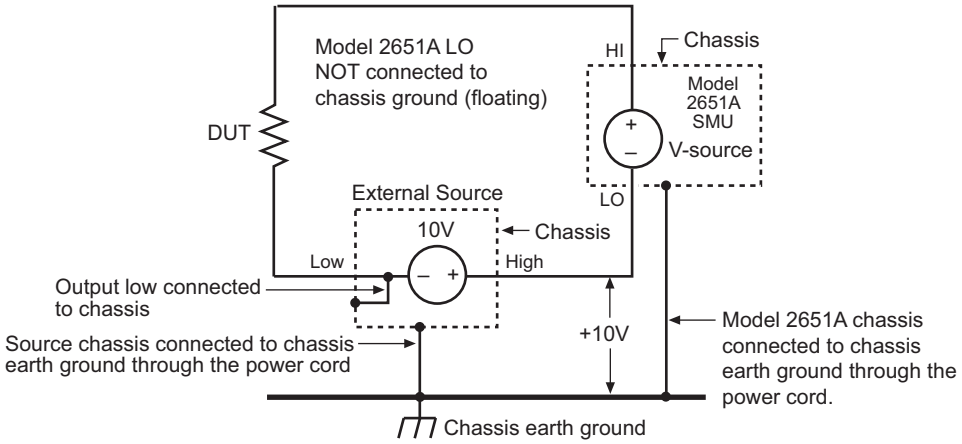
See the topic titled Digital I/O (on page 3-83) for information on the digital I/O port.

Floating a SMU

Using an external source in the test system may require that a Model 2651A 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 voltage source is connected to chassis earth ground.

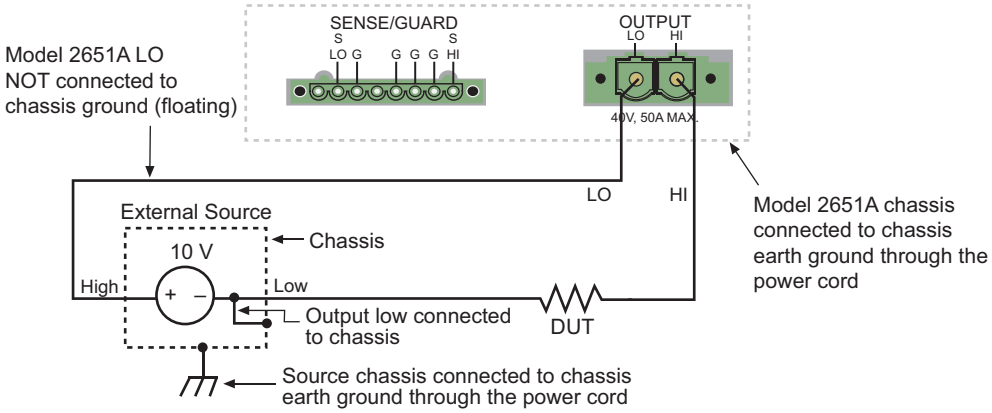
For the test circuit shown below, the Model 2651A must float off chassis earth ground. As shown, SMU LO of the Model 2651A is floating +10 V above chassis earth ground. If SMU LO of the Model 2651A was instead connected to chassis ground, the external voltage source would be shorted through chassis ground.

Figure 23: Floating the Model 2651A schematic



The Model 2651A connections for the floating configuration are shown below. In order to float the SMU, input/output LO must be isolated from chassis ground. This is accomplished by not connecting input/output LO to chassis ground.

Figure 24: Floating the 2651A SMU connections



The external voltage source can be a SMU of a second Model 2651A instrument or other instrument. Keep in mind that if the combined outputs of the sources exceeds 42 V, then a safety shield will be required for the DUT (see the following WARNINGS).

⚠ WARNING

The maximum floating (common mode) voltage for a 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 safety earth ground and chassis ground (see Safety shield (on page 2-54)).

Output-off states

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and compliance levels before connecting the Model 2651A High Power System SourceMeter® Instrument to a device that can deliver energy such as other voltage sources, batteries, capacitors, solar cells, or other System SourceMeter instruments. Program recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance levels may result in damage to the instrument or to the device under test.

Output-off modes

Turning a source-measure unit (SMU) off may not completely isolate the SMU from the external circuit. The output-off mode can be used to place the Model 2651A in a known, safe, non-interactive state during idle periods, for example, when changing devices. A Model 2651A SMU can be in one of three output-off modes: Normal, high-impedance, or 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 2-61)), and you can specify current and voltage output-off limits (see Output-off limits (on page 2-62)).

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 state for tests that turn the output off and on frequently.

Zero output-off mode

The Model 2651A is configured as described below when it is in the ZERO output-off mode.

When the V-Source is the selected source:

- The programmed V-Source value remains on the display.
- Internally, the V-Source is set to 0 V.
- The current compliance setting remains the same as the output-on value. Real compliance detection remains active.
- Measurements are performed and displayed.

When the I-Source is the selected source:

- The programmed I-Source value remains on the display.
- Internally, the V-Source is selected and set to 0 V.
- Current compliance is set to the programmed Source I value or to 10 percent full-scale of the present current range, whichever is greater.
- Measurements are performed and displayed.

The Model 2651A can be used as an I-Meter when it is in ZERO output-off mode because it will output 0 V, but measure current.

Selecting the output-off mode

Setting the output-off mode from the front panel

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

1. Press the **CONFIG** key.
2. Press the **OUTPUT ON/OFF** control. The CONFIGURE OUTPUT A menu displays.
3. In the CONFIGURE OUTPUT A menu, select **OFF-STATE** to display the OUTPUT OFF-STATE A menu.
4. With the OUTPUT OFF STATE A menu displayed, select **MODE** to open the OFF MODE A menu.
5. Select the desired output-off mode: **HI-Z** (high-impedance), **NORMAL**, or **ZERO**.

Remote configuration of the output-off mode

To select the normal output-off mode:

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

To select the high-impedance output-off mode:

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

To select the zero output-off mode:

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

Output-off function

This setting is used only when the when output is turned off and the Model 2651A is in NORMAL output-off mode (`smuX.source.offmode = smuX.OUTPUT_NORMAL`).

When the Model 2651A is in NORMAL output-off mode, you can set the output-off function to CURRENT or VOLTAGE through the CONFIG menu on the 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 (`smua.source.offfunc = smua.OUTPUT_DCVOLTS`):

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

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

- The SMU sources 0 A.
- The voltage limit is set by the `smua.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 not significant.

Selecting the output-off function

NOTE

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

Setting the output-off function from the front panel

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

1. Press the **CONFIG** key.
2. Press the **OUTPUT ON/OFF** control. The CONFIGURE OUTPUT A menu displays.
3. In the CONFIGURE OUTPUT A menu, select **OFF-STATE** to display the OUTPUT OFF STATE A menu.
4. With the OUTPUT OFF STATE A menu displayed, select **FUNCTION** to display the OFF FUNCTION A menu.
5. In the OFF FUNCTION A menu, select **CURRENT** or **VOLTAGE**.

Remote configuration of the output-off function

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

```
smua.source.offfunc = smua.OUTPUT_DCVOLTS
```

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

```
smua.source.offfunc = smua.OUTPUT_DCAMPS
```

Output-off limits

You can set output-off limits for the current and voltage output-off functions using the CONFIG menu on the Model 2651A front panel, or by using the `smu`*`X`*`.source.offlimit`*`Y`* attribute from a remote interface. This setting controls only output-off limits.

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

Setting output-off limits

Setting output-off limits from the front panel

To configure output-off limits from the front panel:

1. Press the **CONFIG** key.
2. Press the **OUTPUT ON/OFF** control. The CONFIGURE OUTPUT A menu displays.
3. In the CONFIGURE OUTPUT A menu, select **OFF-STATE** to display the OUTPUT OFF STATE A menu.
4. With the OUTPUT OFF STATE A menu displayed, select **LIMIT** to display the OFF LIMIT A menu.
5. In the OFF LIMIT A menu, select **CURRENT** or **VOLTAGE**.
6. Use the left or right arrow keys or turn the navigation wheel ⟳ to select the desired current or voltage limit, and then press the **ENTER** key or the navigation wheel ⟳ to save your settings.
7. Press the **EXIT** key as needed to back up in the menu structure or return to the default display.

Remote configuration of output-off limits

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

```
smua.source.offlimiti = iValue
```

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

```
smua.source.offlimitv = vValue
```

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
`smua.source.offmode = smua.OUTPUT_NORMAL`	Selects normal output-off state.
`smua.source.offmode = smua.OUTPUT_HIGH_Z`	Selects high-impedance output-off state.
`smua.source.offmode = smua.OUTPUT_ZERO`	Selects zero output-off state.
`smua.source.offfunc = smua.OUTPUT_DCVOLTS`	Sets 0 V output with current limit set by `smua.source.offlimiti` attribute.
`smua.source.offfunc = smua.OUTPUT_DCAMPS`	Sets 0 A output with voltage limit set by `smua.source.offlimitv` attribute.
`smua.source.offlimiti = iValue`	Sets current limit in normal output-off state.
`smua.source.offlimitv = vValue`	Sets voltage limit in normal output-off state.

USB storage overview

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument includes a USB port on the front panel. To store scripts and to transfer files from the instrument to the host PC, insert a USB flash drive into the USB port.

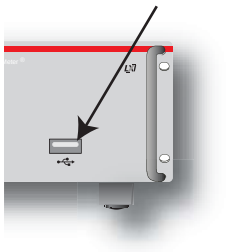
- For additional information about saving reading buffers to the USB flash drive, see Reading buffers *(on page 3-6)*.
- For additional information about storing and loading scripts to and from the USB flash drive, see Save a user script (on page 6-10).
- For additional information about file I/O, see File I/O.
- For additional information about saving user setups, see User setup (on page 2-39).

Connecting the USB flash drive

The Model 2651A supports flash drives that comply with USB 1.0 and 2.0 standards. You can save data to the USB flash drive from the 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 located on the instrument's front panel (see the figure below).

Figure 25: USB port



File system navigation

Use supported commands from the Lua `fs` library to navigate and list available files on a flash drive. The instrument encapsulates this set of commands as an `fs` logical instrument. This makes the file system of any given node available to the entire TSP-Link® system. For example, the command `node[5].fs.readdir(".")` can be used to read the contents of the current working directory on node 5.

To allow for future enhancements, the root folder of the USB flash drive has the absolute path:

```
"/usb1/"
```

NOTE

Both slash (/) and backslash (\) are supported as directory separators.

The following Lua `fs` commands, which support basic navigation and directory listing, are included for your reference:

```
fs.chdir
fs.cwd
fs.is_dir
fs.is_file
fs.mkdir
fs.readdir
fs.rmdir
```

The following Lua `fs` commands are not supported at this time:

```
fs.chmod
fs.chown
fs.stat
```

Error and status messages

Quick Tip

During operation and programming, a number of front-panel messages are briefly displayed. Typical messages are either status or error notifications (refer to Error summary list (on page 8-2) for a complete list of these messages and their meanings).

Status and error messages are held in queues. For information about retrieving error messages from queues, refer to Status Model (on page E-1).

Range

The selected measurement range affects the accuracy of the measurements as well as the maximum signal that can be measured. Note that dashed lines are displayed (for example, --.---- µA) to indicate that the previous measurement is not recent. This usually happens when a change such as selecting a different range occurs.

Available ranges

The following table lists the available source and measurement ranges for the Keithley Instruments Model 2651A High Power System SourceMeter® Instrument.

Model 2651A Source and measurement ranges

Voltage ranges	Current ranges
100 mV 1 V 10 V 20 V 40 V	1 µA 10 µA 100 µA 1 mA 10 mA 100 mA 1 A 5 A* 10 A* 20 A* 50 A**
* Refer to Operational boundaries for power derating information. ** 50 A range accessible only in pulse mode.	

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, ±5.05 A is the full-scale source value for the 5 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. Note, however, that the instrument will autorange at 100 percent of the range.

Measure auto delay

The measure delay is a specific delay applied before each measurement is taken. This delay is disabled by default (measurements are taken immediately). You can either change the default delay by setting the smuX.measure.delay (on page 7-195) attribute to a specific value or to an auto delay setting (set `smuX.measure.delay = smuX.DELAY_AUTO`). If the measure delay is set to the auto delay setting, a range-dependent delay is applied each time the instrument performs a current measurement, or after changing current ranges during an autoranged measurement. The following table contains the measure auto delays associated with each current range.

Range	Measure auto delay
10 mA (and above)	0 µs (no delay)
1 mA	100 µs
100 µA	150 µs
10 µA	500 µs
1 µA	2.5 ms
100 nA	15 ms

You can increase or decrease the auto delay 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 7-196).

Ranging limitations

- **Model 2651A:** The maximum output power and source/sink limit for the Model 2651A is 202 W per channel (maximum). With the 40 V (V-Source) range selected, the highest current measurement range is 5 A. Refer to Operating boundaries (on page 4-5) for power derating information. When pulsing with the 50 A I-Source range selected, you can be at a larger voltage range than 4 V dependent on your duty cycle. See Pulse duty cycle (on page 3-27) for details.
- **Source V Measure I or Source I Measure V:** You can set source and measure ranges separately, but if both source and measure functions are the same, the measure range is locked to the source range.

Manual ranging

Use the range keys, and , to select a fixed range:

- To set the source range, press the **SRC** key, and then use the **RANGE** keys to set the range.
- To set the measure range, press the **MEAS** key, and then set the range with the **RANGE** keys.

If the instrument displays the overflow message on a particular range, select a higher range until an on-range reading is displayed. Use the lowest range possible without causing an overflow to ensure best accuracy and resolution.

Autoranging

To use automatic source ranging, 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. With autoranging selected, the instrument automatically sets the best range to source or measure the applied signal. The instrument will autorange at 100% of range.

NOTE

Source autoranging will turn off when editing the source value.

Low range limits

The low range limit sets the lowest range the Model 2651A will use when autoranging is enabled. This feature is useful for minimizing autorange settling times when numerous range changes are involved.

To individually set low range limits for Source V, Source I, Measure V, and Measure I:

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 desired setting, and then press the **ENTER** key or the navigation wheel ⟳.
5. Press the **EXIT (LOCAL)** key (as needed) to back out of the menu structure.

Range considerations

The source range and measure range settings can interact depending on the source function. Additionally, the output state (on/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, and the present voltage measurement range will be used.
		Model 2651A example:
		`smua.source.func = smua.OUTPUT_DCVOLTS`
		`smua.source.rangev = 1`
		`smua.measure.rangev = 10`
		`-- will print 1, to match source range`
		`print(smua.measure.rangev)`
		`smua.source.func = smua.OUTPUT_DCAMPS`
		`-- will print 10, the user's range`
		`print(smua.measure.rangev)`

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 four by default.
The range is changed with the output off	Hardware settings will not be updated.	Querying will return the range setting that will be used once the output is turned on.
The range is changed with the output on	The new setting will take effect immediately.	N/A
Source autoranging is enabled	The unit will not return an error until the output is turned on.	When the output is turned on, the display will show a series of question marks: `???.???`
Measure autoranging is enabled	Range is changed only when a measurement is taken.	Querying the range after the measurement is taken will return the range that the unit chose.

Range programming

Range commands

The following tables summarize commands necessary to control measure and source ranges. See Remote commands (on page 5-1) for more details about these commands.

Measure range commands*

Commands	Description
`smua.measure.autorangei = smua.AUTORANGE_ON`	Enable current measure autorange.
`smua.measure.autorangei = smua.AUTORANGE_OFF`	Disable current measure autorange.
`smua.measure.autorangev = smua.AUTORANGE_ON`	Enable voltage measure autorange.
`smua.measure.autorangev = smua.AUTORANGE_OFF`	Disable voltage measure autorange.
`smua.measure.lowrangei = lowrange`	Set lowest I measure range for autorange.
`smua.measure.lowrangev = lowrange`	Set lowest V measure range for autorange.
`smua.measure.rangei = rangeval`	Select manual current measure range.
`smua.measure.rangev = rangeval`	Select manual voltage measure range.

* See Available ranges (on page 2-65).

Source range commands*

Commands	Description
`smua.source.autorangei = smua.AUTORANGE_ON`	Enable current source autorange.
`smua.source.autorangei = smua.AUTORANGE_OFF`	Disable current source autorange.
`smua.source.autorangev = smua.AUTORANGE_ON`	Enable voltage source autorange.
`smua.source.autorangev = smua.AUTORANGE_OFF`	Disable voltage source autorange.
`smua.source.limiti = level`	Set voltage source current limit.
`smua.source.limitv = level`	Set current source voltage limit.
`smua.source.lowrangei = lowrange`	Set lowest I source range for autorange.
`smua.source.lowrangev = lowrange`	Set lowest V source range for autorange.
`smua.source.rangei = rangeval`	Select manual current source range.
`smua.source.rangev = rangeval`	Select manual voltage source range.

* See Available ranges (on page 2-65).

Range programming example

The programming example below illustrates how to control both source and measure ranges. The Model 2651A is set up as follows:

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

```
-- Restore Model 2651A defaults.
smua.reset()
-- Set V source range to auto.
smua.source.autorangev = smua.AUTORANGE_ON
-- Select 10 mA measure range.
smua.measure.rangei = 1e-2
-- Set limit level to 10 mA.
smua.source.limiti = 1e-2
```

Digits

The display resolution of the measured reading depends upon the DIGITS setting. The DIGITS setting selects display resolution for all measurement functions.

The DIGITS setting has no effect on the remote reading format. The number of displayed digits does not affect accuracy or speed. Those parameters are controlled by the SPEED setting (see Speed (on page 2-71)).

Setting display resolution

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

Remote digits programming

The following table summarizes digits commands. See Remote commands (on page 5-1) for more information.

Digits commands

Commands	Description
`display.smua.digits = display.DIGITS_4_5`	Set display to 4.5 digits.
`display.smua.digits = display.DIGITS_5_5`	Set display to 5.5 digits.
`display.smua.digits = display.DIGITS_6_5`	Set display to 6.5 digits.

Digits programming example

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

Speed

The **SPEED** key is used to set the integration time, or measurement aperture, of the A/D converter (period of time 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).

Setting speed

Speed is set from the SPEED configuration menu and is structured as follows.

Front-panel speed configuration

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

- **FAST:** Selects the fast analog-to-digital converter (ADC)
- **MED:** Sets the speed to 0.10 PLC
- **NORMAL:** Sets speed to 1.00 PLC
- **HI-ACCURACY:** Sets speed to 10.00 PLC
- **OTHER:** Used to set speed to any PLC value from 0.001 to 25

NOTE

Selecting the FAST menu item selects the fast analog to digital converter (ADC). All other menu items on the speed menu use the integrating ADC.

NOTE

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

Remote speed programming

Speed command

The following table summarizes commands to control speed. See Remote commands (on page 5-1) for more information.

Speed command

Command	Description
`smua.measure.nplc = nplc`	Sets the speed of the integrating ADC only (nplc = 0.001 to 25). *
`smua.measure.adc = adc`	Select analog to digital converter (adc = `smua.ADC_INTEGRATE` or `smua.ADC_FAST`).

* The speed setting is global and affects all measurement functions.

Speed programming example

Use the NPLC command to set the speed. The programming example below illustrates how to set the speed to 10 PLC:

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

Communications interfaces

Selecting an interface

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument supports three remote interfaces:

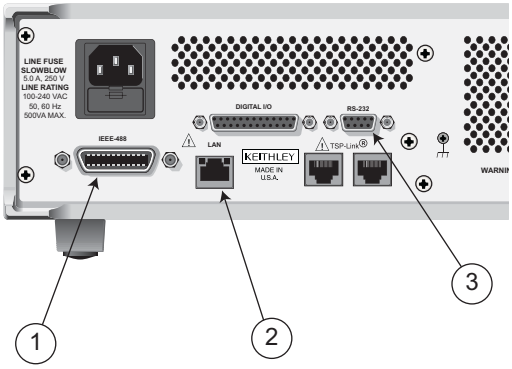
- GPIB (general purpose interface bus)
- RS-232
- LAN

NOTE

This section contains information on the GPIB connection (IEEE-488), Ethernet connection (LAN), and Serial (RS-232) communications interfaces. See LAN concepts and settings (on page C-1) for more information on LAN interfaces.

The Model 2651A can only be controlled from one remote interface at a time. The unit will remote to the first interface on which it receives a message. It will ignore the other interface until the unit is taken back to local operation.

Figure 26: Model 2651A IEEE-488, LAN, and RS-232 connections



(1) IEEE-488 connection

(2) LAN connection

(3) RS-232 connection

Output queue

All interfaces share the same output queue. The output queue sets the message available (MAV) bits in the status model. The data in the output queue clears if the mode changes to local mode.

NOTE

You must save the data from the output queue while the instrument is communicating with the remote command interface. All data in the output queue is cleared when the instrument returns to local mode.

LAN communications

Local area network (LAN) communications provide the flexibility to build scalable and functional test or data acquisition systems with a large degree of flexibility. The Model 2651A is a Class C LXI-compliant instrument that supports TCP/IP and complies with IEEE std 802.3 (Ethernet). There is one LAN port (located on the back of the instrument) that supports full connectivity on a 10 Mbps or 100 Mbps network.

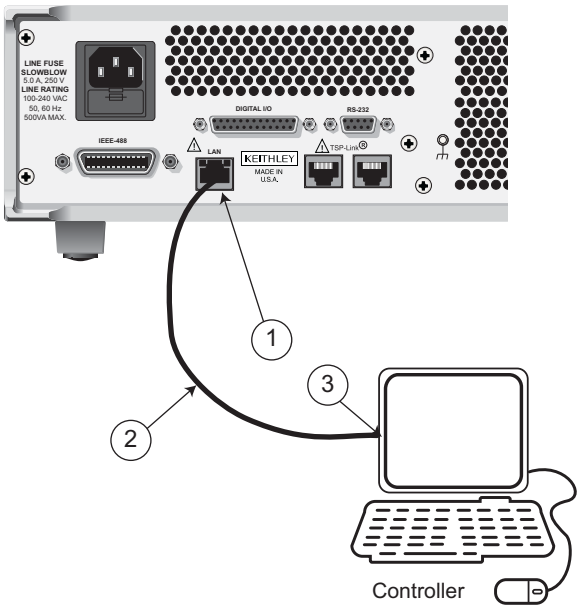
LAN cable connection

The Model 2651A includes two CA-180-3A cables (LAN crossover cables). Use one cable for the TSP-Link® network and use the other cable for the LAN.

Use the following figure as a guide when making LAN connections.

To configure the LAN settings, see LAN concepts and settings (on page C-1).

Figure 27: LAN connection



(1) Model 2651A LAN connection port

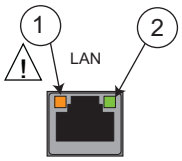
(2) Straight-through LAN cable or crossover CAT5 LAN cable (CA-180-3A)

(3) Ethernet port (located on the host computer)

LAN status LEDs

The figure below illustrates the two status light emitting diodes (LED) located at the top of the instrument's LAN connection port. The table below the figure provides explanations the LAN status LED states.

Figure 28: LAN Status



(1) LED indicates port is connected to a 100 Mbps network
(2) LED indicates port is connected to a 10 Mbps network

When the LED is:	The network:
Off	is NOT connected
On	is connected
Blinking	has traffic traversing the port

Using the LAN with remote operations

The following table lists the Model 2651A remote interface's available protocols:

LAN protocols

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

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 any given time.

Raw socket: Raw socket is a basic Ethernet connection that communicates similarly to RS-232 without explicit message boundaries. The instrument will always terminate 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.

VXI-11: VXI-11 is similar to GPIB and supports message boundaries as well as service requests (SRQs). A VXI-11 driver or VISA software is required. Test Script Builder (TSB) uses VISA and can be used with the VXI-11 interface.

Telnet: Telnet is similar to raw socket and is used when the user needs to interact directly with the instrument, typically for debugging and troubleshooting. Telnet requires a separate telnet program.

Dead socket termination port: The dead socket termination port is used to terminate all existing LAN connections. A dead socket is one which is held open by the instrument because it has not been properly closed. This most often happens when the computer is turned off or reboots without first closing the socket. This port cannot be used for command and control functions.

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.

GPIB operation

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

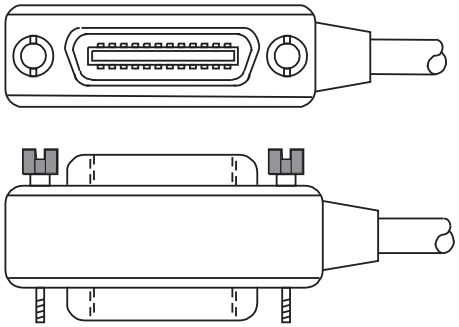
GPIB standards

The GPIB is the IEEE-488 instrumentation data bus with hardware and programming standards originally adopted by the IEEE (Institute of Electrical and Electronic Engineers) in 1975. The instrument is IEEE Std 488.1 compliant and supports IEEE Std 488.2 common commands and status model topology.

Connect the GPIB cable

To connect a to the GPIB bus, use a cable equipped with standard IEEE-488 connectors, as shown below.

Figure 29: IEEE-488 connector



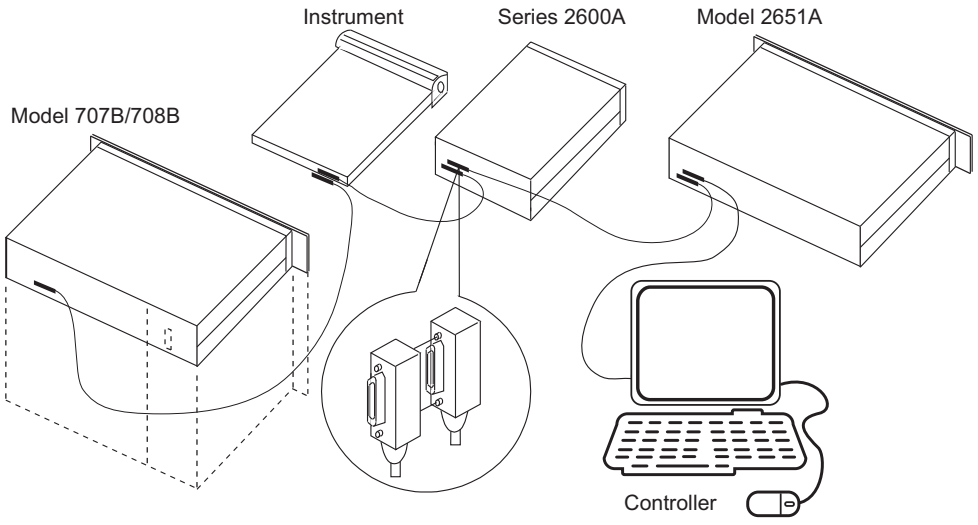
To allow many parallel connections to one instrument, stack the connectors. Two screws are located on each connector to ensure that connections remain secure. The figure below shows a typical connection scheme for a multi-unit test system.

⚠ CAUTION

To avoid possible mechanical damage, stack no more than three connectors on any one unit. To minimize interference caused by electromagnetic radiation, use only shielded IEEE-488 cables. Contact Keithley Instruments for shielded cables.

To connect the Model 2651A to the IEEE-488 bus, line up the cable connector with the connector located on the rear panel. Install and tighten the screws securely, making sure not to overtighten them (the following figure shows the location of the connections).

Figure 30: IEEE-488 connections



Connect any additional connectors from other instruments as required for your application. Make sure the other end of the cable is properly connected to the controller. You can only have 15 devices connected to an IEEE-488 bus, including the controller. The maximum cable length is either 20 meters or two meters multiplied by the number of devices, whichever is less. Not observing these limits may cause erratic bus operation.

Primary address

The Model 2651A ships from the factory with a GPIB primary address of 26. If the GPIB interface is enabled, it momentarily displays the primary address on power-up. You can set the address to a value from 0 to 30, but do not assign the same address to another device or to a controller that is on the same GPIB bus (controller addresses are usually 0 or 21).

Front panel primary address

To set or check the primary address:

1. Press the **MENU** key, then select **GPIB**, and then press the **ENTER** key or the navigation wheel ⟳.
2. Turn the navigation wheel ⟳ to set the primary address to the desired value, then press the **ENTER** key or the navigation wheel ⟳.
3. Press the **EXIT (LOCAL)** key to back out of the menu structure.

Remote primary address

Use the following command to set the primary address by remote:

```
gpib.address = address
```

To remotely set the primary address to 20, send:

```
gpib.address = 20
```

Note that changing the GPIB address takes effect when the command is processed. Any response messages generated after processing this command will be sent with the new settings. If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

Terminator

When receiving data over the GPIB, the instrument terminates messages on any line feed character or any data byte with EOI asserted (line feed with EOI asserted is also valid). When sending data, it appends a line feed character to all outgoing messages. The EOI line is asserted with the terminating line feed character.

General bus commands

General commands are commands that have the same general meaning regardless of the instrument (for example, DCL). The following table lists the general bus commands.

General bus commands

Command	Effect on Model 2651A
REN	Goes into remote when next addressed to listen.
IFC	Goes into talker and listener idle states.
LLO	**LOCAL** key locked out.
GTL	Cancel remote; restore Model 2651A front panel operation.
DCL	Returns all devices to known conditions.
SDC	Returns Model 2651A to known conditions.
GET	Initiates a trigger.
SPE, SPD	Serial polls the Model 2651A.

REN

The remote enable command (REN) is sent to the Model 2651A by the controller to set up the instrument for remote operation. Generally, the instrument should be placed in the remote mode before you attempt to program it over the bus. Setting REN true does not place the instrument in the remote state. You must address the instrument to listen after setting REN true before it goes into remote.

IFC

The interface clear command (IFC) is sent by the controller to place the Model 2651A in the local, talker, or listener idle states. The unit responds to the IFC command by cancelling front panel TALK or LSTN lights, if the instrument was previously placed in one of these states.

Transfer of command messages to the instrument and transfer of response messages from the instrument are not interrupted by IFC. If a response message was suspended by IFC, transfer of the message will resume when the unit is addressed to talk. If a command message transfer was suspended by IFC, the rest of the message can be sent when the unit is addressed to listen.

LLO

When the unit is in remote operation, all front-panel controls are disabled except the EXIT (LOCAL) key and the OUTPUT ON/OFF control (and, of course, the POWER switch). The LLO command (local lockout) disables the EXIT (LOCAL) key, but it does not affect the OUTPUT ON/OFF control, which cannot be disabled.

GTL

Use the go to local command (GTL) to put a remote-mode instrument into local mode. Leaving the remote state also restores operation of all front panel controls.

DCL (device clear)

Use the device clear command (DCL) to clear the GPIB interface and return it to a known state. Note that the DCL command is not an addressed command, so all instruments equipped to implement DCL will do so simultaneously.

When the Model 2651A receives a DCL command, it clears the Input Buffer and Output Queue, cancels deferred commands, and clears any command that prevents the processing of any other device command. A DCL does not affect instrument settings and stored data.

SDC

The selective device clear command (SDC) is an addressed command that performs essentially the same function as the DCL command. However, since each device must be individually addressed, the SDC command provides a method to clear only selected instruments instead of clearing all instruments simultaneously, as is the case with DCL.

GET

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

SPE, SPD

Use the serial polling sequence to obtain the Model 2651A serial poll byte. The serial poll byte contains important information about internal functions (see Status Model (on page E-1)). Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line. The serial polling sequence may be performed at any time to obtain the status byte from the Model 2651A.

Front-panel GPIB operation

This section describes aspects of the front panel that are part of GPIB operation, including messages, status indicators, and the LOCAL key.

Error and status messages

See Error summary list (on page 8-2) for a list of status and error messages associated with IEEE-488 programming. The instrument can be programmed to generate an SRQ, and command queries can be performed to check for specific error conditions.

GPIB status indicators

The remote (REM), talk (TALK), listen (LSTN), and service request (SRQ) indicators show the GPIB bus status. Each of these indicators is described below.

REM

This indicator shows when the instrument is in the remote state. When the instrument is in remote, all front panel keys, except for the EXIT (LOCAL) key and OUTPUT ON/OFF control, are locked out. When REM is turned off, the instrument is in the local state, and front panel operation is restored.

TALK

This indicator is on when the instrument is in the talker active state. Place the unit in the talk state by addressing it to talk with the correct talk command. TALK is off when the unit is in the talker idle state. Place the unit in the talker idle state by sending a UNT (Untalk) command, addressing it to listen, or sending the IFC (Interface Clear) command.

LSTN

This indicator is on when the Model 2651A is in the listener active state, which is activated by addressing the instrument to listen with the correct listen command. LSTN is off when the unit is in the listener idle state. Place the unit in the listener idle state by sending UNL (Unlisten), addressing it to talk, or sending IFC (Interface Clear) command over the bus.

SRQ

You can program the instrument to generate a service request (SRQ) when one or more errors or conditions occur. When this indicator is on, a service request has been generated. This indicator stays on until the serial poll byte is read or all the conditions that caused SRQ have been cleared.

LOCAL key

The EXIT (LOCAL) key cancels the remote state and restores local operation of the instrument. Pressing the EXIT (LOCAL) key also turns off the REM indicator and returns the display to normal if a user-defined message was displayed.

If the LLO (Local Lockout) command is in effect, the EXIT (LOCAL) key is also inoperative. For safety reasons, the OUTPUT ON/OFF control can be used to turn the output off while in LLO. Note that pressing the EXIT (LOCAL) key or the OUTPUT ON/OFF control will also abort any commands or scripts that are being processed.

RS-232 interface operation

Setting RS-232 interface parameters

To set interface parameters:

1. Press the **MENU** key, select **RS232** and then press the **ENTER** key or the navigation wheel ◷.
2. Select and enter the following interface parameters:
   - BAUD: Set baud rate (see Baud rate (on page 2-82))
   - BITS: Set number of bits (see Data bits and parity (on page 2-83))
   - PARITY: Set parity
   - FLOW-CTRL: Set Flow control and signal handshaking (on page 2-83)
   - ENABLE: Enable or disable the RS-232 interface
3. Press the **EXIT (LOCAL)** key as needed to back out of the menu structure.

Remote RS-232 parameters

Commands to set RS-232 parameters are listed in the following table. See Remote commands (on page 5-1) for more information.

RS-232 interface commands

Command	Description
`serial.baud = baud`	Set baud rate (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200)
`serial.databits = bits`	Set number of bits (7 or 8)
`serial.flowcontrol = flow`	Set flow control: `serial.FLOW_NONE` (no flow control) `serial.FLOW_HARDWARE` (hardware flow control)
`serial.parity = parity`	Set parity: `serial.PARITY_NONE` (no parity) `serial.PARITY_EVEN` (even parity) `serial.PARITY_ODD` (odd parity)

Changing the serial port settings takes effect when the command is processed. Any response messages generated after processing these commands will be sent with the new settings. If command messages are being queued (sent before these commands have executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting these attributes from the RS-232 interface.

RS-232 programming example

The programming example below illustrates how to set the baud rate to 9600 with no flow control:

```
serial.baud = 9600
serial.flowcontrol = serial.FLOW_NONE
```

Sending and receiving data

The RS-232 interface transfers data using 7 or 8 data bits, 1 stop bit, and no, even, or odd parity. Make sure the device you connect to the Model 2651A also uses the same settings.

Terminator

When receiving data over the RS-232 interface the command interface terminates on line feeds. A line feed is appended to all output messages when the RS-232 interface is being used as a command interface.

Sending data using the `serial.write()` function does not append a terminator. Be sure to append the appropriate terminator to the message before sending it.

Baud rate

The baud rate is the rate at which the Model 2651A and the programming terminal communicate. Select one of the following available rates:

- 115200
- 57600
- 38400
- 19200
- 9600
- 4800
- 2400
- 1200
- 600
- 300

The factory-selected baud rate is 9600.

Both the Model 2651A and the other device must be configured for the same baud rate. Make sure the device connected to the Model 2651A RS-232 port can support the selected baud rate.

Data bits and parity

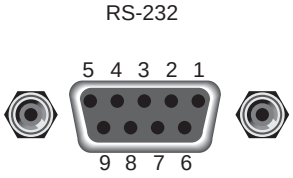
The RS-232 interface can be configured to send/receive data that is 7 or 8 bits long using even, odd, or no parity.

Flow control and signal handshaking

Signal handshaking between the controller and the instrument allows the two devices to communicate to each other regarding being ready or not ready to receive data.

The RS-232 interface provides two control lines (RTS and CTS) for this purpose (see the following figure and table). When the Model 2651A is ready to send (RTS) data, it will transmit when it receives the clear to send (CTS) signal from the computer.

Figure 31: RS-232 interface connector



RS-232 connector pinout

Pin number	Description
1	Not used
2	TXD, transmit data
3	RXD, receive data
4	Not used
5	GND, signal ground
6	Not used
7	RTS, ready to send
8	CTS, clear to send
9	Not used

To enable or disable flow control using the RS-232 configuration menu:

1. Press the **MENU** key, select **RS232 > FLOW-CTRL,** and then press the **ENTER** key or the navigation wheel ⟳.
2. Select **HARDWARE** to enable flow control or **NONE** to disable it.

RS-232 connections

The RS-232 serial port is connected to the serial port of a computer using a straight-through RS-232 cable terminated with DB-9 connectors. Do not use a null modem cable. The serial port uses the transmit (TXD), receive (RXD), CTS and RTS (if flow control is enabled), and signal ground (GND) lines of the RS-232 standard.

See Flow control and signal handshaking (on page 2-83) for the rear-panel connector for the RS-232 interface and the pinouts. The connector location is shown in Communications interfaces (on page 2-73).

If your computer uses a DB-25 connector for the RS-232 interface, you will need a standard cable or adapter with a DB-25 connector on one end and a DB-9 connector on the other. An RS-232 cable is available from the Keithley Instruments website (http://www.keithley.com).

The following table provides pinout identification for the 9-pin (DB-9) or 25-pin (DB-25) serial port connector on the computer.

Computer serial port pinout

Signal*	DB-9 pin number	DB-25 pin number
DCD, data carrier detect	1	8
RXD, receive data	2	3
TXD, transmit data	3	2
DTR, data terminal ready	4	20
GND, signal ground	5	7
DSR, data set ready	6	6
RTS, request to send	7	4
CTS, clear to send	8	5
RI, ring indicator	9	22

* The Model 2651A does not use all RS-232 signals. See the topic Flow control and signal handshaking (on page 2-83).

Section 3

Functions and features

In this section:

Rel 3-1
Filters 3-3
Reading buffers 3-6
Sweep operation 3-19
Triggering 3-32
High-capacitance mode 3-65
Display operations 3-70
Digital I/O 3-83

Rel

The rel (relative) feature can be used to set offsets to zero (0) or subtract a baseline reading from present and future readings. With rel enabled, subsequent readings will be the difference between the actual input value and the rel value, as follows:

Displayed Reading = Actual Input - Rel Value

Once a rel value is established for a measurement function, the value is the same for all ranges. For example, if 0.5 A is set as a rel value on the 1 A range, the rel value is also 0.5 A on the lower current ranges.

Selecting a range that cannot accommodate the rel 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 Model 2651A still overflows for a >1.02 A input.

NOTE

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

Front panel rel

Enabling and disabling rel

This feature can be used to reset zero (0) offsets or to establish a zero (0) baseline. To enable and use this feature, press the **REL** key. The reading (which becomes the rel value) is subtracted from itself. As a result, a zero (0) reading is displayed. Pressing the **REL** key a second time disables rel.

Defining a rel value

A unique rel value can be established for the selected measurement function.

To establish a unique rel value from the front panel:

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

Remote rel programming

Rel commands

Rel commands are summarized in the following table.

Rel commands

Command	Description
To set rel values:	
`smua.measure.rel.leveli = relval`	Set current rel value.
`smua.measure.rel.levelp = relval`	Set power rel value.
`smua.measure.rel.levelr = relval`	Set resistance rel value.
`smua.measure.rel.levelv = relval`	Set voltage rel value.
To enable/disable rel:	
`smua.measure.rel.enablei = smua.REL_OFF`	Disable current rel.
`smua.measure.rel.enablep = smua.REL_OFF`	Disable power rel.
`smua.measure.rel.enabler = smua.REL_OFF`	Disable resistance rel.
`smua.measure.rel.enablev = smua.REL_OFF`	Disable voltage rel.
`smua.measure.rel.enablei = smua.REL_ON`	Enable current rel.
`smua.measure.rel.enablep = smua.REL_ON`	Enable power rel.
`smua.measure.rel.enabler = smua.REL_ON`	Enable resistance rel.
`smua.measure.rel.enablev = smua.REL_ON`	Enable voltage rel.

Rel programming example

The programming example below illustrates how to set and enable current rel to 100 mA:

```
-- Set current rel to 100 mA.
smua.measure.rel.leveli = 0.1
-- Enable current rel.
smua.measure.rel.enablei = smua.REL_ON
```

Filters

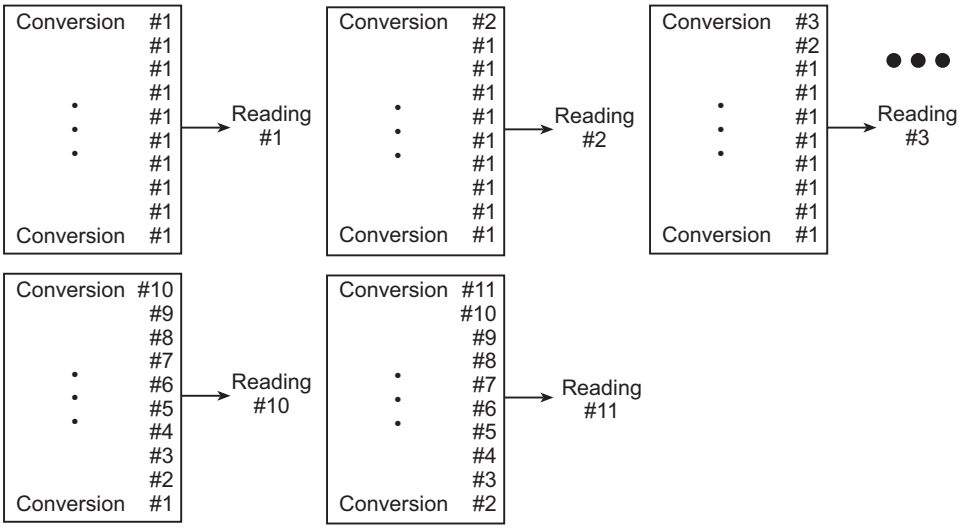
The filter feature lets you set the filter response to stabilize noisy measurements. The Model 2651A uses a digital filter, which is based on reading conversions. The displayed, stored, or transmitted reading is calculated using one or more reading conversions (from 1 to 100).

Filter types

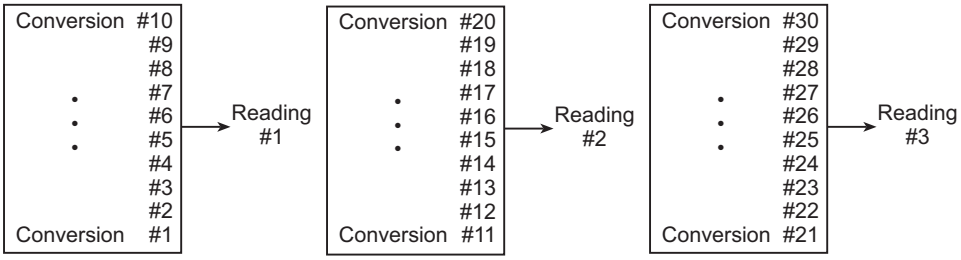
The Model 2651A has three filter types. These three filter types are broken down into two averaging filters and one median filter.

The two averaging filters are repeating and moving (see figure below). For the repeat filter (which is the power-on default), 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 32: Moving average and repeating filters



A. Type: Moving average, readings = 10



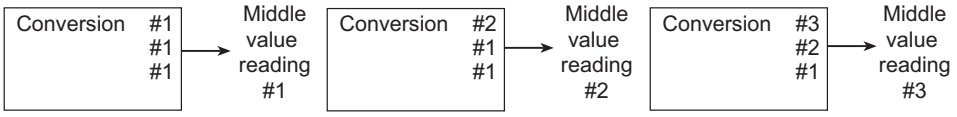
B. Type: Repeating, readings = 10

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 averaged again, yielding a new reading.

The median filter is used to pass the “middle-most” reading 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 average filter. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The median is then redetermined.

When a moving 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 in order to fill it. Thus, the first filtered reading is the same as the first reading conversion. The normal moving filter process continues. Note that a true average or median reading is not yielded until the stack is filled with new reading conversions (no copies in stack). For example, in the figure below, it takes ten filtered readings to fill the stack with new reading conversions. The first nine filtered readings are calculated using copied reading conversions.

Figure 33: Median filter



A. Type: Median, readings = 3

Front panel filter control

Enabling filter

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

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 averaging filter is much faster than the repeat averaging filter because the unit 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.

Remote filter programming

Configuring the filter

Filter type and count are configured from the filter configuration menu. 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 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 filter count (1 to 100 readings).

Filter commands

The following table summarizes filter commands. See Remote commands (on page 5-1) for details about commands.

Filter commands

Commands	Description
`smua.measure.filter.count = count`	Set filter count (1 to 100).
`smua.measure.filter.enable = smua.FILTER_ON`	Enable filter.
`smua.measure.filter.enable = smua.FILTER_OFF`	Disable filter.
`smua.measure.filter.type = smua.FILTER_MEDIAN`	Select median filter type.
`smua.measure.filter.type = smua.FILTER_MOVING_AVG`	Select moving average filter type.
`smua.measure.filter.type = smua.FILTER_REPEAT_AVG`	Select repeat average filter type.

Filter programming example

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

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

```
-- Program count to 10
smua.measure.filter.count = 10
-- Moving average filter type.
smua.measure.filter.type = smua.FILTER_MOVING_AVG
-- Enable filter.
smua.measure.filter.enable = smua.FILTER_ON
```

Reading buffers

Reading buffers capture measurements, ranges, instrument status, and output state of the Keithley Instruments Model 2651A High Power System SourceMeter® Instrument. The Model 2651A has two dedicated reading buffers. You can use the dedicated reading buffers to acquire readings, or you can use the `smuX.makebuffer()` function to create dynamic reading buffers.

Each dedicated reading buffer in the Model 2651A can store over 60,000 readings with the timestamps and source values options enabled. Disable the timestamps and source values options to store over 140,000 readings internally.

You can save the dedicated reading buffers to internal nonvolatile memory in the instrument or to the USB flash drive.

Once you save the reading buffers to the 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.

NOTE

Reading buffers (other than the dedicated reading buffers) have fixed capacity and are not specifically limited to 60,000 or 140,000 readings.

Front-panel reading buffer control

Reading buffers can be configured, stored, and recalled when in local mode operation. Use the front panel to navigate and configure the reading buffers options, as well as save and recall stored readings. For information on menu operation, see Menu overview (on page 2-15).

Configuring reading buffers

To configure reading buffers from the front panel:

NOTE

Clearing the reading buffers, and enabling or disabling the source value or the timestamp are optional.

1. Press the **CONFIG** key, and then select **STORE > CHANA-BUFF > DEST**. The destination menu is now shown:
   - CHANA-BUFF1
   - CHANA-BUFF2
   - NONE
2. Select buffer 1 or 2 (for example, select **CHANA-BUFF1** to select buffer 1).
3. Clear the buffer by turning the navigation wheel ◌ to select **CLEAR > YES**.
4. 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 enable or disable the source value or the timestamp options.

5. Configure the reading buffer's timestamp elements:
   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).
6. Configure the reading buffer's source value elements:
   a. Turn the navigation wheel ⟳ to highlight **SRC-VAL**.
   b. Press the navigation wheel ⟳ (or the **ENTER** key).
   c. Select **OFF** or **ON**.
7. Press the **EXIT (LOCAL)** key to return to the main menu.

Reading buffer options

The following listing outlines the menus and menu items associated with front panel reading buffer control. This section provides a description for the reading buffer options.

- **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 source values.
    - **TSTAMP**: Enable timestamps.
- **BUFFER2**: Configure buffer 2.
  - **CLEAR**: Clear buffer (YES or NO).
  - **ELEMENTS**: Enable (ON) or disable (OFF) data storage elements:
    - **SRC-VAL**: Enable source values.
    - **TSTAMP**: Enable timestamps.

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. Complete the steps from Saving reading buffers (on page 3-8).
2. Press the **CONFIG** key. 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 acquired during a storage operation. The reading count can range from 1 to 60,000 with timestamps and source values enabled; the count can range to over 140,000 with timestamps and source values disabled.

NOTE

To store the maximum number of readings in a reading buffer (over 140,000), disable the source values and timestamps for that reading buffer.

To specify the number of readings:

1. From the front panel, press the **STORE** key, and then select **TAKE_READINGS**.
2. Use the navigation wheel to select the number of readings.
3. Push the navigation wheel to switch to edit mode.
4. Turn the navigation wheel to change the numeric value, and then push the navigation wheel to save the numeric value.
5. Press the **ENTER** key to save the count.
6. Press the **OUTPUT ON/OFF** control to start taking readings.

NOTE

If the output-off mode is set to output zero, the instrument will immediately start acquiring data as soon as the **OUTPUT ON/OFF** control is pressed.

Saving reading buffers

You can save the dedicated reading buffers to nonvolatile memory, or you can save them to a USB flash drive. Note that the instrument will restore the dedicated reading buffers from internal nonvolatile memory when the unit is turned off and back on.

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 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**
4. The 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 the USB flash drive

After the measurements are complete, you can save the reading buffer data to a 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**. Then select **USB1**.
3. Select one of the following file formats:
   - **CSV**
   - **XML**
4. Use the navigation wheel ◌ to select the desired reading buffer.
5. Use the navigation wheel ◌ to change the file name.
6. Press the navigation wheel ◌ or the **ENTER** key to save the file.
7. Press the **EXIT (LOCAL)** key to return to the main menu.

Recalling readings

To recall the data stored in a reading buffer:

1. Press the **RECALL** key, and then select **DATA** or **STATISTICS**.
2. Select the buffer to display. The data (or statistics) will be 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 positioned at the lower left side of the display (if enabled); the timestamp (if used) is positioned at the lower right side. If statistics are being recalled, the information will include values for MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, and PK-PK. The source display field will identify the buffer: SrcA1 (buffer 1) or SrcA2 (buffer 2).

Buffer location number

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.

Timestamp

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.

Displaying other buffer readings and statistics

To display other readings and statistics in the reading buffer:

1. While still in the buffer recall mode, if viewing the data stored in the buffer, turn the navigation wheel ◌ to increment and decrement the selected digit of the location number by one. Press the navigation wheel ◌ and then turn it or use the **CURSOR** keys to move to the next digit that the navigation wheel ◌ will change. If viewing the statistics stored in the buffer, turn the navigation wheel ◌ or use the **CURSOR** keys to scroll between MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, and PK-PK.
2. To exit from the reading buffer recall mode, press the **EXIT (LOCAL)** key.

Remote reading buffer programming

Readings can be obtained in multiple ways, including synchronous or overlapped measurements. Routines that make single-point measurements can be configured to make multiple measurements where one would ordinarily be made. 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 also store the measurements in the reading buffers. Overlapped measurements always return readings in a reading buffer. Synchronous 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 accessed by ordinary array accesses. If `rb` is a reading buffer, the first measurement is accessed as `rb[1]` and the 9th measurement as `rb[9]`, and so on. 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.

Reading buffer designations

The source-measure unit (SMU) contains two dedicated reading buffers:

- `smua.nvbuffer1` (buffer 1)
- `smua.nvbuffer2` (buffer 2)

To access the reading buffer, include the name of the SMU in the attribute. For example, the following command would store readings from channel A into buffer 1:

```
smua.measure.overlappedi(smua.nvbuffer1)
```

Reading buffer commands

The following table summarizes commands associated with the reading buffers. See Remote commands (on page 5-1) for detailed reading buffer command information.

Reading buffer commands

Command	Description
Commands save/clear readings:	
`smuX.savebuffer(smuX.nvbufferY)`	Saves the reading buffer to the Model 2651A.
`smuX.nvbuffer1.clear()`	Clears buffer 1.
`smuX.nvbuffer2.clear()`	Clears buffer 2.
`mybuffer = smuX.makebuffer(n)`	Creates a dynamically allocated buffer for *n* readings.
`mybuffer = nil`	Deletes the dynamically allocated buffer.
`savebuffer(smuX.nvbuffer1,"csv", "/usb1/mybuffer.csv")`	Saves the reading buffer to the USB flash drive.
Commands to store readings:	
`smuX.measure.count = count`	The number of measurements to acquire.
`smuX.measure.overlappedi(rbuffer)`	Stores the current readings in buffer.
`smuX.measure.overlappediv(ibuffer, vbuffer)`	Stores the current and voltage readings in respective buffers (current and voltage are stored in separate buffers).
`smuX.measure.overlappedp(rbuffer)`	Stores the power readings in buffer.
`smuX.measure.overlappedr(rbuffer)`	Stores the resistance readings in buffer.
`smuX.measure.overlappedv(rbuffer)`	Stores the voltage readings in buffer.
`smuX.measure.v(rbuffer)`	Reading buffer where voltage readings will be stored.
`smuX.measure.i(rbuffer)`	Reading buffer where current readings will be stored.
`smuX.measure.iv(ibuffer, vbuffer)`	Stores the current and voltage readings in respective buffers (current and voltage are stored in separate buffers).
`smuX.measure.r(rbuffer)`	Reading buffer where resistance readings will be stored.
`smuX.measure.p(rbuffer)`	Reading buffer where power readings will be stored.
`smuX.trigger.measure.v(rbuffer)`	Reading buffer where voltage readings will be stored.
`smuX.trigger.measure.i(rbuffer)`	Reading buffer where current readings will be stored.
`smuX.trigger.measure.r(rbuffer)`	Reading buffer where resistance readings will be stored.
`smuX.trigger.measure.p(rbuffer)`	Reading buffer where power readings will be stored.
`smuX.trigger.measure.iv(ibuffer, vbuffer)`	Stores the current and voltage readings in respective buffers (current and voltage are stored in separate buffers).
Commands to access readings:	
`printbuffer(start_index, end_index, st_1, st_2, ... st_n)`	Prints data from buffer subtables: *start_index* (starting index of values to print). *end_index* (ending index of values to print). *st_1, st_2, ... st_n* (subtables from which to print each separated by a comma).

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 `smua.nvbuffer1.clear()` or `smua.nvbuffer2.clear()` command.

Buffer storage control attributes: Describes buffer storage attributes

Storage attribute	Description
`appendmode`	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 what was formerly `rb[n+1]`. This attribute is initialized to off when the buffer is created.
`cachemode`	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.
`collectsourcevalues`	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.
`collecttimestamps`	When this attribute is on, timestamps will be 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.
`fillcount`	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 `fillmode` attribute is set to `FILL_WINDOW`.
`fillmode`	The reading buffer fill mode controls how new data is added to the reading buffer. When this attribute is set to `FILL_ONCE`, the reading buffer will not overwrite readings. If the buffer fills up, new readings will be discarded. When this attribute is set to `FILL_WINDOW`, new readings will be added after existing data until the buffer holds `fillcount` elements. Once there are `fillcount` elements, new data starts overwriting data starting at index 1.
`timestampresolution`	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 very long tests. Note: The minimum resolution setting is 1 µs (0.000001 seconds).

Buffer read-only attributes

The following table contains buffer read-only attributes.

Buffer read-only attributes: Read-only attributes used to access buffer parameters

Storage attribute	Description
`basetimestamp`	The timestamp of when the reading at `rb[1]` was stored, in seconds from midnight January 1, 1970 GMT. See Time and date values (on page 7-4) 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 will be added to the reading buffer will be stored.

Buffer storage control programming examples

The programming examples below illustrate the use of buffer storage control attributes.

Buffer control programming examples:
Provides control examples for channel A, buffer 1

Command	Description
`smua.nvbuffer1.collectsourcevalues = 1`	Enable source value storage
`smua.nvbuffer1.appendmode = 1`	Enable buffer append mode
`smua.nvbuffer1.collecttimestamps = 0`	Disable timestamp storage
`smua.nvbuffer1.timestampresolution = 0.001`	Set timestamp resolution to 0.001024 s
`smua.nvbuffer1.fillcount = 50`	Set 50 as the number of readings the buffer will take before restarting at index 1.
`smua.nvbuffer1.fillmode = 0`	Set the reading buffer to fill once (do not overwrite old data).

Buffer read-only attribute programming examples

The follow programming examples illustrate use of buffer read-only attributes.

Buffer read-only attribute programming examples:
Provides read-only attribute programming examples

Command	Description
`number = smua.nvbuffer1.n`	Request the number of readings in the buffer
`buffer_size = smua.nvbuffer1.capacity`	Request buffer size

Statistic attributes

Use the `smua.buffer.getstats()` function to access the reading buffer data statistics. The table below displays the attributes that you can use to access the reading buffer data.

The returned parameter has the following attributes:

Attributes for accessing reading buffer data

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 will be `nil` because there is no data to base any statistics on. If `n` equals 1, the `stddev` attribute will be `nil` because the standard deviation of a sample size of 1 is undefined.

The `min` and `max` entries each have the following attributes:

Min and max entry attributes

Attribute	Description
`measurefunction`	String indicating the function measured for the reading (current, voltage, ohms or watts)
`measurerange`	The full-scale range value for the measure 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 *`bufferVar`*`.collectsourcevalues` 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 *`bufferVar`*`.collecttimestamps` 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 will give the actual time the measurement was acquired
`bufferVar` is the name of the buffer. See smuX.buffer.getstats() (on page 7-172) for additional information.	

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 attributes

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, `rb[2]` and `rb.readings[2]` 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 of 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 3-16).
`timestamps`	If enabled, an array (a Lua table) of timestamps, in seconds, of when each reading occurred. These are relative to the `basetimestamp` for the buffer. See Reading buffer commands (on page 3-11).

* The default attribute is `readings`, which can be omitted. For example, both `smua.nvbuffer1` and `smua.nvbuffer1.readings` will access readings from buffer 1.

Examples:

The following programming example illustrates how to output 100 readings from Buffer 1:

```
printbuffer(1, 100, smua.nvbuffer1.readings)
```

Similarly, the following would output 100 source values from buffer 1:

```
printbuffer(1, 100, smua.nvbuffer1.sourcevalues)
```

The default reading attribute is `readings`, and can be omitted. Thus, the following would also output 100 readings from buffer 1:

```
printbuffer(1, 100, smua.nvbuffer1)
```

Buffer status

The buffer reading status attribute can include the status information as a numeric value; see the following table for values. For example, to access status information for second element, use the following command:

```
stat_info = smua.nvbuffer1.statuses[2]
```

Buffer status bits

Bit	Name	Hex value	Description
B0	FastADC	0x01	Fast ADC was used to make the reading
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	Rel applied to reading
B6	Compliance	0x40	Source function in compliance
B7	Filtered	0x80	Reading was filtered

Dynamic reading buffers

Reading buffers can also be allocated dynamically. Dynamic reading buffers are created and allocated with the `smua.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)
```

Allocated reading buffers can be deleted as follows:

```
mybuffer = nil
```

Dynamically allocated reading buffers can be used interchangeably with the `smua.nvbufferY` buffers that are described in Reading buffer designations (on page 3-10).

Buffer examples

Dedicated reading buffer example

The following programming example illustrates how to store data using dedicated reading buffer 1 for channel A. In the example, the Model 2651A loops for voltages from 0.01 V to 1 V with 0.01 V steps (essentially performing a staircase sweep), stores 100 current readings and source values in buffer 1, and then recalls all 100 readings and source values.

```
-- Restore Model 2651A defaults.
smua.reset()
-- Select channel A display.
display.screen = 0
-- 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 count to 1.
smua.measure.count = 1
-- Select source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set bias voltage to 0 V.
smua.source.levelv = 0.0
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Loop for voltages from 0.01 V to 1 V.
for v = 1, 100 do
   -- Set source voltage
   smua.source.levelv = v * 0.01
   -- Measure current, store in buffer.
   smua.measure.i(smua.nvbuffer1)
end
-- Turn off 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 for storing both current and voltage readings using buffer 1 for current and buffer 2 for voltage readings. The Model 2651A stores 100 current and voltage readings and then recalls all 100 sets of readings.

```
-- Restore Model 2651A 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 Model 2651A stores 100 current readings in `mybuffer` and then recalls all the readings.

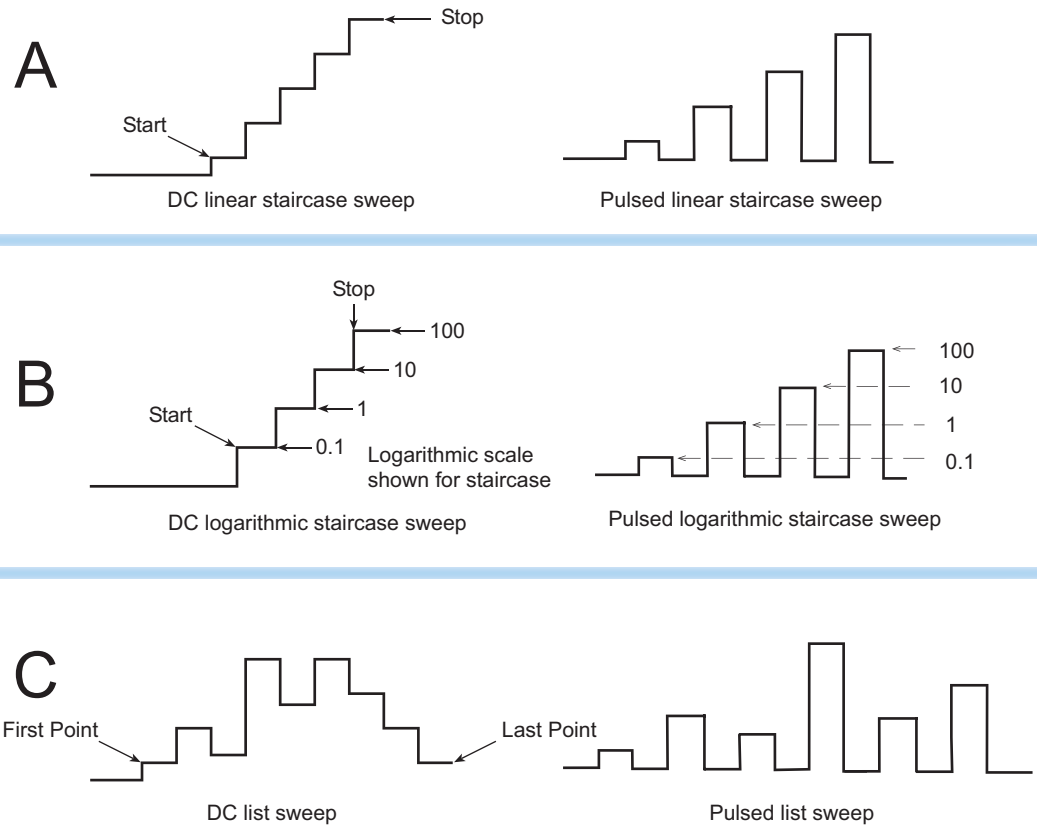
```
-- Restore Model 2651A defaults.
smua.reset()
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select measure V autorange.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Set buffer count to 100.
smua.measure.count = 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
-- Create a temporary reading buffer.
mybuffer = smua.makebuffer(smua.measure.count)
-- Store current readings in mybuffer.
smua.measure.overlappedi(mybuffer)
-- Wait for buffer to fill.
waitcomplete()
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
-- Output readings 1 to 100 from mybuffer.
printbuffer(1, 100, mybuffer)
-- Delete mybuffer.
mybuffer = nil
```

Sweep operation

Overview

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument can generate DC and pulsed sweeps to perform source-only sweeps, source-and-measure sweeps, or measure-only sweeps. There are three sweep types shown in the following figure: DC and pulsed linear staircase sweeps (A), DC and pulsed logarithmic staircase sweeps (B), and DC and pulsed list sweeps (C). Details about each kind of sweep follow the figure.

Figure 34: Sweep types



DC and pulsed linear staircase sweeps (A): With 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. This portion of the figure (A) 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 pulsed linear staircase sweeps return to the idle level between pulses.

DC and pulsed logarithmic staircase sweeps (B): 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. This portion of the figure (B) 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 pulsed logarithmic staircase sweeps return to the idle level between pulses.

DC and pulsed list sweeps (C): The list sweep allows you to program arbitrary sweep steps anywhere within the output voltage or current range of the Model 2651A. This portion of the figure (C) 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 pulsed list sweeps return to the idle level between pulses.

Sweep characteristics

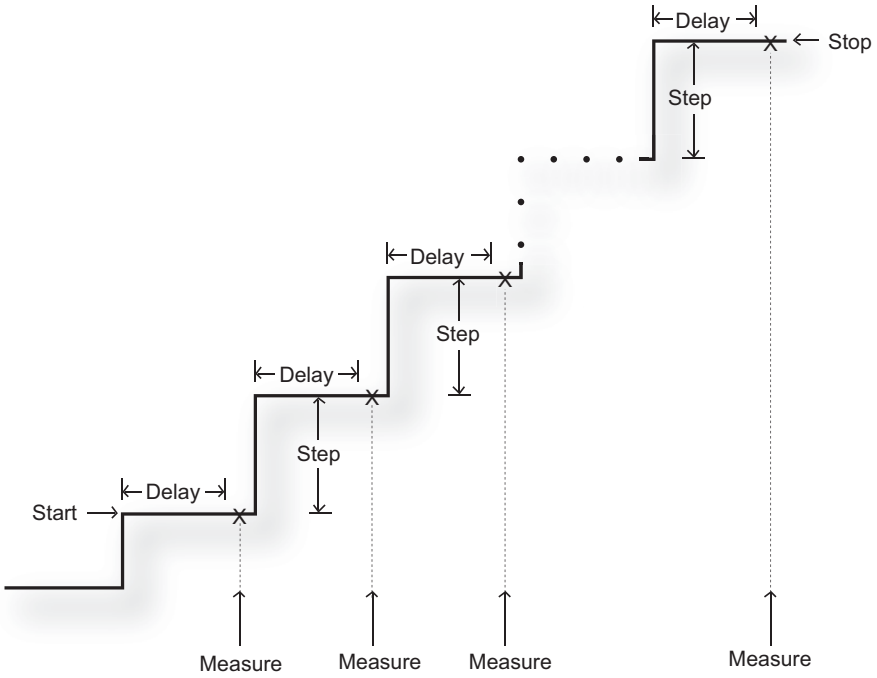
NOTE

For any of the sweep types, program a pulse mode sweep by configuring the end pulse action. Refer to Pulse mode sweeps (on page 3-26) for more information.

Linear staircase sweeps

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

Figure 35: 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:

step = (stop - start) / (points - 1)

NOTE

The number of sweep steps actually performed is determined by the trigger count. Refer to Triggering (on page 3-32) 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 will go to the start source level. The output will then change in equal steps until the stop level is reached. If the trigger count is greater than the number of points specified, the SMU will start over at the beginning value.

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

```
smua.trigger.source.action
```

Example:

```
-- 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 7-243).

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. 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 3-32) 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 number of points in the sweep
N = 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. Both of the following figures depict the effect of the asymptote on the inflection of the sweep curve. The following two figures show sample sweeps.

Figure 36: Increasing logarithmic sweep

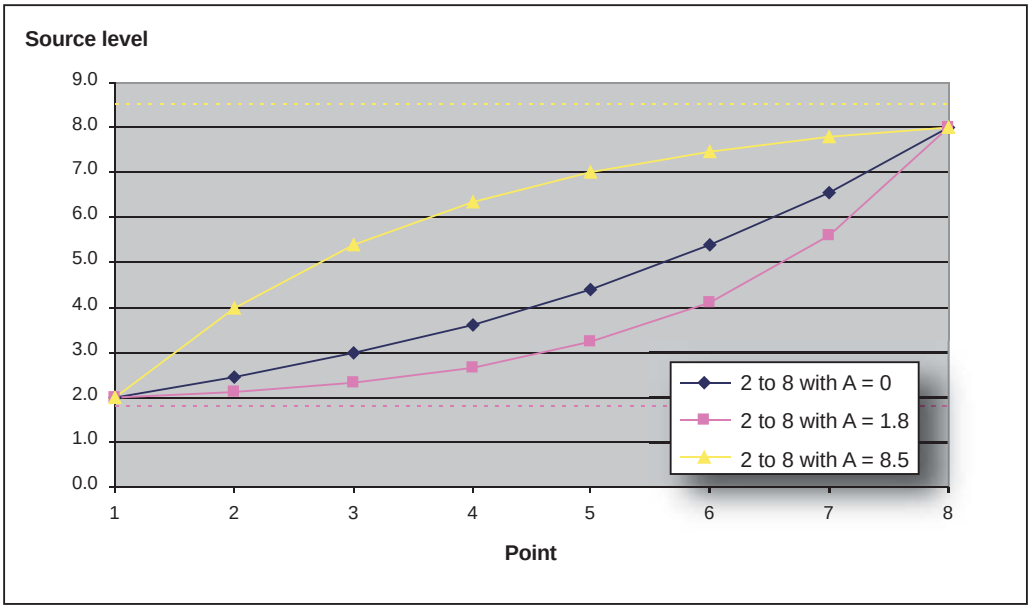
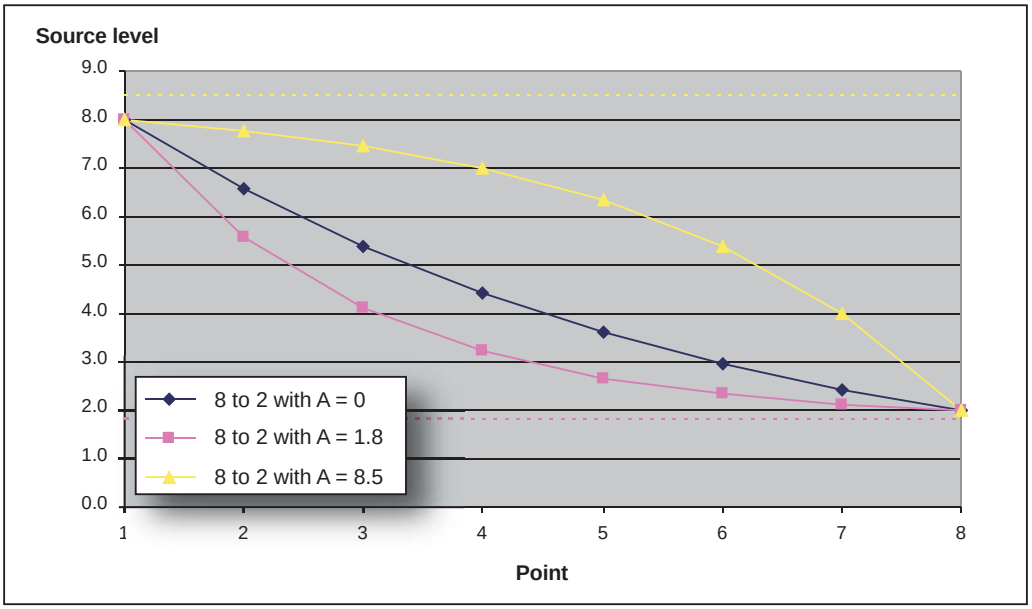


Figure 37: Decreasing logarithmic sweep



Solving for *k* and *b* provides the following formulas:

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

$$b = 10^{\left(\frac{log10(V_{end} - A) - log10(V_{start} - A)}{N-1}\right)}$$

Where:

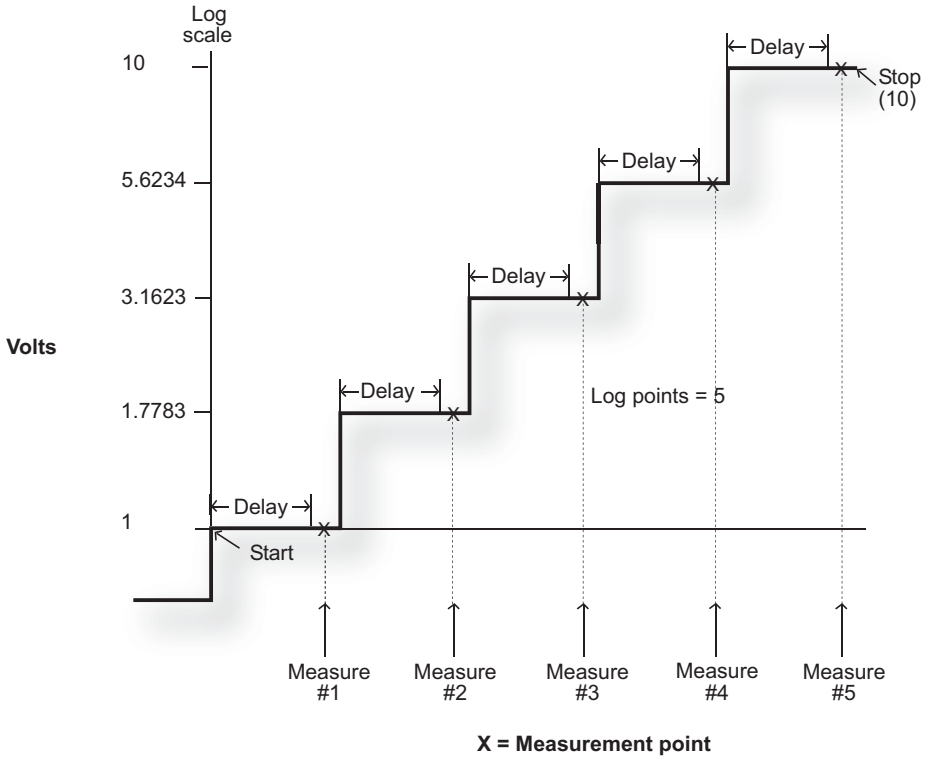
V_{end} = The source value at the end point
V_{start} = The source value at the start point
N = The number of points in the sweep
A = The asymptote value

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



In this example:

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

Using the formula above, $k = 1$

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

$$\left(\begin{array}{c} \text{Log Step Size} = \dfrac{\log10(\text{stop}-0) - \log10(\text{start}-0)}{\text{Points} - 1} \\ = \dfrac{\log10(10) - \log10(1)}{5 - 1} \\ = \dfrac{(1 - 0)}{4} \\ = 0.25 \end{array}\right)$$

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

The five log steps for this sweep are listed in the Logarithmic sweep points 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 will go to the start level (1 V) and sweep 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) will output when performing a logarithmic sweep. After configuring the sweep, you must also enable the source action by setting the following attribute:

```
smua.trigger.source.action
```

Example:

```
-- 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()`.

List sweeps

Use a list sweep to configure a sweep with arbitrary steps. 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 the that the source-measure unit (SMU) will output when performing a list sweep. After configuring the sweep, you must also enable the source action by setting the `smua.trigger.source.action` attribute.

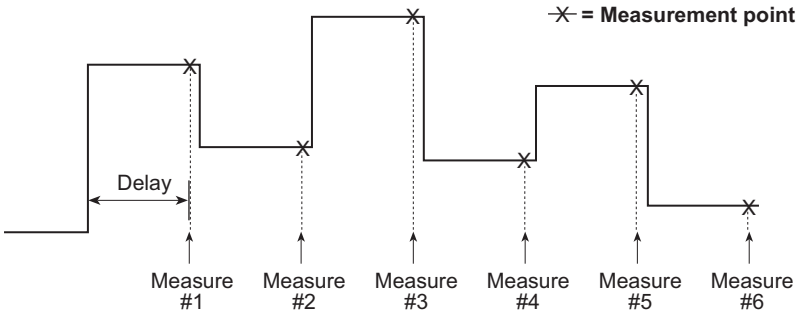
Example:

```
-- 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 will continue 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 unit cycles through the sweep points in the programmed order.

Figure 39: List sweep example



Pulse mode sweeps

A pulse mode sweep can be created for any of the sweep types by configuring the end pulse action.

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

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

To configure a DC sweep for SMU A, send:

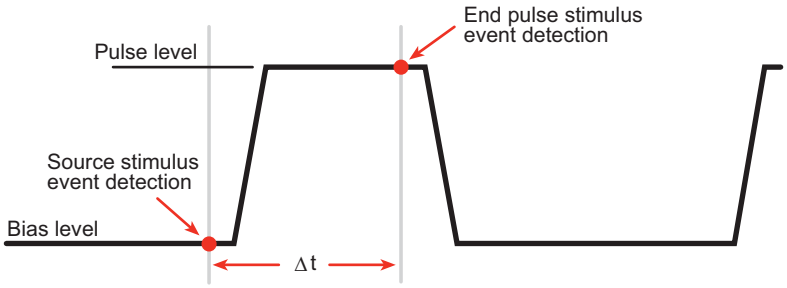
```
smua.trigger.endpulse.action = smua.SOURCE_HOLD
```

Timers must be used to configure the pulse width and period. Refer to Using timers to perform pulse mode sweeps (on page 3-45) for details.

The pulse width is managed by controlling the duration between the source stimulus event and the end pulse stimulus event. Note that 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 fast ADC mode can be used to characterize this latency, in order to better control the shape of the pulse under a particular set of test conditions.

The figure below shows the source and end pulse stimulus events in relationship to the pulse. Any change in Δt will result in a corresponding change in the pulse width.

Figure 40: 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:

Duty cycle = Pulse width / (Pulse width + 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:

Duty cycle = 10 ms / (10 ms + 90 ms)

= 10 ms / 100 ms

= 0.10

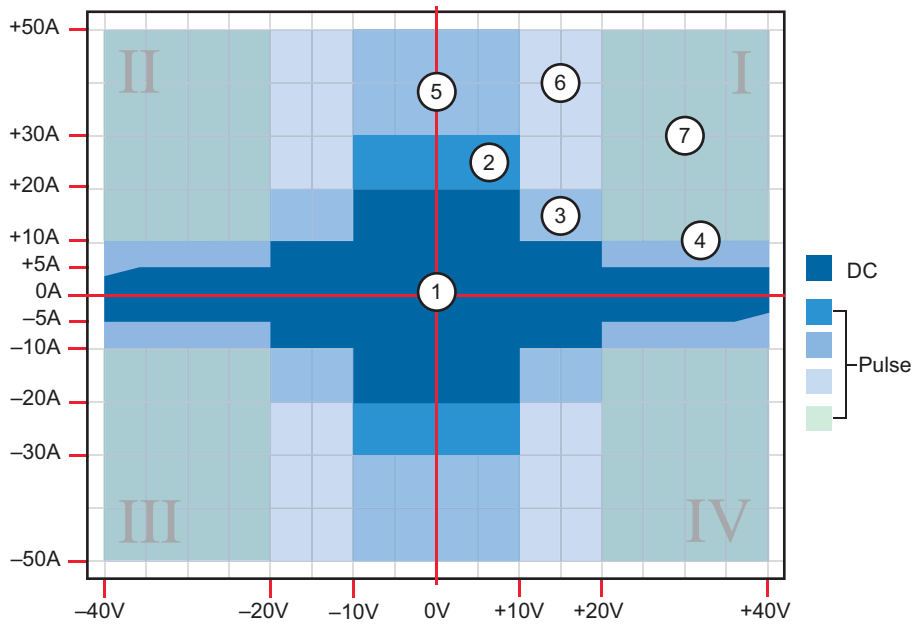
= 10 percent

See Maximum duty cycle equation (on page 4-4) for additional information on calculating the maximum duty cycle for a SMU.

Pulsing in the extended operating area (EOA)

Pulse sweeps can be performed outside of the standard operating area by setting the appropriate compliance level. Review the specifications for the Model 2651A to determine the maximum current and voltage values available in pulse mode. When pulsing in the extended operating area (EOA), the source-measure unit (SMU) will force the pulse to end early if the pulse width exceeds the maximum value. It will also delay the next source action as necessary to stay within the duty cycle capabilities of the SMU. The following figure and table illustrate the pulse regions for a SMU when pulsing in the EOA. Refer to the Model 2651A specifications on the Keithley Instruments website (http://www.keithley.com) for the latest pulse width and duty cycle information.

Figure 41: Pulsing in the extended operating area (EOA)



Pulse region specification

Region (quadrant diagram)	Region maximum	Maximum pulse width	Maximum duty cycle
1	5 A at 40 V	DC, no limit	100%
1	10 A at 20 V	DC, no limit	100%
1	20 A at 10 V	DC, no limit	100%
2	30 A at 10 V	1 ms	50%
3	20 A at 20 V	1.5 ms	40%
4	10 A at 40 V	1.5 ms	40%
5	50 A at 10 V	1 ms	35%
6	50 A at 20 V	330 µs	10%
7	50 A at 40 V	300 µs	1%

Configuring and running sweeps

Configuring compliance limits remotely

Voltage and current limits can be configured using the `smuX.trigger.source.limitY` attribute, which sets the sweep source limits. For example, to set the `smua` sweep limit to 10 V, execute:

```
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 program SMU A to return the source back to the idle source level at the end of a sweep:

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

Configuring measurements during a sweep

Measurements can be performed 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 will store the readings. For example, to store the voltage readings taken during the sweep:

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

To recall sweep data:

- **Using the front panel:** Press the **RECALL** key, and then select **DATA** or **STATISTICS**. For DATA: select the buffer, and then choose reading numbers to display using the navigation wheel ⟳ or cursor keys. For STATISTICS: select the buffer, and then choose MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, or PK-PK to display using the navigation wheel ⟳ or cursor keys. Recalling readings from the reading buffer using the front panel can only be done if one of the dedicated reading buffers is used to store the sweep data.
- **Remote:** Use the `printbuffer()` function to request buffer readings.

See Reading buffers (on page 3-6) 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 performs 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. Use the trigger count to set the number of sweep points to output. Use the arm count to set the number of times to perform the sweep. See Triggering (on page 3-32) 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 as a way 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

The `smuX.abort()` function can be used 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 3-32) for more information.

Sweeping using factory scripts

Factory script functions that perform linear staircase, logarithmic staircase, and list sweeps are defined in Remote Commands (on page 5-1). You can use the factory script functions to perform and execute simple sweeps, or use them as examples for programming your own custom sweeps.

Front panel

To run a sweep from the 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, and then follow the display prompts to complete the test.

See Remote Commands (on page 5-1) for more information about using factory scripts.

Press the **RECALL** key to access sweep data stored in dedicated reading buffer 1. See Reading Buffers for more details about the buffer.

Sweep programming examples

Procedures for programming and running a sweep for three sweep types are given on the following pages. Each of these procedures includes commands for a typical sweep example. The following table summarizes parameters for each of these examples.

Sweep example parameters

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

Linear staircase sweep example

The programming example below illustrates a staircase sweep.

`-- Restore Model 2651A defaults.`
`smua.reset()`
`-- Set compliance to 1 V.`
`smua.source.limitv = 1`
	1. Configure source functions.
	Restores defaults and sets the compliance to 1 V.

`-- Linear staircase sweep`
`-- 1 mA to 10 mA, 0.1 second delay,`
`-- 10 points.`
`SweepILinMeasureV(smua, 1e-3, 10e-3, 0.1, 10)`
	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.

`printbuffer(1, 10, smua.nvbuffer1.readings)`	**3. Request readings.** Requests readings from buffer 1.

Pulse sweep examples

The programming example below illustrates a pulse sweep.

`-- Restore Model 2651A defaults.`
`smua.reset()`
`-- Set compliance to 10 V.`
`smua.source.limitv = 10`
	1. Configure source functions.
	Restores defaults and set the compliance to 10 V.

`-- 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)`
	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.

`printbuffer(1, 10, smua.nvbuffer1.readings)`	**3. Request readings.** Requests readings from buffer 1.

List sweep example

The programming example below illustrates a list sweep.

`-- Restore Model 2651A defaults.`
`smua.reset()`
`-- Set compliance to 10 mA.`
`smua.source.limiti = 10e-3`
	1. Configure source functions.
	Restores defaults and set the compliance to 10 mA.

`-- 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)`
	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.

`printbuffer(1, 5, smua.nvbuffer1.readings)`	**3. Request readings.** Requests readings from buffer 1.

Triggering

Remote triggering overview

There are two programming methods for triggering:

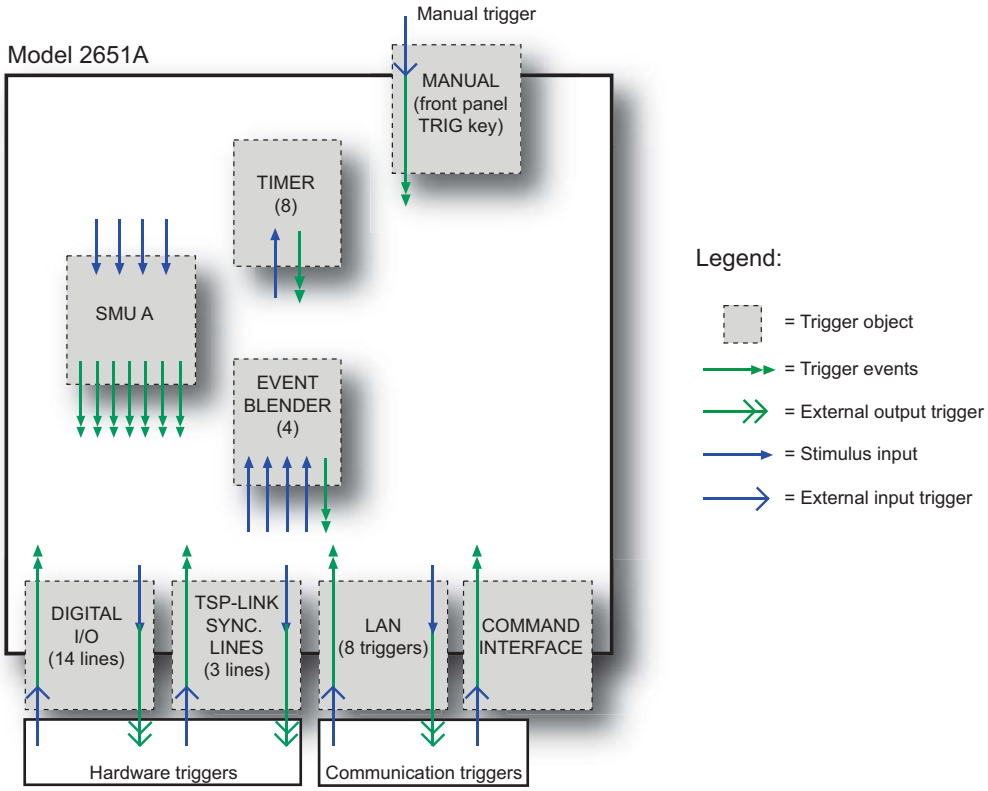
- Using the trigger model
- Interactive triggering

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 Model 2651A instrument.

Figure 42: 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
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the 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 measure action

Trigger event IDs*

Event ID	Event description
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.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 *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed.
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command device_trigger. 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.

* 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).

Using the remote trigger model

The source-measure unit (SMU) in the Model 2651A 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 43: Remote trigger model: Normal (synchronous) mode

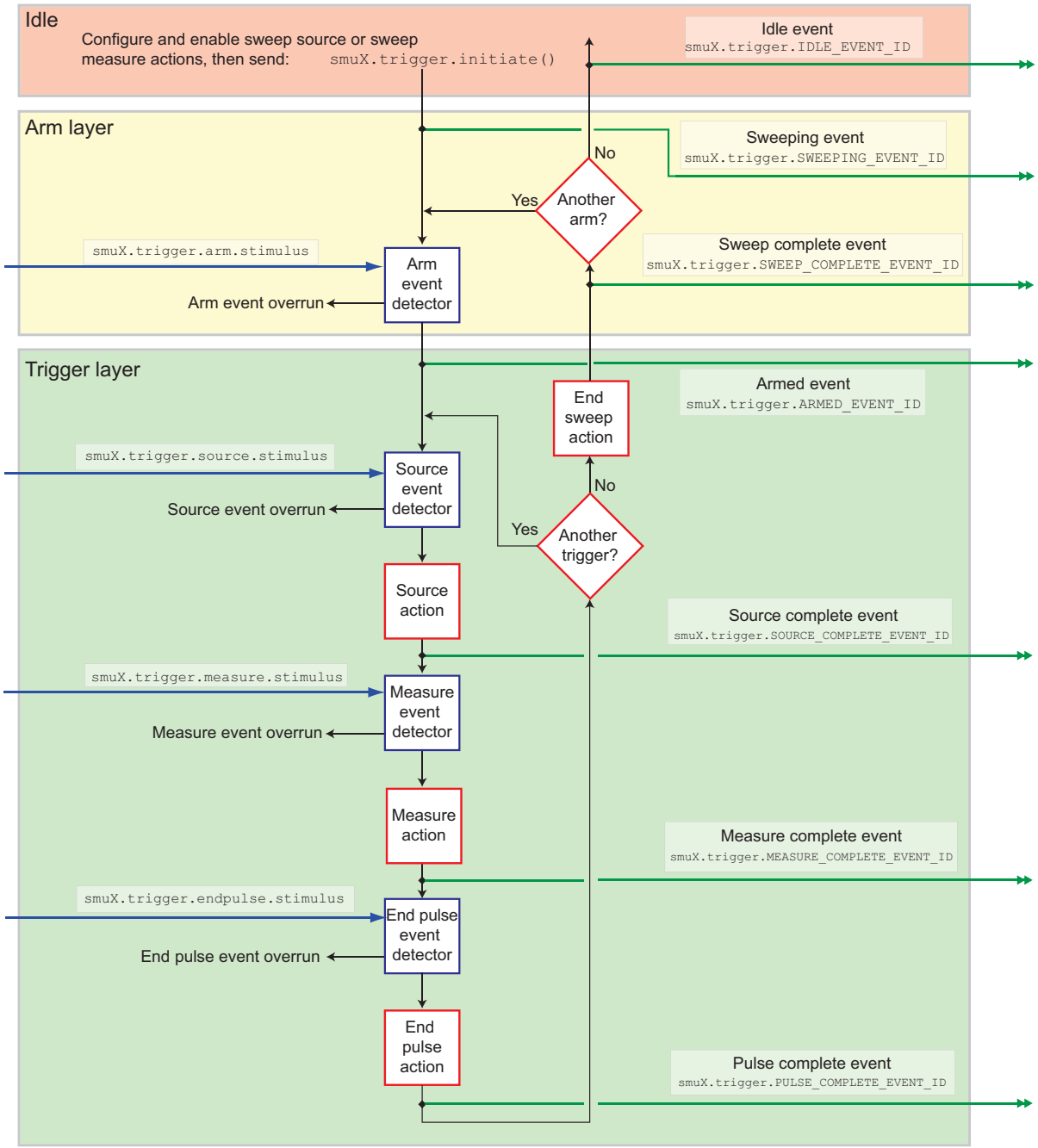
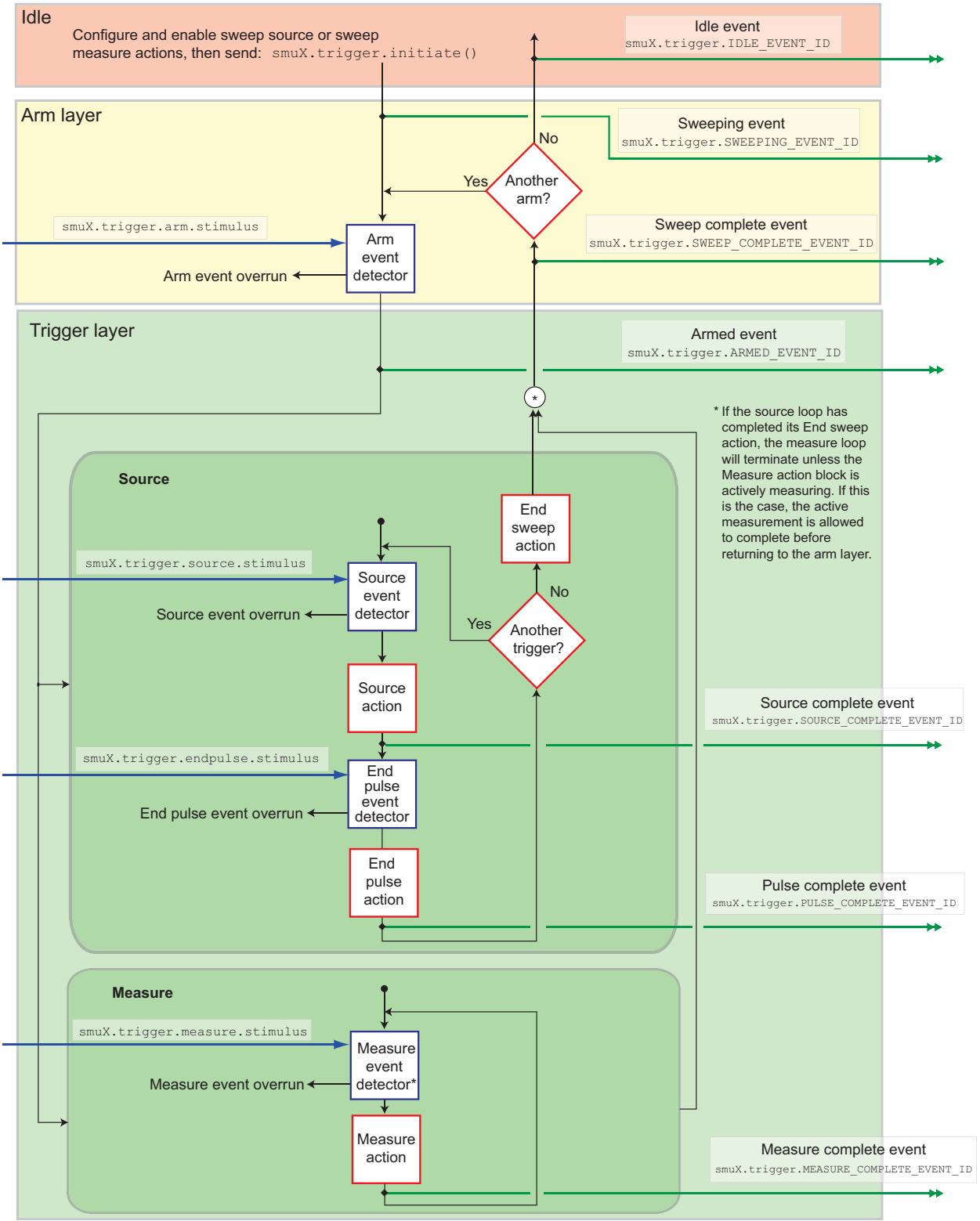


Figure 44: Remote trigger model: Asynchronous mode



When the `smuX.trigger.measure.action` attribute is set to `smuX.DISABLE` or `smuX.ENABLE`, the trigger model will operate in synchronous measurement mode. When it is set to `smuX.ASYNC`, it will operate 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 `smuX.trigger.initiate()` 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, measure, and pulse actions occur in the trigger layer. • **Source**: Outputs the programmed voltage or current source value. • **Measure**: 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

The source action can be configured using any of the following functions:

```
smua.trigger.source.linearY()
smua.trigger.source.logY()
smua.trigger.source.listY()
```

Where:

Y = Source function

Source functions cannot be changed within a sweep. See Sweep Operation (on page 3-19) for more details about the sweep functions.

To enable the source action, set the `smua.trigger.source.action` attribute to `smua.ENABLE`.

The source-measure unit (SMU) can be configured to perform any or all available measurements during a sweep using the `smua.trigger.measure.Y()` function. To enable the measure action for a simple synchronous sweep, set the `smua.trigger.measure.action` attribute to `smua.ENABLE`. To enable the measure action for an asynchronous sweep, set the `smua.trigger.measure.action` attribute to `smua.ASYNC`.

NOTE

In asynchronous mode, trigger your measurements before the source completes the sweep (before the end sweep action occurs).

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. Refer to Sweep Operation (on page 3-19) for information about configuring source and measure sweeps.

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 as configured by `smua.trigger.source.linearY()`, `smua.trigger.source.logY()`, or `smua.trigger.source.listY()`, then the SMU will satisfy the trigger count by restarting the sweep values from the beginning.
- If the trigger count is less than the number of source values configured, then the SMU will satisfy the trigger count and ignore the remaining source values.

For example, configure a three-point linear voltage sweep from 1 to 3 V, with the trigger count set to 2. The SMU will output 1 V, 2 V. If the trigger count is set to 6, then the SMU will output 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
```

Timers can be used to configure the pulse width and period (see Timers (on page 3-43) for more information). To disable pulse mode sweeps, set the `smua.trigger.endpulse.action` attribute to `smua.SOURCE_HOLD`.

SMU event detectors

As shown in the Using the remote trigger model (on page 3-34) topic, the source-measure unit (SMU) has multiple event detectors (see the table below) in order to control the timing of various actions. 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), then 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.

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 `smua.trigger.initiate()` function.
- The SMU moves from the arm layer into the trigger layer and the `smua.trigger.autoclear` attribute is enabled.

Using the TRIG key to trigger a sweep

The source-measure unit (SMU) can be configured to perform a sweep where each source step is triggered by the front-panel TRIG key. The source action is preceded by the source event detector (see the following figure). 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 front panel TRIG key is pressed.

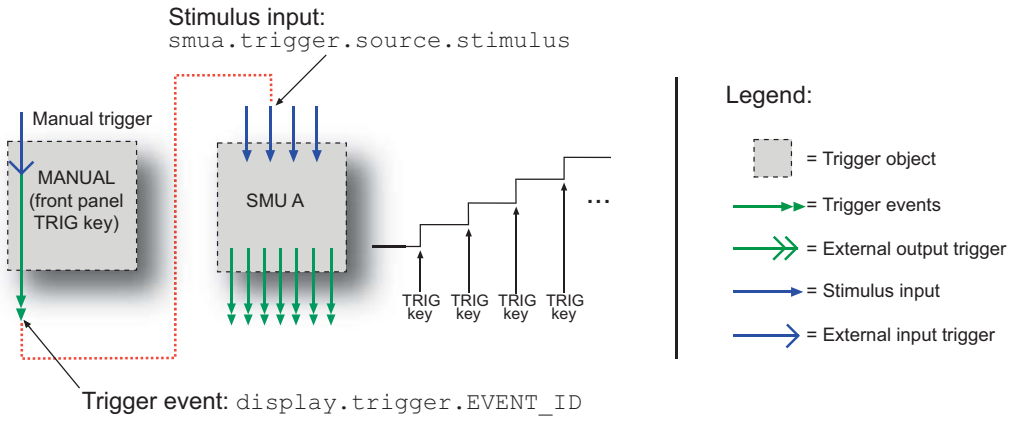
To configure the front panel TRIG key to trigger the source action, assign the trigger event created by the TRIG key (`display.trigger.EVENT_ID`) to the source stimulus input (`smua.trigger.source.stimulus`).

The programming example below illustrates how to configure a 10-point linear voltage sweep on SMU A, where each step is triggered by the front-panel TRIG key:

```
-- Configure a 10-point source voltage sweep.
smua.trigger.source.linearv(1, 10, 10)
smua.trigger.source.action = smua.ENABLE
-- Configure TRIG key press as input trigger for source action.
smua.trigger.source.stimulus = display.trigger.EVENT_ID
-- Command 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 will wait for the 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 3-19) for more information about sweep operation.

Figure 45: Front panel TRIG key triggering



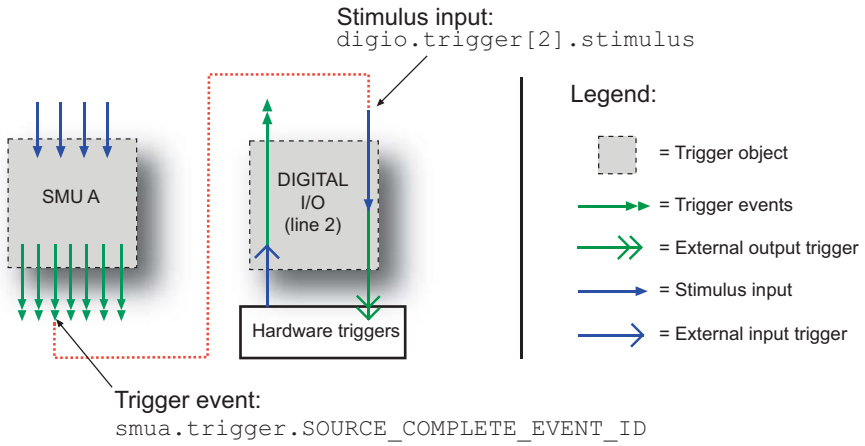
Using trigger events to start actions on trigger objects

Trigger objects can be configured 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 will perform 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 46: 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 3-49) 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. Please refer to Status Model (on page E-1) and the appropriate sections on each trigger object for further details on conditions under which an object generates an action overrun.

Digital I/O port and TSP-Link synchronization lines

The Model 2651A 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 3-83) for more information about connections and direct control of the digital I/O and TSP-Link synchronization lines.

Common attributes

Mode

Indicates the type of edge the hardware lines detects 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. The hardware trigger modes are described in greater detail in Hardware trigger modes (on page 3-56).

NOTE

Setting the mode to bypass will not allow the hardware line to be used for triggering.

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	None
Synchronous	High latching	Low	Falling
SynchronousA	High latching	High	Falling
SynchronousM	High	Low	Rising

Pulsewidth

Specifies the pulse width of the output trigger signal when the hardware line is asserted.

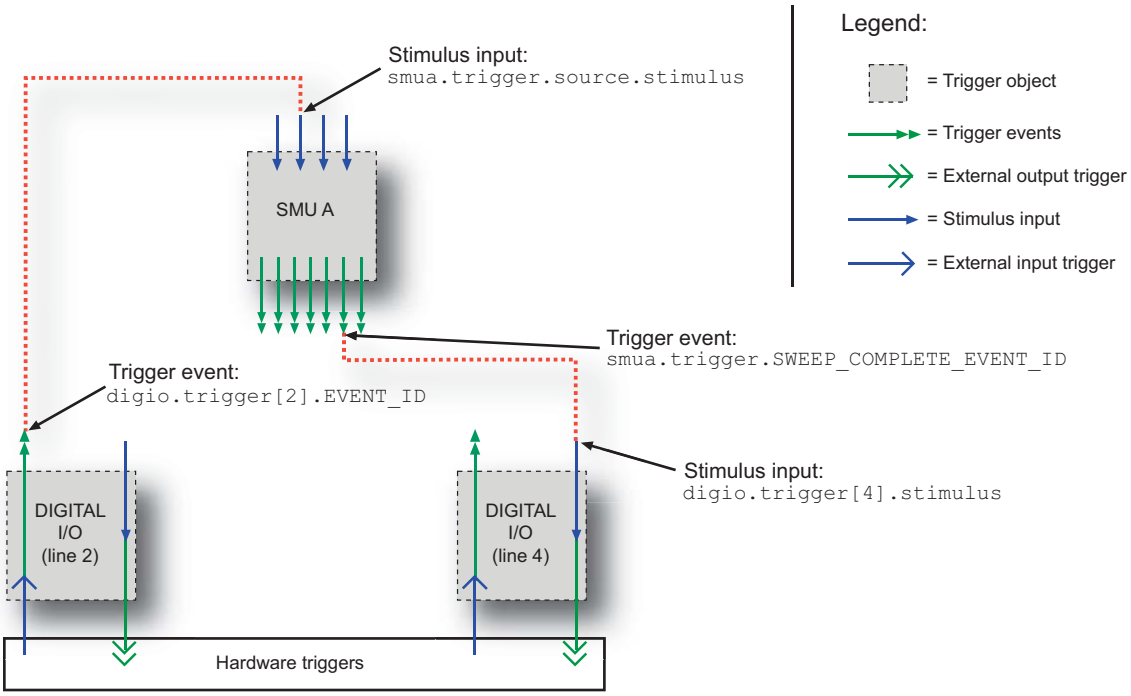
Trigger configuration on hardware lines

The Model 2651A can be configured to send digital signals to trigger external instruments. Linking these output triggers to the completion of certain source-measure actions enables hardware handshaking. The programming example below illustrates this:

```
-- Configure the Model 2651A 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
-- 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
```

This example's triggering setup is shown in the following figure.

Figure 47: External instrument triggering



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 3-56).

Timers

A timer is a trigger object that performs a delay when triggered. Timers can be used to create delays and to start measurements and step the source value at timed intervals. When a delay expires the timer generates a trigger event. The Model 2651A has 8 independent timers.

Timer attributes

Each timer has four attributes that can be configured.

Count

Configures the number of events to generate each time the timer is triggered. Each event is separated by a delay.

To configure the count, use the following attribute: `trigger.timer[N].count`

Set the count number to 0 (zero) to cause the timer to generate trigger events indefinitely.

Timer delays

Timers can be configured to perform the same delay each time or configured with a delay list that allows the timer to sequence through an array of delay values. All delay values are specified in seconds.

Delay: A delay is the period of time 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-second delay:

```
trigger.timer[3].delay = 10
```

Delay list: A custom list can be configured to allow the timer to use a different interval each time it performs a delay. Each time the timer is triggered, it uses the next delay in the list. The timer repeats the delay list after all of 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 seconds, 10 seconds,
-- 15 seconds, and 7 seconds.
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

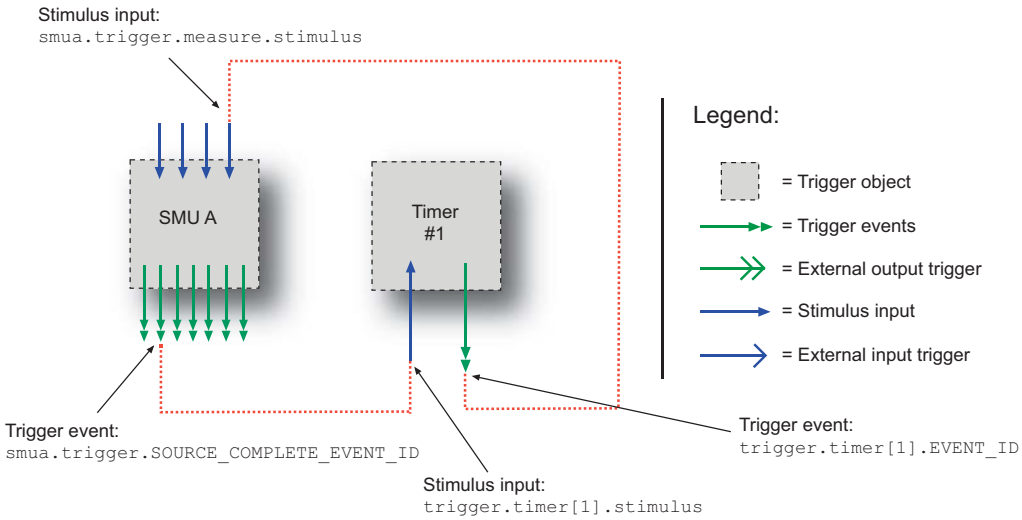
A timer can be configured to start a delay when a trigger object generates a trigger event. Timers cannot be started with a command. A trigger event from a trigger object must be used to initiate a delay.

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 48: Using a timer for an SDM cycle



Timer action overruns

The timer generates an action overrun when it is triggered while a timer delay is still in progress.

Using timers to perform pulse mode sweeps

Timers can also be used 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 examples below show a single pulse output and a pulse train output.

NOTE

The SMU end pulse action `smua.trigger.endpulse.action` must be set to `smua.SOURCE_IDLE` in order to create a pulse.

Single pulse example:

The 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 desired 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.

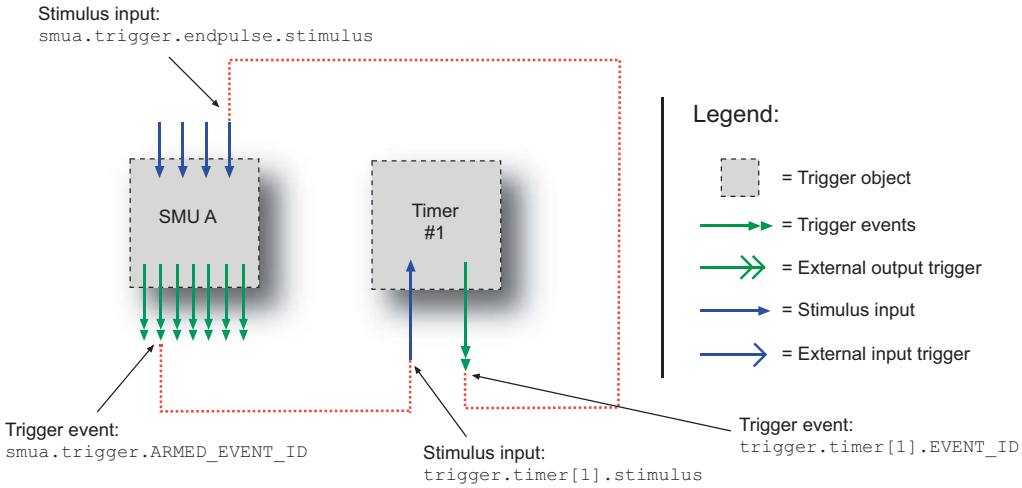
SMU A

- 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`.

```
-- Generate a single 500 us, 5 V pulse.
-- Configure a single-point voltage list sweep.
smua.trigger.source.listv({5})
smua.trigger.source.action = smua.ENABLE
smua.trigger.measure.action = smua.DISABLE
-- Configure other source parameters for best timing possible.
smua.trigger.source.limiti = 0.1
smua.source.rangev = 5
-- Configure 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
-- Trigger timer when the SMU passes through the ARM layer.
trigger.timer[1].stimulus = smua.trigger.ARMED_EVENT_ID
-- Configure source action to start immediately.
smua.trigger.source.stimulus = 0
-- Configure endpulse action to achieve a pulse.
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
-- Set appropriate counts of trigger model.
smua.trigger.count = 1
smua.trigger.arm.count = 1
-- Turn on output and trigger SMU to output a single pulse.
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
-- Wait for the sweep to complete.
waitcomplete()
```

The following figure shows the trigger setup for this example.

Figure 49: Single pulse triggering



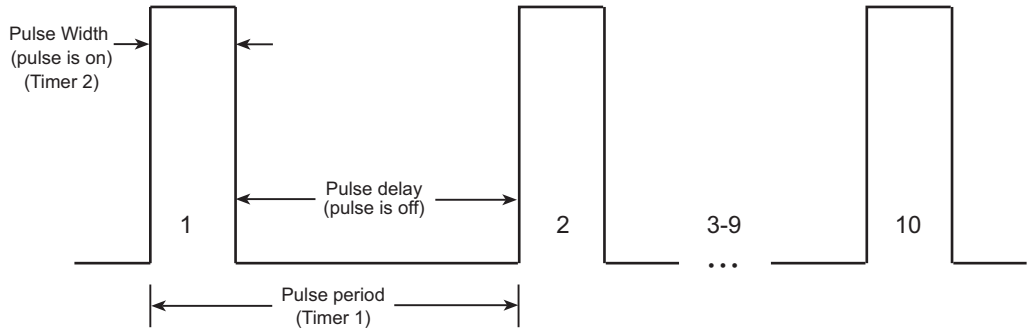
Pulse train example:

The 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 delay attribute to the desired 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 50: Pulse train



Timer 2: Pulse width timer

- Set the delay attribute to the desired pulse width (see the following figure).
- Set the stimulus input to Timer 1's event ID (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.

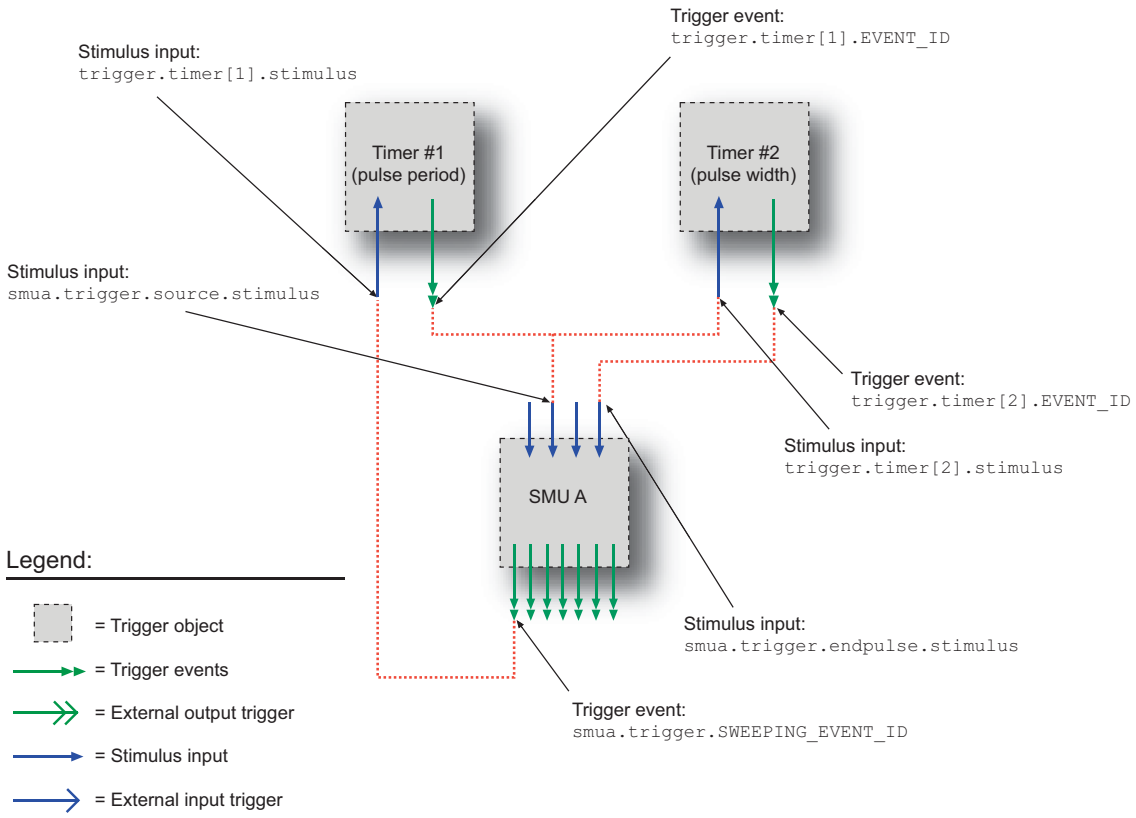
SMU A

- Set the source stimulus input to Timer 1's event ID 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 Timer 2's event ID so that the end pulse action executes when the pulse width timer expires.
- Set the trigger count equal to 1.
- Set the arm count equal to the total number of pulses to output.

```
-- Generate a 10-point pulse train where each pulse has a width of 600
-- microseconds and a pulse period of 5 milliseconds.
-- Alias the trigger timers to use for pulse width and period.
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
-- Create a fixed level voltage sweep.
smua.trigger.source.listv({5})
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
-- Set pulse width.
pulse_timer.delay = 0.0006
-- Trigger pulse width timer with period timer.
pulse_timer.stimulus = period_timer.EVENT_ID
-- Output one pulse per period.
pulse_timer.count = 1
-- Set the pulse period.
period_timer.delay = 0.005
-- Set pulse period count to generate 10 pulses.
period_timer.count = 9
-- Trigger pulse period timer when a sweep is initiated.
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
-- Configure the timer to output a trigger event when it
-- starts the first delay.
period_timer.passthrough = true
-- Trigger SMU source action using pulse period timer
smua.trigger.source.stimulus = period_timer.EVENT_ID
-- Trigger SMU end pulse action using pulse width timer.
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
-- Set Trigger Model counts.
smua.trigger.count = 1
-- Configure the SMU to execute a 10-point pulse train.
smua.trigger.arm.count = 10
-- Prepare SMU to output pulse train.
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
-- Wait for the sweep to complete.
waitcomplete()
```

The following figure shows the trigger setup for this example.

Figure 51: Pulse train triggering



Event blenders

The ability to combine trigger events that occur at different times is known as event blending. An event blender can be used to wait for a specific input trigger or to wait for up to four input triggers to occur before responding with an output event.

There are four event blenders that can be used to monitor and respond to multiple stimulus events. Each event blender can be configured to monitor a maximum of four different trigger events.

Event blender modes

Event blenders can be used to 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. A different trigger event ID can be assigned to each stimulus input. The programming example below illustrates how to assign the source complete event ID of SMU A and the trigger event ID of digital I/O line 1 to stimulus inputs 1 and 2 of event blender 1:

```
trigger.blender[1].stimulus[1] = smua.trigger.SOURCE_COMPLETE_EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[1].EVENT_ID
```

Action overruns

Action overruns are generated by event blenders depending on the mode, as shown in the following table.

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 Model 2651A 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 very similar to hardware synchronization except LXI trigger packets are used instead of hardware signals. The hardware value is a bit in the LXI trigger packet that simulates the state of a hardware trigger line. The Model 2651A stores the hardware value of the last LXI trigger packet 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 set, the Model 2651A ignores the hardware value of the packet and generates a trigger event. The Model 2651A always sets the stateless flag for outgoing LXI trigger packets. If the stateless event flag is not set, then 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 Model 2651A assumes an edge transition was missed and generates a trigger event. The following table illustrates edge detection in LAN triggering.

NOTE

Instruments that are compliant to LXI versions prior to 1.2 always process the hardware value. Instruments compliant to LXI version 1.2 and later are required to ignore the hardware value when the stateless event flag is set.

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 edge detection method in incoming LXI trigger packets. The mode 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
RisingM	Rising	Positive	Same as Rising
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 have falling-edge mode.
lan.trigger[2].mode = lan.TRIG_FALLING
```

Understanding LXI trigger event designations

LAN trigger objects generate LXI trigger events. The LXI standard designates trigger events as 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

The Model 2651A can be configured to output an LXI trigger packet to other LXI instruments. To generate LXI trigger packets, you must first call the `lan.trigger[N].connect()` function. Select the event that triggers the outgoing LXI trigger packet by assigning the specific event ID to the LAN stimulus input.

Command interface triggering

A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 `device_trigger` method is invoked (VXI-11 only)
- A `*TRG` message is received

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 input 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 Model 2651A input queue. The Model 2651A does not process incoming commands while a script is running. Input triggers that are not processed can cause an overflow in the input queue. It is important to make sure a script processes triggers while it is running.

NOTE

The input queue can fill up with trigger entries if too many `*TRG` messages are received while a test script is running. This can be averted by using the `localnode.prompts4882` attribute (see Remote commands (on page 5-1) for more information), and by using `trigger.wait()` calls that remove the `*TRG` messages from the input queue. If the input queue fills with too many trigger entries, messages like `abort` will not be processed.

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 3-39) 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 would require more dynamic control of triggering than the static trigger setup provides. For such cases, a setup providing interactive trigger programming would allow the generation and detection of trigger events that can be controlled on demand under remote control. For example, interactive triggering can be used when you need to make multiple source function changes or implement conditional branching to other test setups based on recent measurements.

Detecting trigger events using the wait() function

All of the Model 2651A trigger objects (except for SMUs) 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 Model 2651A 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 trigger events

Certain trigger objects can be used 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:

```
-- 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. The `release()` function must be used to release the line in order to output another external trigger.

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()
```

Using the set() function to bypass SMU event detectors

The `set()` function is 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.

A common example of when the `set()` function can be used 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. The `set()` function can be used 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 will generate a detector overrun. Detector overruns can be checked by reading the `overrun` attribute of the trigger object. The attribute is set to `true` when an overrun occurs. The `clear()` function can be used 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 action overruns that are reported in the status model.

The programming example below illustrates how to check 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

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. Upon receipt of the trigger, the Model 2651A 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 voltage reading.
reading = smua.measure.v()
-- Send 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 they are ready to continue. If the TRIG key is not pressed, the test will continue 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. It commands the Model 2651A to wait for an external input trigger on digital I/O line 2. If a trigger event occurs, the Model 2651A 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, then turn off the output of smua and issue a message
-- on the 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

Different hardware trigger modes can be used for digital I/O and TSP-Link synchronization. Use hardware triggers to integrate Keithley instruments and non-Keithley instruments in a test system. The Model 2651A 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 Remote Commands (on page 5-1).

NOTE

For direct control of the line state, use the bypass trigger mode.

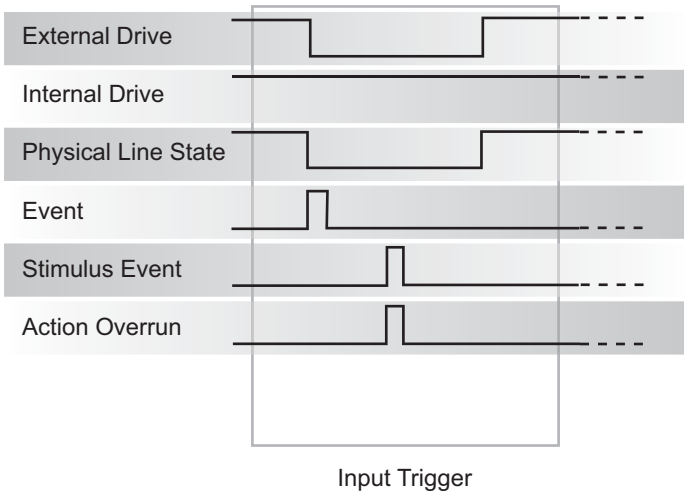
Falling edge trigger mode

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.

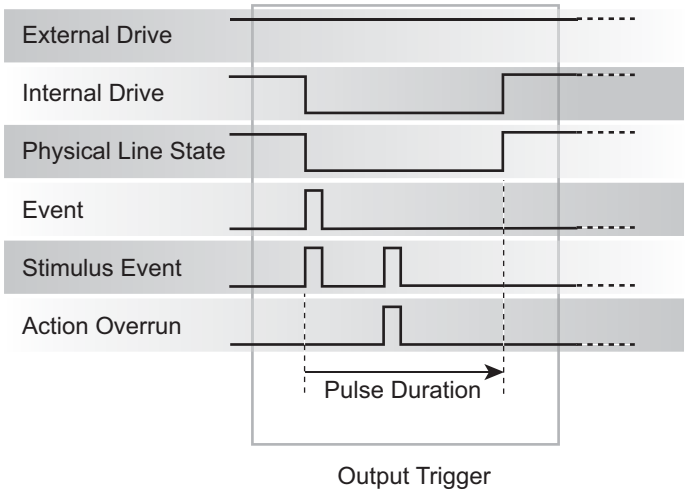
Figure 52: Falling edge input trigger



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 53: 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.

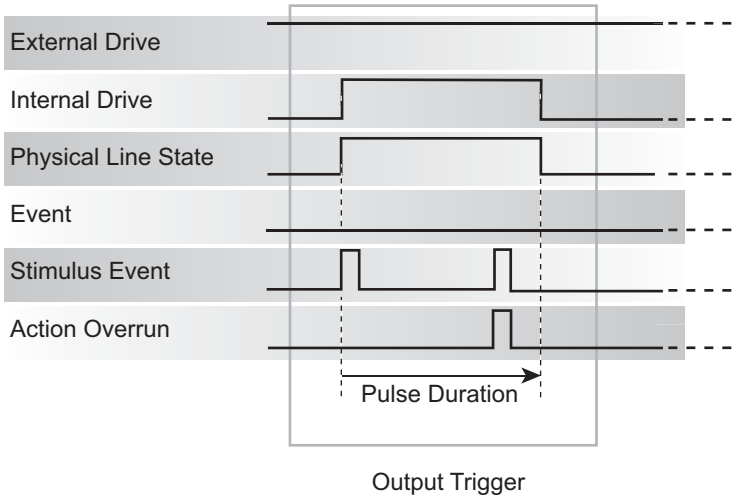
NOTE

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 while a stimulus event occurs.

Figure 54: RisingM output trigger



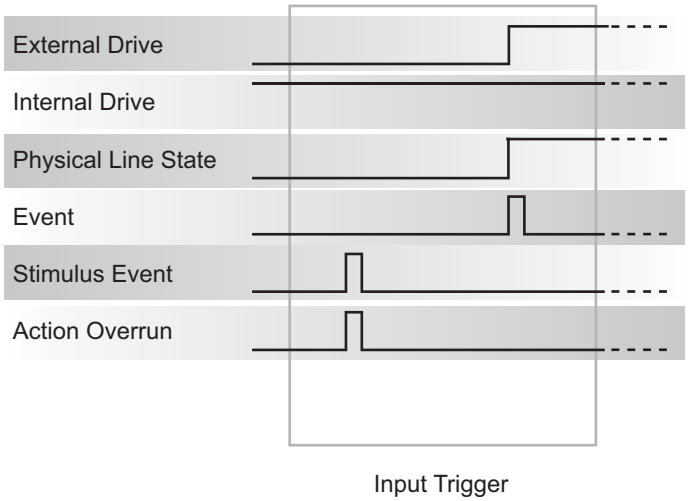
Rising edge acceptor trigger mode

The rising edge acceptor trigger mode (RisingA) generates a low pulse and detects rising edge pulses (see the following figures).

Input characteristics:

- All rising edges generate an input event.

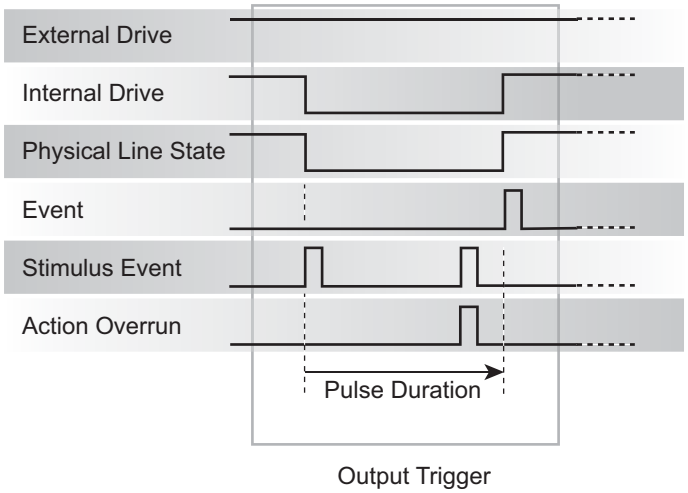
Figure 55: RisingA input trigger



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 56: RisingA output trigger



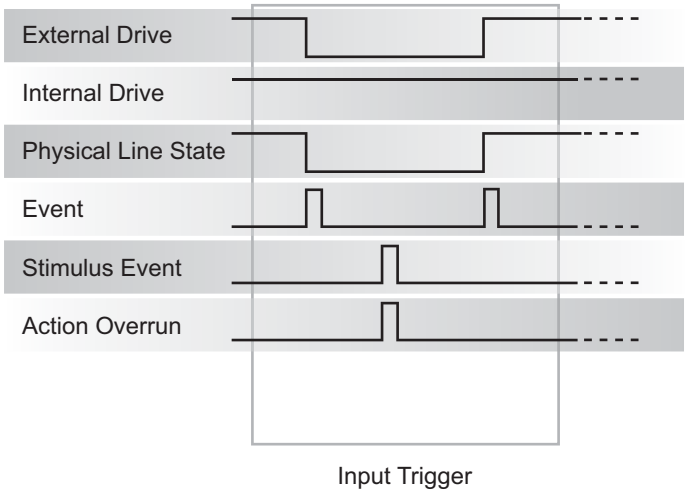
Either edge trigger mode

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.

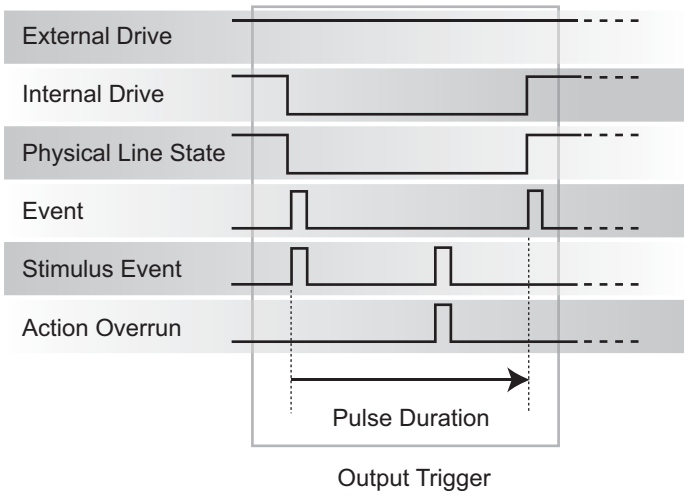
Figure 57: Either edge input trigger



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 occur if the physical line state is low while a stimulus event occurs.

Figure 58: 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 (SynchronousM) 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.

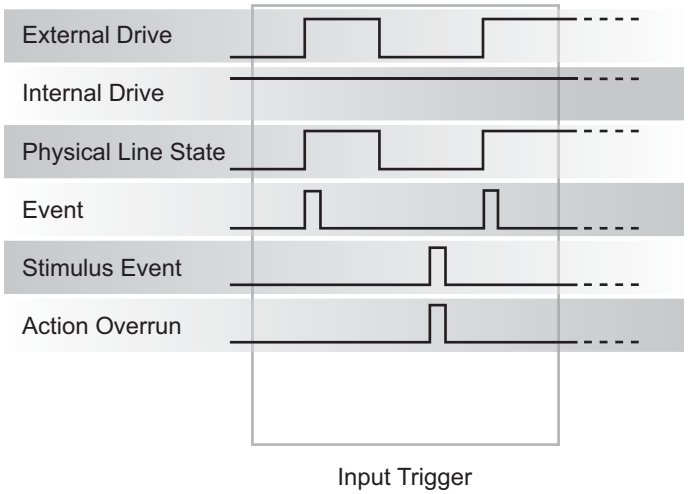
NOTE

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.

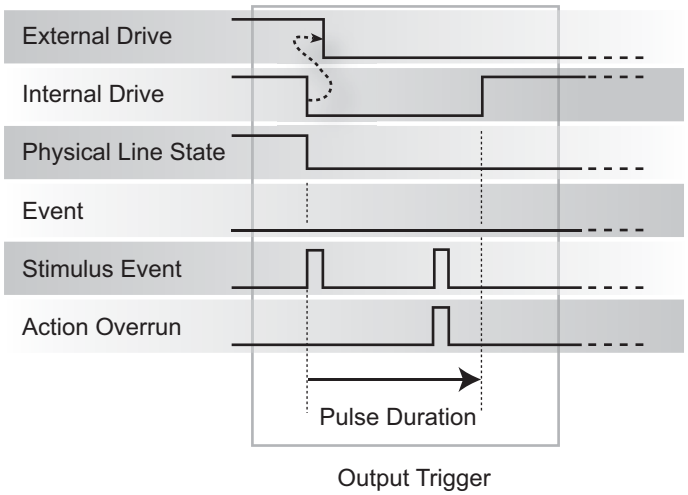
Figure 59: SynchronousM input trigger



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 while a stimulus event occurs.

Figure 60: SynchronousM output trigger



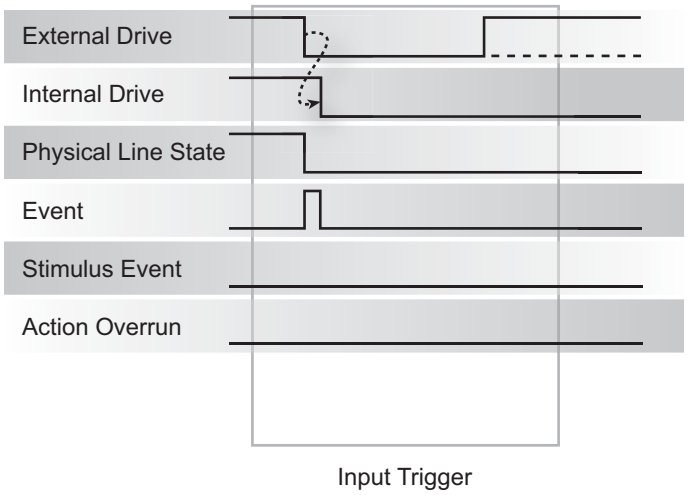
Synchronous acceptor trigger mode (SynchronousA)

Use the synchronous acceptor trigger mode (SynchronousA) in conjunction with the SynchronousM trigger mode. The role 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.

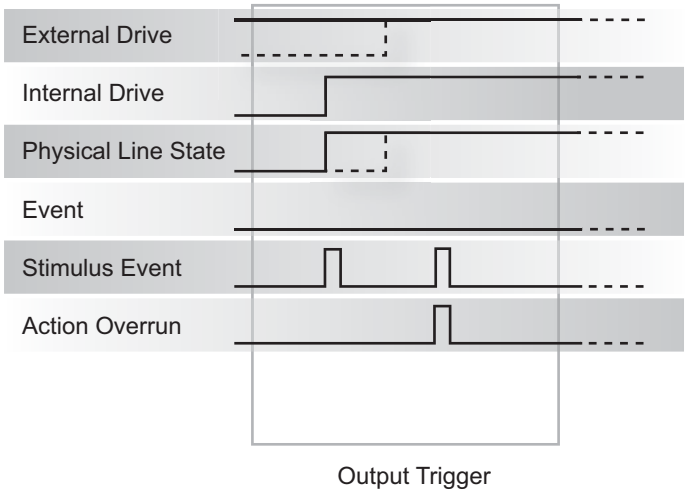
Figure 61: SynchronousA input trigger



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 62: SynchronousA 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.

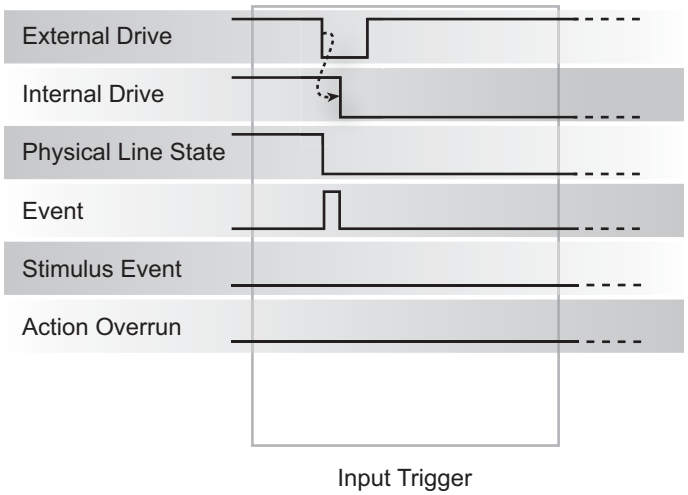
NOTE

Keithley Instruments recommends using SynchronousA and SynchronousM modes only.

Input characteristics:

- The falling edge generates an input event and latches the internal drive low.

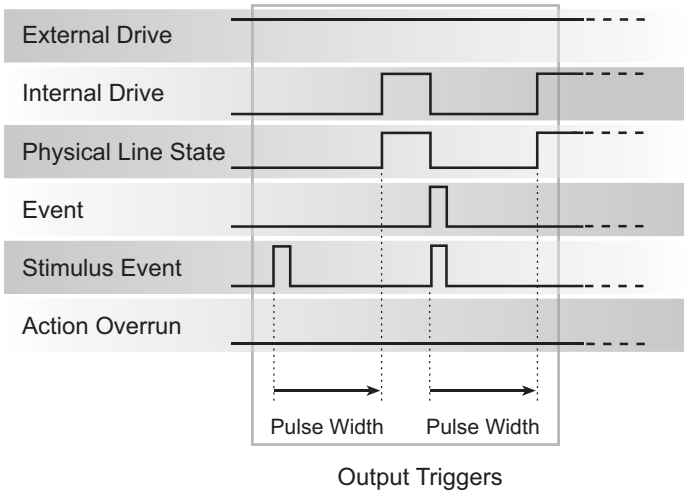
Figure 63: Synchronous input trigger



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 µs or any small nonzero value.
- Action overruns are disabled.

Figure 64: Synchronous output trigger



High-capacitance mode

Overview

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument feature a high-capacitance mode.

Because the source-measure unit (SMU) has the ability to 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 IC (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 Model 2651A 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.

It is important to note that the SMU in the Model 2651A 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 Model 2651A can drive capacitive loads as large as 10 nF. In high-capacitance mode, the SMU can drive a maximum of 50 µF 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 Model 2651A 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.

High-capacitance mode settings apply when operating using the 100 nA through the 100 mA current ranges. When operating using the 1 A, 5 A, 10 A, 20 A, or 50 A ranges, the high-capacitance mode setting will not affect the instrument rise time or current measure settling time, because these ranges are inherently immune to capacitive loads below 50 μF.

The Model 2651A is specified for operating into high Q inductances up to 3 μH on all ranges of voltage and current.

Understanding source settling times

The Model 2651A source-measure unit (SMU) can drive up to 50 μF of a capacitance in high-capacitance mode. In order to accomplish this, the speed of the Model 2651A SMU is reduced. Source settling times increase when high-capacitance mode is enabled. The following table compares the source settling times in normal and high-capacitance modes.

Models 2651A source settling times

Range	Normal mode (typical)	High capacitance mode
1 V	<50 μs	600 μs
10 V	<110 μs	1 ms
20 V	<125 μs	1.5 ms
40 V	<150 μs	7.5 ms

In high-capacitance mode, the frequency compensation capacitance across the measure range resistors increases. This increase leads to longer settling times on some current measure ranges. The same range elements that are used to measure current are used to source current. Therefore, the current limit response times will respond in a similar manner.

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 the Understanding source settling times (on page 3-66) topic. 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 will be 100 seconds (see the following equation).

$$i = C\frac{\Delta t}{\Delta v}$$

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 3-66) 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 taking readings

The following steps outline the procedure to charge and read a capacitor in high-capacitance mode:

1. Set the current limit to a value that is higher than will be used for the measurement (for example, if measuring at 10 µA, the initial current limit can be set for 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 the previous section to understand the impact of high-capacitance mode.
- Test the DUT and the capacitor to determine the best current source and range of output voltages.
- The settling times can vary based on the DUT. It is important to test the limits of the DUT before you use high-capacitance mode.
- Failure to test the DUT for the appropriate current source and output voltages can result in damage to or destruction of the DUT.
- For optimal performance, do not continuously switch between normal mode and high-capacitance mode.
- Before you charge the capacitor, start with 0 (zero) voltage across the capacitor.
- When high-capacitance mode is enabled, a minimum load capacitance of 100 nF is recommended. In absence of this minimum load capacitance, overshoot and/or ringing may occur.

Front panel

To enable high-capacitance mode from the front panel:

1. Press the **CONFIG** key, then select **SRC** > **HIGHC-MODE**.
2. Select **SRC-ENABLE** > **ENABLE**. High-capacitance mode is enabled.
3. Push the **ENTER** key.
4. Press the **EXIT (LOCAL)** key to back out of the menu structure.

Command interface

Turning on High-C mode has the following effects on the SMU settings:

- `smua.measure.autorangei` is set to `smua.AUTORANGE_FOLLOW_LIMIT` and cannot be changed.
- Current ranges below 1 μA are not accessible.
- If `smua.source.limiti` is less than 1 μA, it is raised to 1 μA.
- If `smua.source.rangei` is less than 1 μA, it is raised to 1 μA.
- If `smua.source.lowrangei` is less than 1 μA, it is raised to 1 μA.
- If `smua.measure.lowrangei` is less than 1 μA, it is raised to 1 μA.

Measuring current

The following inputs are required to test leakage using the factory leakage script, as shown in the script example below.

- **SMU:** Sets the Model 2651A source-measure unit to use
- **levelv:** Setting the voltage level to source
- **limiti:** Sets the current limit for discharging or charging the capacitor
- **Sourcedelay:** Solve the following equation to determine the amount of time before taking a current reading:

  $i = C\frac{\Delta v}{\Delta t}$

  Where: i is the limiti setting and current limit
- **measurei:** Sets the current measure range
- **measuredelay:** Defines the delay after the limit is lowered to measure before the measurement is taken

Script example

Use the `smua.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
```

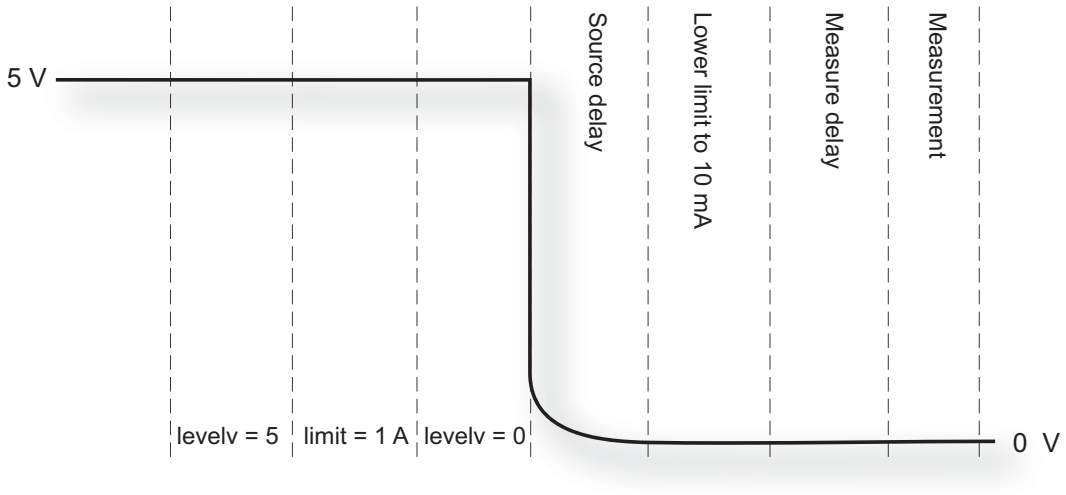
2. To run the `i_leakage_measure()` function in the KIHighC factory script, send:

```
-- 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 along with the magnitude of the voltage step and the current measure range.

Figure 65: Enabling high-capacitance mode



Display operations

Display functions and attributes

The display functions and attributes are used to perform the display operations covered in this section. The following table lists each display function/attribute (in alphabetical order) and cross references it to the section topic where the function/attribute is explained.

Remote commands (on page 5-1) provides additional information about the display functions and attributes.

Cross-referencing functions and attributes to section topics

Function or attribute	Section topic
`display.clear()`	Clearing the display (on page 3-73)
`display.getannunciators()`	Indicators (on page 3-78)
`display.getcursor()`	Cursor position (on page 3-73)
`display.getlastkey()`	Capturing key-press codes (on page 3-81)
`display.gettext()`	Displaying text messages (on page 3-74)
`display.inputvalue()`	Parameter value prompting (on page 3-76)
`display.loadmenu.add()` `display.loadmenu.delete()`	Load test menu (on page 3-79)
`display.locallockout`	LOCAL lockout (on page 3-79)
`display.menu()`	Menu (on page 3-76)
`display.prompt()`	Parameter value prompting (on page 3-76)
`display.screen`	Display screen (on page 3-71)
`display.sendkey()`	Sending key codes (on page 3-81)
`display.setcursor()`	Cursor position (on page 3-73)
`display.settext()`	Displaying text messages (on page 3-74)
`display.smua.digits`	Display resolution (on page 3-72)
`display.smua.measure.func`	Measurement functions (on page 3-72)
`display.trigger.clear()` `display.trigger.wait()`	Display trigger wait and clear (on page 3-72)

Display features

Display screen

Keithley Instruments Model 2651A High Power System SourceMeter® Instrument displays source-measure values and readings or user defined messages. The display screen options include the following:

- **Source-measure, compliance screens:** Displays SMU source and compliance values, and measure readings.
- **User screen:** Display 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.smua.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 4-1/2, 5-1/2 or 6-1/2 digit resolution. Configure the type of resolution displayed by setting the `display.smua.digits` attribute. The following programming example illustrates how to set SMU A for 5-1/2 digit resolution for measured readings:

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

Display trigger wait and clear

The `display.trigger.wait()` function waits for the front panel TRIG key to be pressed, while the `display.trigger.clear()` function clears the trigger event detector.

Display messages

NOTE

Most of the display functions and attributes that are associated with display messaging will automatically select the user screen. The attribute for the display screen is explained in Display screen (on page 3-71).

The reset functions, `reset()` or `smua.reset()`, have no effect on the defined display message or its configuration, but will set the display mode back to the previous source-measure display mode.

The display of the Model 2651A can be used to display user-defined messages. For example, while a test is running, the following message can be displayed on the Model 2651A.

```
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.clear()`, `display.setcursor()`, and `display.settext()` functions (which are explained in the following paragraphs) are overlapped, nonblocking commands. The script will 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 shadow and will update the display as soon as processing time becomes available.

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 will remain 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:

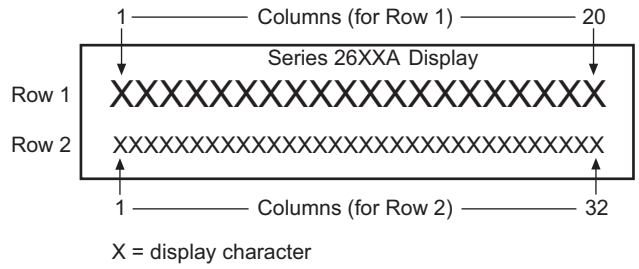
```
display.clear()
```

Cursor position

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

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

Figure 66: Row/column format for display messaging



The function to set cursor position can be used two ways:

```
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 will not be seen. When set to 1, a display character will blink to indicate its position.

The `display.getcursor()` function returns the present cursor position, and can be used three 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: `2.00000e+00 1.00000e+00`

Displaying text messages

To define and display a message, use the `display.settext(`*`text`*`)` function (*`text`* is the text string to be displayed). The message will start 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:

`$N` Starts text on the next line (newline). If the cursor is already on line 2, text will be ignored after the '`$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 alpha-numeric characters, other special characters can be displayed. Refer to Display character codes for a compete listing of special characters and their corresponding codes. 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 would be interpreted as an undefined character code and will be ignored. The message "You owe me" will instead be 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, assume you want to display "Hello" on the top line and "Nate" on the bottom line, and so you 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 can be used in five ways:

```
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:

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

Sending the command without the *row* parameter returns both lines of the display. The $N character code will be included to show where the top line ends and the bottom line begins. The $N character code will be 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 will be returned along with the message. With *embellished* set to false, only the message will be 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

Display messaging can be used along with 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 front panel controls. While displaying an input prompt, the test will pause and wait for the operator to perform the prescribed action from the front panel.

Menu

A user-defined menu can be presented 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 will wait for the operator to select one of the menu items. Rotate the navigation wheel ⟳ to place the blinking cursor on the desired menu item. Items that don't fit in the display area will be displayed by rotating the navigation wheel ⟳ to the right. With the cursor on the desired menu item, press the navigation wheel ⟳ (or the **ENTER** key) to select it.

Pressing the EXIT (LOCAL) key will not abort the script while the menu is displayed, but it will return `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` will be displayed. If Test2 is selected, the message `Running Test2` will be 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

There are two functions to create an editable input field on the user screen at the present cursor position: `display.inputvalue()` and `display.prompt()`.

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

Each of these two functions can be used in four 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:

format 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

default Option to set a default value for the parameter, which will be displayed when the command is sent

min 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`)

max Option to specify maximum limits for the input field

units Text string to identify the units for the value (8 characters maximum), for example: Units text is "V" for volts and "A" for amperes

help 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 will pause and wait for the operator in input the source level. The program will continue 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` will be 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 will not include the "V" units designator and the "`Enter source value`" message.

Indicators

To determine which display indicators are turned on, use the `display.getannunciators()` function. The following programming example illustrates how to determine which display indicators are turned on:

```
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.

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 EDIT.

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 of the above indicators shown in above table may used by the Model 2651A.

Local lockout

The front-panel EXIT (LOCAL) key is used to cancel remote operation and return control to the 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`

Example:

The following programming example illustrates how 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 front panel. Factory tests are preloaded and saved in nonvolatile memory at the factory. They are available in the FACTORY TESTS submenu. Named scripts that have been loaded into the run-time environment can be selected from the SCRIPTS submenu. Refer to Manage scripts (on page 6-3) for additional information.

User tests

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

Adding USER TESTS menu entries

The following function can be used in two ways to add an entry into the USER TESTS submenu:

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

Where:

displayname	Name string to add to the menu
chunk	The code to be executed
memory	Specifies whether the *chunk* and *displayname* parameters are saved in nonvolatile memory; set to one of the following values: `0` or `display.DONT_SAVE` `1` or `display.SAVE` (default is `display.SAVE`)

The *chunk* parameter can be made up of any valid Lua code. With the *memory* parameter set to `display.SAVE`, the entry is saved in nonvolatile memory. Scripts, functions, and variables used in the chunk are not saved when `display.SAVE` is used. Functions and variables need to be saved with the script (see Manage scripts (on page 6-3)). If the script is not saved in nonvolatile memory, it will be lost when the Model 2651A is turned off. See Example 1 below.

Example 1:

Assume a script with a function named “DUT1” has already been loaded into the Model 2651A, 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 chunk, and then save *displayname* and *chunk* in nonvolatile memory:

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

When “Test” is run from the front-panel USER TESTS menu, the function named “DUT1” will execute and the beeper will beep for two seconds.

Now assume you turn the Model 2651A 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 front panel, the beeper will beep, but “DUT1” will not execute because it no longer exists in the runtime environment.

Example 2:

The following command adds an entry called “Part1” to the front-panel USER TESTS submenu for the chunk “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

The following function can be used to delete an entry from the front-panel USER TESTS submenu:

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

Where:

displayname Name to delete from the menu

The following programming example removes the entry named “Part1” from the front-panel USER TESTS submenu:

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

Running a test from the front panel

To run a user, factory, or script test from the 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

Sending key codes

Key codes are provided to remotely simulate pressing a 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). Use the `display.sendkey()` function to perform these actions. 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 front panel key is maintained by the Model 2651A. 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 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 is used to immediately return 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 will return the key code value (see the following table). Remember that 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	display.KEY_NONE		82	display.KEY_ENTER
65	display.KEY_RANGEUP		85	display.KEY_RECALL
68	display.KEY_MENU		86	display.KEY_MEASA
69	display.KEY_MODEA		87	display.KEY_DIGITSA
70	display.KEY_RELA		92	display.KEY_TRIG
71	display.KEY_RUN		93	display.KEY_LIMITA
72	display.KEY_DISPLAY		94	display.KEY_SPEEDA
73	display.KEY_AUTO		95	display.KEY_LOAD
75	display.KEY_EXIT		97	display.WHEEL_ENTER
77	display.KEY_FILTERA		103	display.KEY_RIGHT
78	display.KEY_STORE		104	display.KEY_LEFT
79	display.KEY_SRCA		107	display.WHEEL_LEFT
80	display.KEY_CONFIG		114	display.WHEEL_RIGHT
81	display.KEY_RANGEDOWN			

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 will pause and wait for the operator to press a front-panel key. For example, if the MEAS key is pressed, the function will return the value 86, which is the key code for that key. The key code values are the same as listed in display.getlastkey() (on page 3-82).

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.

Digital I/O

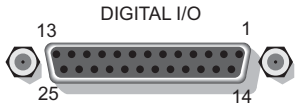
Digital I/O port

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument has a digital input/output port that can be used to control external digital circuitry. For example, a handler that is used to perform binning operations can be used with a digital I/O port.

Port configuration

The digital I/O port, a standard female DB-25 connector (shown below), is located on the rear panel.

Figure 67: Digital I/O port



1 = Digital I/O #1
2 = Digital I/O #2
3 = Digital I/O #3
4 = Digital I/O #4
5 = Digital I/O #5
6 = Digital I/O #6
7 = Digital I/O #7
8 = Digital I/O #8
9 = Digital I/O #9
10 = Digital I/O #10
11 = Digital I/O #11
12 = Digital I/O #12
13 = Digital I/O #13
14 = Digital I/O #14
15-21 = Ground
22 = +5 V
23 = +5 V
24 = Output enable (OE)
25 = +5 V

Connecting cables

Use a cable equipped with a male DB-25 connector (Keithley Instruments part number CA-126-1), or a Model 2600-TLINK cable to connect the digital I/O port to other Keithley Instruments models equipped with a Trigger Link (TLINK).

Digital I/O lines

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.

+5 V output

The digital I/O port provides a +5 V output that is used to drive external logic circuitry. Maximum current output for this line is 250 mA. This line is protected by a self-resetting fuse (one hour recovery time).

Output enable line

The Model 2651A 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 3-88) for more details.

⚠ WARNING

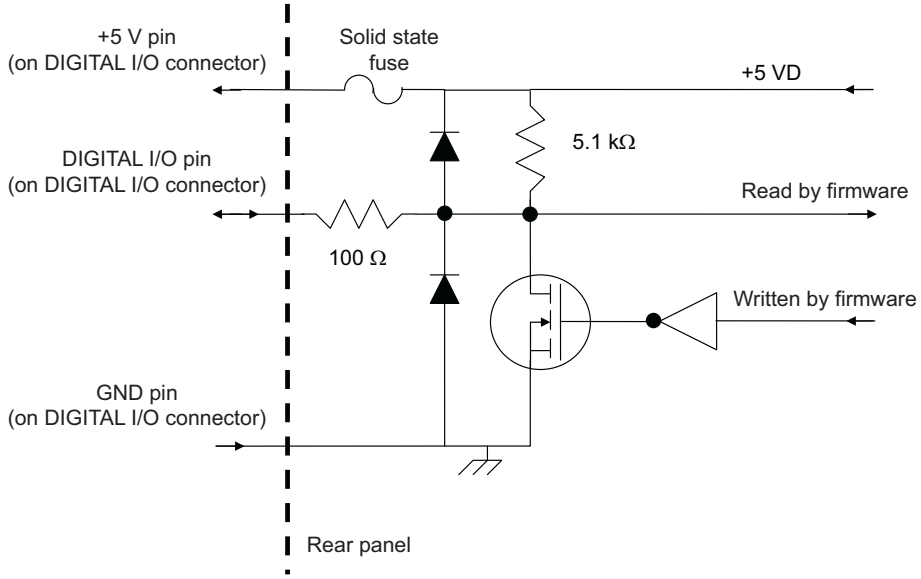
The digital I/O port of the Model 2651A is not suitable for control of safety circuits and should not be used to control a safety interlock. When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages.

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 (~ +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 480 μA per I/O line.

Figure 68: Digital I/O port configuration

DIGITAL I/O INTERFACE:
Connector: 25-pin female D
Input/Output pins: 14 open-drain I/O bits
Absolute maximum input voltage: 5.25 V
Absolute minimum input voltage: -0.25 V
Maximum logic low input voltage: 0.7 V @ +850 µA
Minimum logic high input voltage: 2.1 V @ +570 µA
Maximum source current (flowing out of digital I/O bit): +960 µA
Absolute Maximum sink current (flowing into digital I/O bit): -11.0 A
Maximum Sink Current @ Maximum Logic Low Voltage (0.7 V): -5.0 mA.



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 front panel or over a remote interface.

NOTE

The trigger mode for the line must be set to `digio.TRIG_BYPASS` in order to use the line for digital I/O. See Triggering (on page 3-32) for more information.

To set digital I/O values from the 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 ⏲.
3. Set the decimal value as required to set digital I/O lines within the range of 0 to 16,383 (see the table in Digital I/O bit weighting (on page 3-87)), and then press the **ENTER** key or the navigation wheel ⏲.
4. 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 3-87)), and then press the **ENTER** key or the navigation wheel ⟳.
4. Press the **EXIT (LOCAL)** key as needed to return to the main menu.
5. To remove write protection, repeat the above steps, but enter the original value in step 3.

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	1024	0x0400
12	B12	2048	0x0800
13	B13	4096	0x1000
14	B14	8192	0x2000

Remote digital I/O commands

Commands that control and access the digital I/O port are summarized in the following table. See Remote Commands (on page 5-1) for complete details on these commands. See the following table for decimal and hexadecimal values used to control and access the digital I/O port and individual lines. Use these commands to trigger the Model 2651A 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

The digital I/O lines can be used 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
`digio.readbit(bit)`	Read one digital I/O input line
`digio.readport()`	Read digital I/O port
`digio.writebit(bit, data)`	Write data to one digital I/O output line
`digio.writeport(data)`	Write data to digital I/O port
`digio.writeprotect = mask`	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

Overview

The Model 2651A digital I/O port provides an output enable line for use with a test fixture switch. When properly used, the output of the System SourceMeter® instrument will turn OFF when the lid of the test fixture is opened. See DUT Test Connections (on page 2-42) for important safety information when using a test fixture.

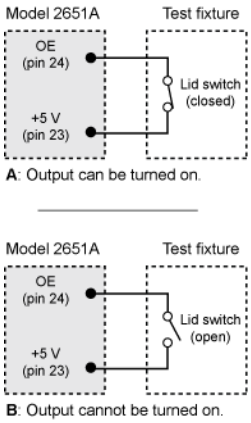
⚠ WARNING

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 Model 2651A is not suitable for control of safety circuits and should not be used to control a safety interlock.

Operation

When enabled, the output of the Model 2651A can only be turned on 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 System SourceMeter® instrument 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 69: Using the output enable line



Front-panel control of output enable

To activate the output enable line:

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. Select **OE_OUTPUT_OFF** to activate the output enable signal causing the source-measure unit (SMU) output to be blocked if the output enable is not asserted (connected to +5 V). Select **NONE** to deactivate the output enable signal so that its state 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
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

When set to `smuX.OE_NONE`, the Model 2651A does not take action when the output enable line is low. When set to `smuX.OE_OUTPUT_OFF`, the instrument will turn its output off as if the `smua.source.output = smua.OUTPUT_OFF` command had been received. The instrument will 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 synchronization lines

The Model 2651A has three synchronization lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

Connecting to the TSP-Link system

The TSP-Link® synchronization lines are built into the TSP-Link connection. Use the TSP-Link connectors located on the back of the Model 2651A. If you are using a TSP-Link network, you do not have to modify any connections. See TSP-Link system expansion interface (on page 6-47) for detailed information about connecting to the TSP-Link system.

Using TSP-Link synchronization lines for digital I/O

Each synchronization line is an open-drain signal. When using the TSP-Link® synchronization 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. See the table in the Digital I/O bit weighting (on page 3-87) topic for digital bit weight values.

Remote TSP-Link synchronization line commands

Commands that control and access the TSP-Link® synchronization port are summarized in the following table. See Remote commands (on page 5-1) for complete details on these commands. See the table in Digital I/O bit weighting (on page 3-87) 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 Model 2651A to read and write to a specific TSP-Link synchronization line or to the entire port.

NOTE

The TSP-Link synchronization lines can be used for both input and output. You must write a 1 to all TSP-Link synchronization lines that are used as inputs.

Remote synchronization line commands

Command	Description
`tsplink.readbit(bit)`	Reads one digital I/O input line.
`tsplink.readport()`	Reads the digital I/O port.
`tsplink.writebit(bit, data)`	Writes data to one digital I/O line.
`tsplink.writeport(data)`	Writes data to the digital I/O port.
`tsplink.writeprotect = mask`	Sets write-protect mask of the digital I/O port.

Programming example

The programming example below illustrates how to set Bit B1 of the digital I/O port high, and then read the entire port value:

```
tsplink.trigger[1].mode = tsplink.TRIG_BYPASS
-- Set Bit B1 high.
tsplink.writebit(1, 1)
-- Read I/O port.
data = tsplink.readport()
```

Section 4

Theory of operation

In this section:

Analog to digital converter 4-1
Source-measure concepts 4-2
Reduction in gain-bandwidth 4-20
Pulse rise times 4-21
Pulse width 4-23

Analog to digital converter

The Model 2651A has two analog-to-digital converters (ADC): an integrating ADC and a fast ADC.

The integrating ADC uses a ratiometric analog-to-digital conversion technique. Depending on the configuration of the integrating ADC, 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 2-22). To help optimize operation of this ADC, the instrument caches the reference and zero values for up to ten of the most recent number of power line cycles. For additional information, see NPLC caching (on page 2-24).

The fast ADC can acquire measurements at speeds up to 1 million samples per second. The fast ADC does not take reference measurements, nor does it have any other of the ratiometric concerns. A reading measurement acquisition buffer allows up to 5,000 readings to be made at the maximum acquisition rate of the fast ADCs. If this buffer is filled, the instrument slows its acquisition rate to the rate at which the instrument can process the data.

Data acquisition takes priority over both source operation and display operation. Sustained high data acquisition rates will cause the display to stop updating. A sustained high data acquisition rate during a sweep or pulse train may slow source operations. This can cause erratic sweep or pulse timing and may lead to triggers being missed (trigger overruns). Use the status model to monitor for trigger overruns. If a high sustained data acquisition rate causes undesirable sweep or pulse timing, reduce the data acquisition rate or reduce the total number of measurements until the desired sweep or pulse timing is achieved.

Source-measure concepts

Overview

This section provides detailed information about source-measure concepts, including:

- Compliance principles (on page 4-2)
- Overheating protection (on page 4-3)
- Operating boundaries (on page 4-5)
- Basic circuit configurations (on page 4-14)
- Guard (on page 4-17)

Compliance principles

Compliance acts as a clamp. If the output reaches the compliance value, the High Power System SourceMeter® Instrument attempts to prevent the output from exceeding that value. This action implies that the source will switch from a V-source to an I-source (or from an I-source to a V-source) when in compliance.

As an example, assume the following:

System SourceMeter® instrument: V_{SRC} = 10 V; I_{CMPL} = 10 mA

Device under test (DUT) resistance: 10 Ω

With a source voltage of 10 V and a DUT resistance of 10 Ω, the current through the DUT should be: 10 V / 10 Ω = 1 A. However, because the compliance is set to 10 mA, the current will 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 Model 2651A output does not exceed the compliance limit, except for the compliance limit conditions described in Source-measure capabilities (on page 2-11).

The Model 2651A can also be set to limit power. This limit can be set in addition to any voltage or current compliance limits specified. The instrument's compliance limit operation changes dependent on the source mode (current or voltage), load, and the configured limits (current, voltage, and power). For additional details on using limits, including load considerations when specifying both a current (or a voltage) limit and a power limit, see the Operating boundaries (on page 4-5) topic.

For information on implementing compliance limits, see Compliance limit (on page 2-12).

Overheating protection

Proper ventilation is required to keep the High Power System SourceMeter® Instrument from overheating. Even with proper ventilation, the Model 2651A can overheat if the ambient temperature is too high or the High Power System SourceMeter® Instrument is operated in sink mode for long periods of time. The High Power System SourceMeter® Instrument has an over-temperature protection circuit that will turn the output off if the instrument overheats. When the over-temperature protection circuit turns the output off, a message indicating this condition is displayed. You will not be able to turn the output back 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 maximum duty cycle (DC_{MAX}), which is computed using the General power equation (on page 4-3) below. Factors such as the ambient temperature, quadrant of operation, and high-power pulse levels (if applicable) affect the maximum duty cycle. Exceeding the calculated maximum duty cycle may cause the temperature protection mechanism to engage. When this happens, an error message displays and the instrument output is disabled until the internal temperature of the instrument is reduced to an acceptable level.

You do not have to be concerned about overheating if all of the following are true:

- The instrument is used as a power source and not a power sink.
- The ambient temperature is ≤ 30° C.
- Extended operating area (EOA) pulsing is not being performed.

However, if any one of these is false, the instrument may overheat if operated in a manner that exceeds the calculated maximum duty cycle, DC_{MAX}.

The maximum duty cycle equation is derived from the power equation below by solving for DC_{MAX}. The general power equation describes how much power an instrument channel can source and sink before the total power cannot be fully dissipated by the instrument's cooling system. This equation incorporates all of the factors that can influence the power dissipated by the instrument.

General power equation

$$\left|(V_{OA} - V_P)(I_P)\right| \; DC_{MAX} + \left|(V_{OA} - V_B)(I_B)\right| \; (1 - DC_{MAX}) \leq (P_{CS} - P_{DER})$$

P_{CS} The maximum power generated in an instrument channel that can be properly dissipated by the instrument cooling system measured in watts. For the Model 2651A, this constant equals 370.

T_{AMB} The ambient temperature of the instrument operating environment.

P_{DER} = $3(T_{AMB} - 30)$

This factor represents the number of watts the instrument is derated when operating in environments above 30° C. The maximum output power is reduced by 3 W per degree C above 30° C.

P_{DER} is 0 when the ambient temperature is below 30° C.

V_{OA} The instrument output amplifier voltage. This constant can be found in the tables below.

V_P The voltage level the instrument is attempting to force while at the pulse level.

When operating in quadrants 1 or 3 (sourcing power), the sign of this voltage must be positive when used in the power equations.

When operating in quadrants 2 or 4 (sinking power), the sign of this voltage must be negative when used in the power equations.

V_B The voltage level the instrument is attempting to force while at the bias level.

When operating in quadrants 1 or 3 (sourcing power), the sign of this voltage must be positive when used in the power equations.

When operating in quadrants 2 or 4 (sinking power), the sign of this voltage must be negative when used in the power equations.

I_P The current flowing through the instrument channel while at the pulse level.

I_B The current flowing through the instrument channel while at the bias level.

Maximum duty cycle equation

$$DC_{MAX} \leq \left[\frac{(P_{CS} - P_{DER}) - |(V_{OA} - V_B)(I_B)|}{|(V_{OA} - V_P)(I_P)| - |(V_{OA} - V_B)(I_B)|} \right] \times 100$$

NOTE

When attempting to determine the maximum duty cycle, where the off state will be 0 V or 0 A:

I_B is 0

I_P and V_P are the voltage and current levels when the instrument is on

Model 2651A maximum duty cycle equation constants

Constant	100 mV range	1 V range	10 V range	20 V range	40 V range
V_{OA}	18.5	18.5	18.5	27	64

Operating boundaries

Source or sink

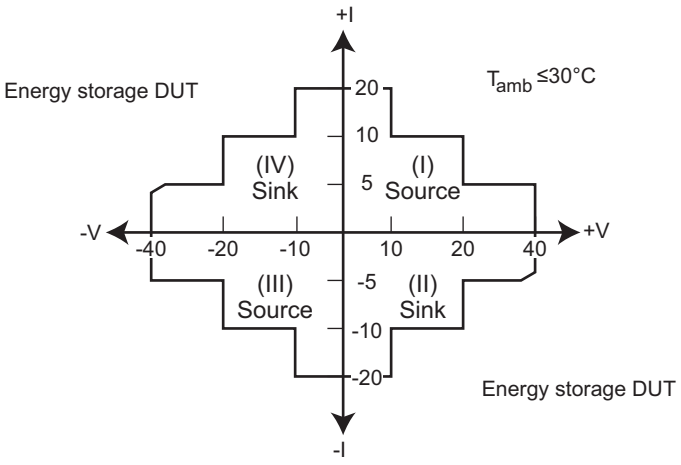
Depending on how it is programmed and what is connected to the output (load or source), the instrument can operate in any of the four quadrants. The four quadrants of operation are shown in the continuous operating boundaries figures. When operating in the first (I) or third (III) quadrant, the instrument is operating as a source (V and I have the same polarity). As a source, the instrument is delivering power to a load.

When operating in the second (II) or fourth (IV) quadrant, the instrument is operating as a sink (V and I 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 or battery, can force operation in the sink region.

Continuous power operating boundaries

The general operating boundaries for Model 2651A continuous power output are shown in the following figure. For derating factors, see the General power equation (on page 4-3). In this drawing, the illustrated voltage and current magnitudes are nominal values. Also note that the boundaries are not drawn to scale.

Figure 70: Model 2651A continuous power operating boundaries

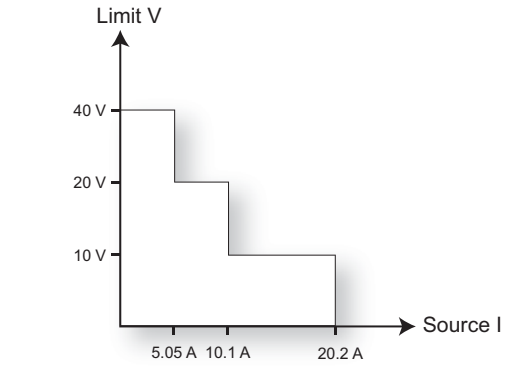


I-source operating boundaries

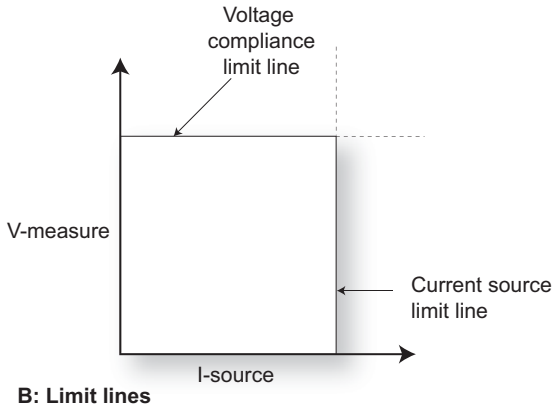
Model 2651A I-source operating boundaries

The following figure shows the operating boundaries for the I-source. Only the first quadrant of operation is shown; operation in the other three quadrants is similar.

Figure 71: Models 2651A I-source boundaries



A: Output characteristics



B: Limit lines

The first graph in the figure, labeled "A: Output characteristics," shows the output characteristics for the I-source. As shown, Model 2651A instruments can continuously output up to 20.2 A at 10 V, up to 10.1 A at 20 V, or up to 5.05 A at 40 V. Note that when continuously sourcing more than 5.05 A, voltage is limited to 20 V, and when sourcing more than 10.1 A, voltage is limited to 10 V.

The second graph in the figure, labeled "B: Limit lines," shows the limit lines for the I-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 Compliance principles (on page 4-2)). These limit lines are boundaries that represent the operating limits of the System SourceMeter 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 are similar.

Load considerations (I-source)

The boundaries within which the High Power System SourceMeter® Instrument operates depend on the load (device-under-test (DUT)) that is connected to its output. The following figure shows operation examples for resistive loads that are 50 Ω and 200 Ω, respectively. For these examples, the System SourceMeter instrument is programmed to source 100 mA, and limit voltage (10 V). In addition, for **A: Normal I-source operation** and **C: I-source operation in power compliance** the System SourceMeter instrument is programmed to limit power (600 mW).

In the following figure's first graph (labeled "A: Normal I-source operation"), the System SourceMeter instrument is sourcing 100 mA to the 50 Ω load and subsequently measures 5 V. As shown, 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).

The second graph in the figure (labeled "B: I-source operation in voltage compliance"), 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 System SourceMeter instrument in voltage compliance. In compliance, the System SourceMeter instrument will not be able to source its programmed current (100 mA). For the 200 Ω DUT, the System SourceMeter instrument will only output 50 mA (at the 10 V limit).

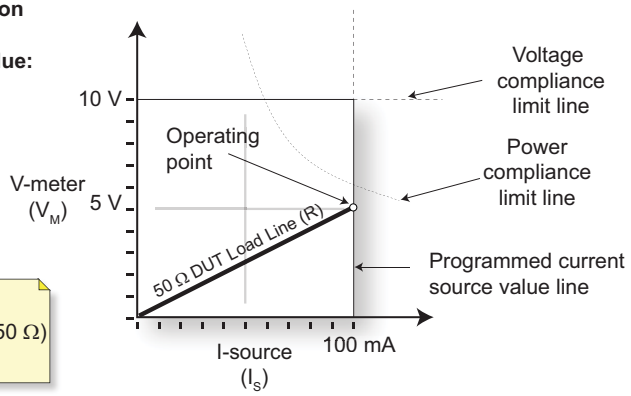
Notice that as resistance increases, the slope of the DUT load line increases. As resistance increases and approaches infinity (open output), the System SourceMeter instrument will source virtually 0 mA at 10 V. Conversely, as resistance decreases, the slope of the DUT load line decreases. At zero resistance (shorted output), the System SourceMeter instrument will source 100 mA at virtually 0 V.

The third graph in the figure (labeled "C: I-source operation in power compliance"), shows what happens if 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 System SourceMeter instrument in power compliance. The System SourceMeter instrument enforces the power compliance limit by setting the voltage compliance limit line to the new power limited voltage compliance limit line setting (which in this case is 6 V). In compliance, the System SourceMeter instrument will not be able to source its programmed current (100 mA). For the 200 Ω DUT, the System SourceMeter instrument will only output 30 mA (at the 6 V limit). In this example, voltage will never exceed the programmed compliance of 10 V, or the programmed power compliance of 600 mW, under any load.

Figure 72: Model 2651A I-source load considerations

A: Normal I-source operation

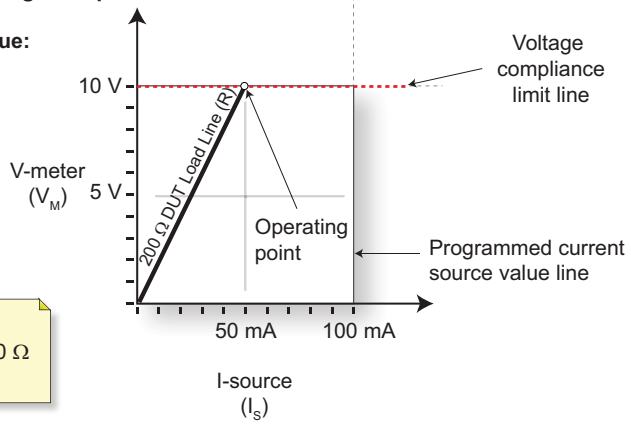
Programmed source value:
100 mA
Actual source value:
100 mA



$V_M = I_S \bullet R$
$= (100\ mA)(50\ \Omega)$
$= \mathbf{5\ V}$

B: I-source operation in voltage compliance

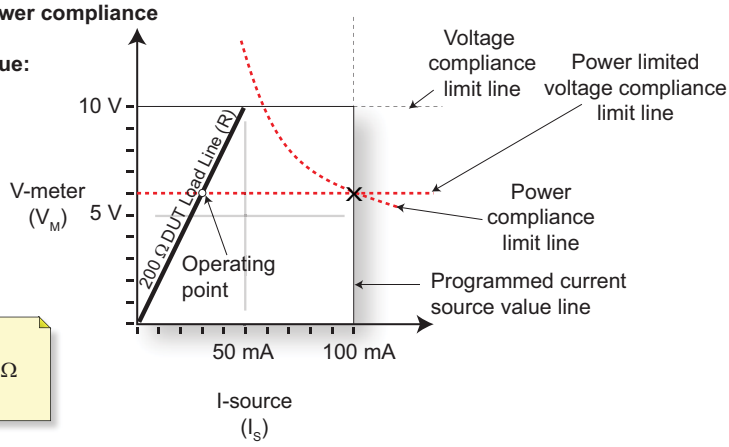
Programmed source value:
100 mA
Actual source value:
50 mA



$I_S = V_M / R$
$= 10\ V / 200\ \Omega$
$= \mathbf{50\ mA}$

C: I-source operation in power compliance

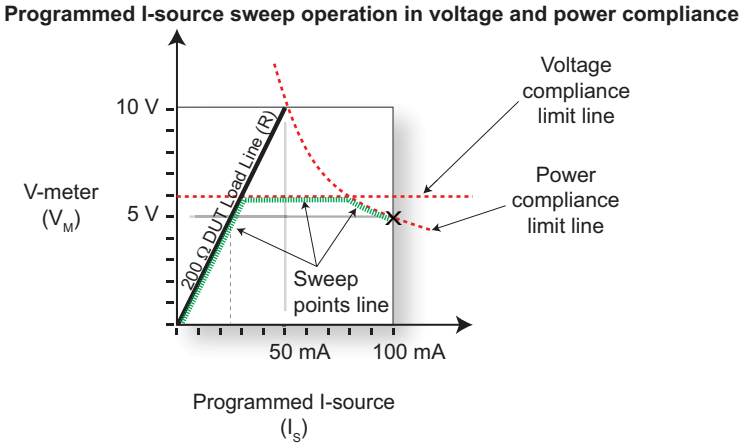
Programmed source value:
100 mA
Actual source value:
30 mA



$I_S = V_M / R$
$= 6\ V / 200\ \Omega$
$= \mathbf{30\ mA}$

The following figure shows a current sweep on a resistive load of 200 Ω. For this example, the System SourceMeter instrument is programmed to sweep current to 100 mA, limit voltage (6 V) and limit power (500 mW). When sweeping, the actual source output will vary according to the programmed source value until the voltage limit is reached. As the figure shows, the output will source the programmed value until placed in voltage compliance at the 6 V limit. The sweep will then continue (programmed I-source values will increase along the green sweep points line), but the output will remain 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 will 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 would be 25 mA (at 5 V).

Figure 73: Model 2651A I-source load considerations while sweeping I

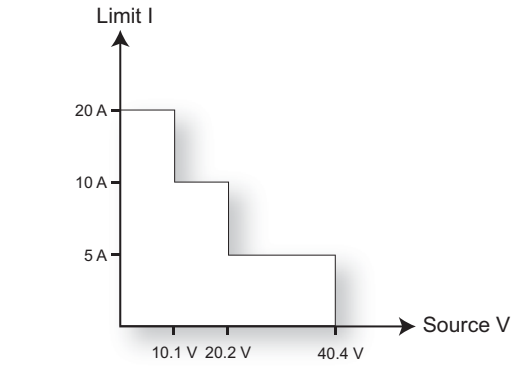


V-source operating boundaries

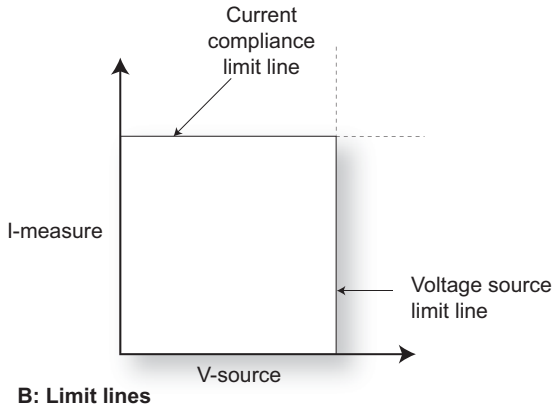
Model 2651A V-source operating boundaries

The following figure shows the operating boundaries for the V-source. Only the first quadrant of operation is shown; operation in the other three quadrants is similar.

Figure 74: Model 2651A V-source boundaries



A: Output characteristics



B: Limit lines

The first graph in the figure (labeled "A: Output characteristics"), shows the output characteristics for the V-source. As shown, the Model 2651A can continuously output up to 40.4 V at 5 A, up to 20.2 V at 10 A, or up to 10.1 V at 20 A. Note that when continuously sourcing more than 10.1 V, current is limited to 10 A and when continuously sourcing more than 20.2 V, current is limited to 5 A.

The second graph in the figure (labeled "B: Limit lines"), shows the limit lines for the V-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 compliance in effect (see Compliance principles (on page 4-2)). These limit lines are boundaries that represent the operating limits of the High Power System SourceMeter® 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 are similar.

Load considerations (V-source)

The boundaries within which the High Power System SourceMeter® Instrument operates depend on the load (device-under-test (DUT)) that is connected to the output. The following figure shows operation examples for resistive loads that are 2 kΩ and 800 Ω, respectively. For these examples, the System SourceMeter instrument is programmed to source 10 V and limit current (10 mA). In addition, for **A: Normal V-source operation** and **C: V-source operation in power compliance** the System SourceMeter instrument is programmed to limit power (60 mW).

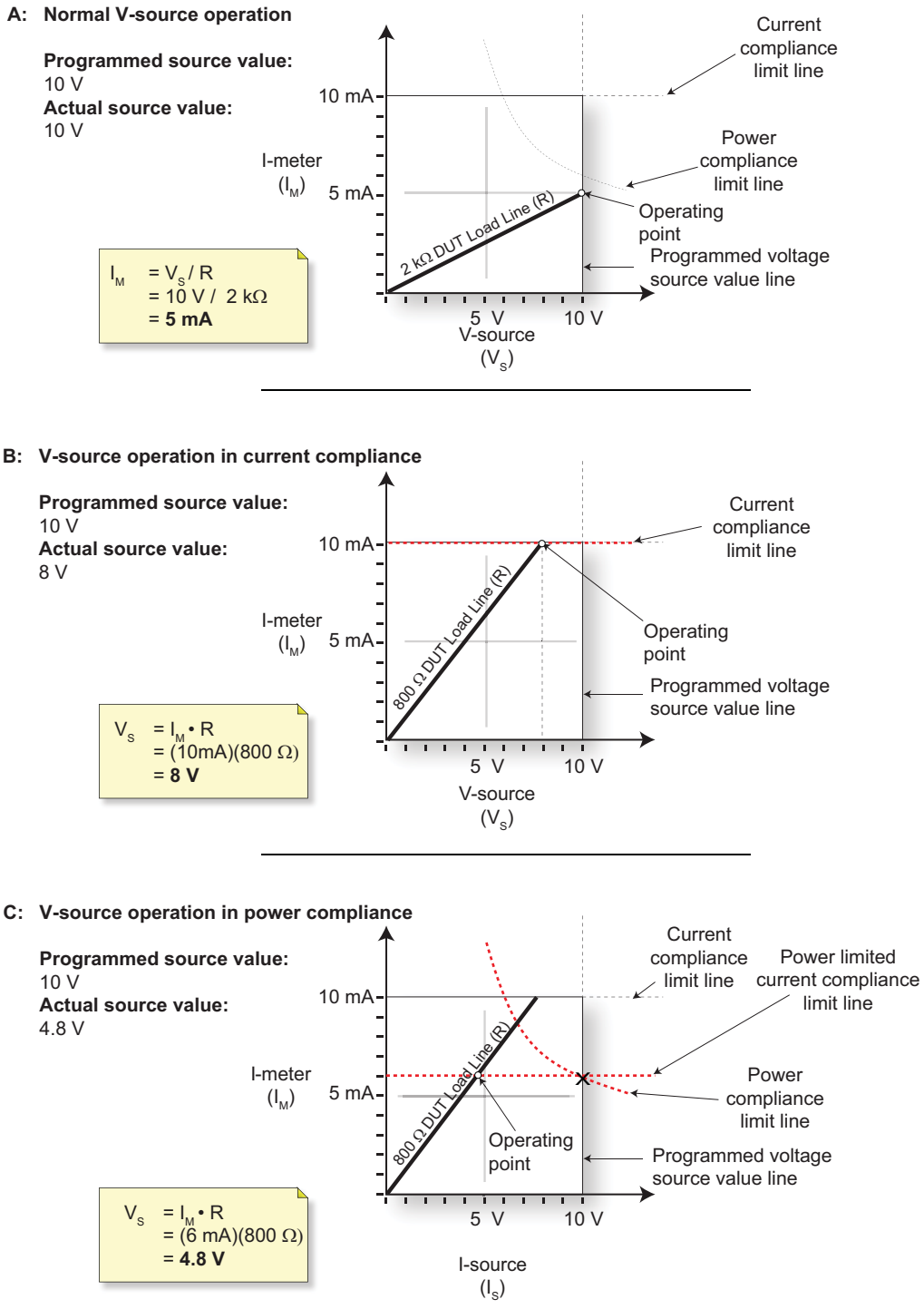
In the following figures first graph (labeled "A: Normal V-source operation"), the System SourceMeter instrument is sourcing 10 V to the 2 kΩ load and subsequently measures 5 mA. As shown, the load line for 2 kΩ intersects the 10 V voltage source line at 5 mA. The current compliance limit and the power compliance limit are not reached (the instrument is not limited through its compliance settings).

The second graph in the figure (labeled "B: V-source operation in current compliance"), shows what happens if 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 System SourceMeter instrument in compliance. When in compliance, the System SourceMeter instrument will not be able to source its programmed voltage (10 V). For the 800 Ω DUT, the System SourceMeter instrument will only output 8 V (at the 10 mA limit).

Notice that as resistance decreases, the slope of the DUT load line increases. As resistance approaches infinity (open output), the System SourceMeter instrument will source virtually 10 V at 0 mA. Conversely, as resistance increases, the slope of the DUT load line decreases. At zero resistance (shorted output), the System SourceMeter instrument will source virtually 0 V at 10 mA.

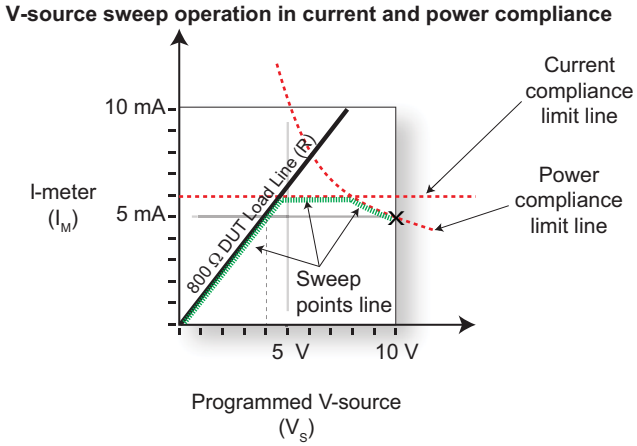
The third graph in the figure (labeled "C: V-source operation in power compliance"), shows what happens if a power limit of 60 mW is applied. As the instrument attempts to output the programmed source value of 10 V, the power compliance limit line is reached placing the System SourceMeter instrument in power compliance. The System SourceMeter instrument 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 6 mA). In compliance, the System SourceMeter instrument will not be able to source its programmed voltage (10 V). For the 800 Ω DUT, the System SourceMeter instrument will only output 4.8 V (at the 5 mA limit). In this example, current will never exceed the programmed compliance of 10 mA, or the programmed power compliance of 60 mW, under any load.

Figure 75: Model 2651A V-source load considerations



The following figure shows a voltage sweep on a resistive load of 800 Ω. For this example, the System SourceMeter instrument is programmed to sweep voltage to 10 V, limit current (6 mA) and limit power (50 mW). When sweeping, the actual source output will vary according to the programmed source value until the current limit is reached. As the figure shows, the output will source the programmed value until placed in current compliance at the 6 mA limit. The sweep will then continue (programmed I-source values will increase along the green sweep points line), but the output will remain 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 (50 mW). At this point, the current and voltage will 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 would be 4 V (at 5 mA).

Figure 76: Model 2651A V-source load considerations while sweeping V



Source I measure I, source V measure V

The High Power System SourceMeter® Instrument can measure the function it is sourcing. When sourcing a voltage, you can measure voltage. Conversely, if you are sourcing current, you can measure the output current. For these measure-source 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. Thus, measuring the source lets you measure the actual output level.

Basic circuit configurations

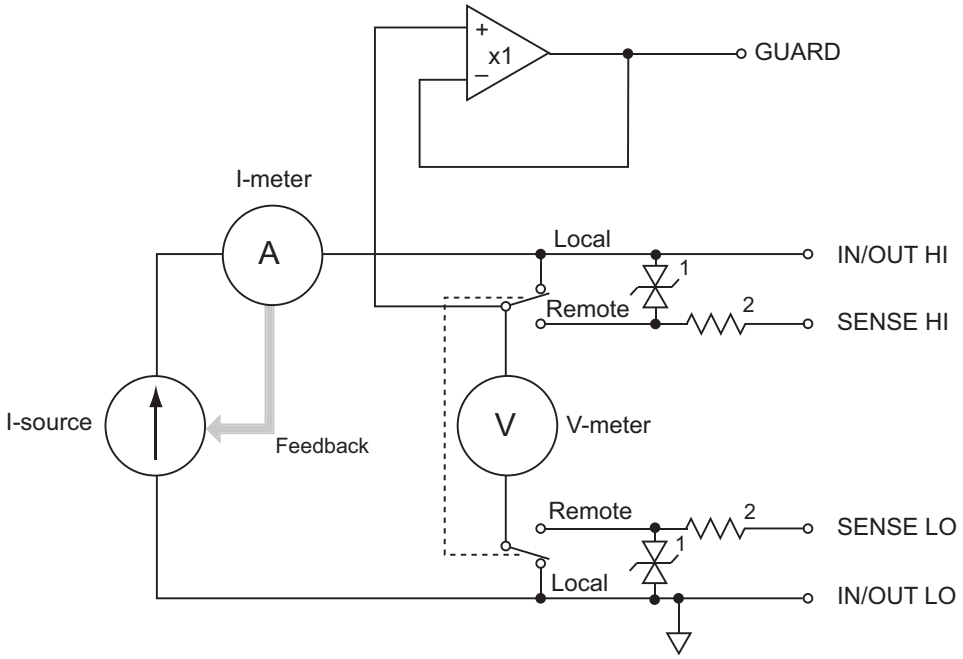
Source I

When configured to source current (I-source) as shown in the figure below, the High Power System SourceMeter® 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 input/output terminals of the System SourceMeter 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 System SourceMeter instrument and the DUT.

The current source does not require or use the sense leads to enhance current source accuracy. With 4-wire remote sensing selected, the sense leads must be connected or incorrect operation will result.

Figure 77: 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.

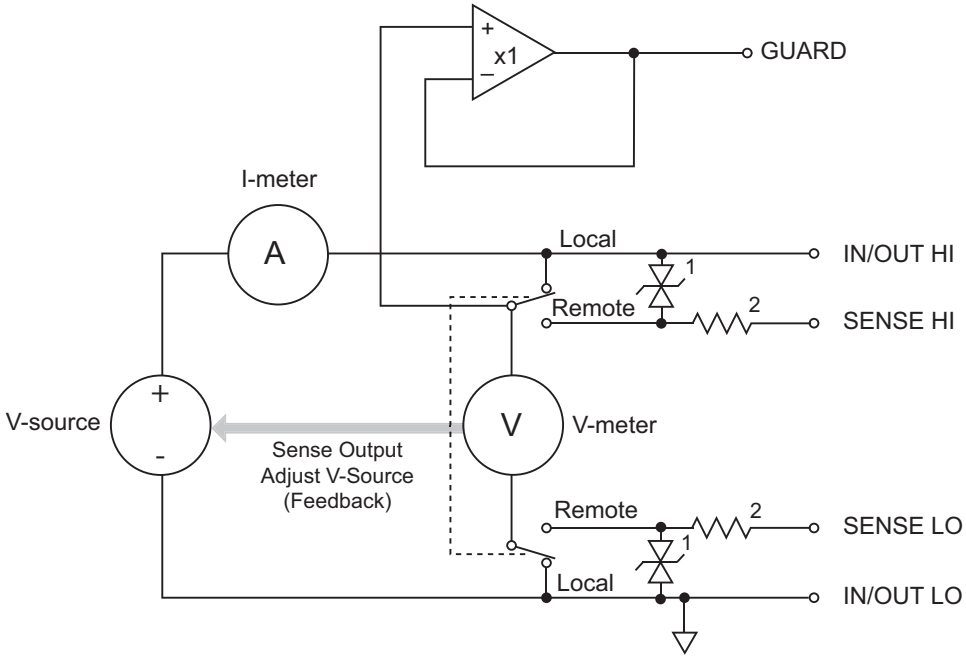
Source V

When configured to source voltage (V-source) as shown in the figure below, the High Power System SourceMeter® Instrument 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 input/output 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.

The voltage error feedback to the V-source is an analog function. The source error amplifier is used to compensate for IR drop in the test leads.

Figure 78: 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.

Measure only (V or I)

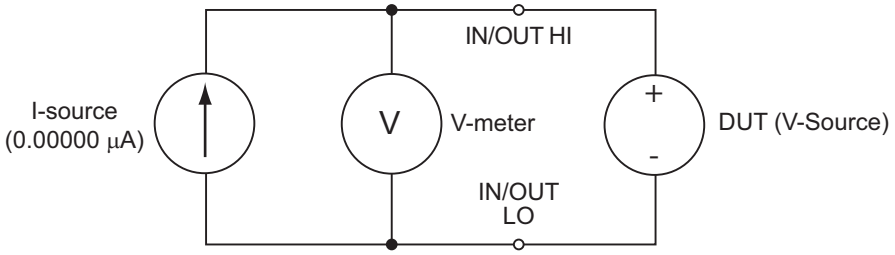
The figure below shows the configurations for using the High Power System SourceMeter® Instrument exclusively as a voltmeter or ammeter. As shown in the figure, the instrument is configured to measure voltage only by setting it to source 0 A and measure voltage.

⚠ CAUTION

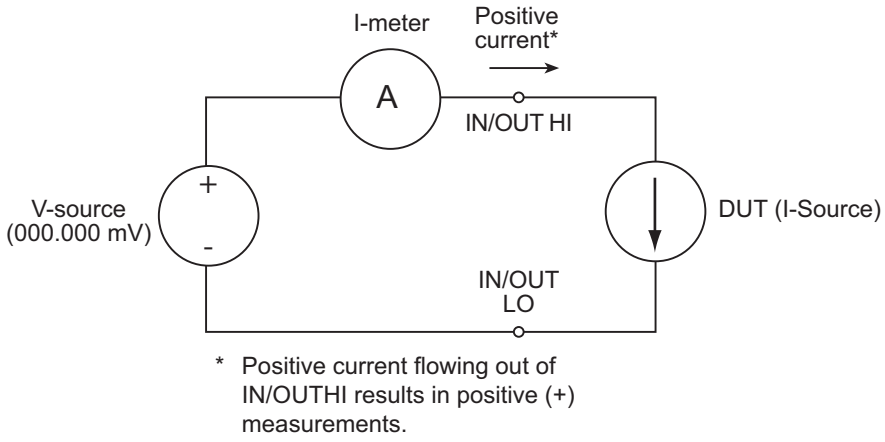
V-compliance must be set to a level that is higher than the measured voltage. If it is not, excessive current will flow into the High Power System SourceMeter® Instrument instrument. This current could damage the instrument. Also, when connecting an external voltage to the I-source, set the output off state to the high-impedance mode. See Compliance limit (on page 2-12) for details.

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 IN/OUT HI to LO.

Figure 79: Measure-only configurations



A. Measure voltage only

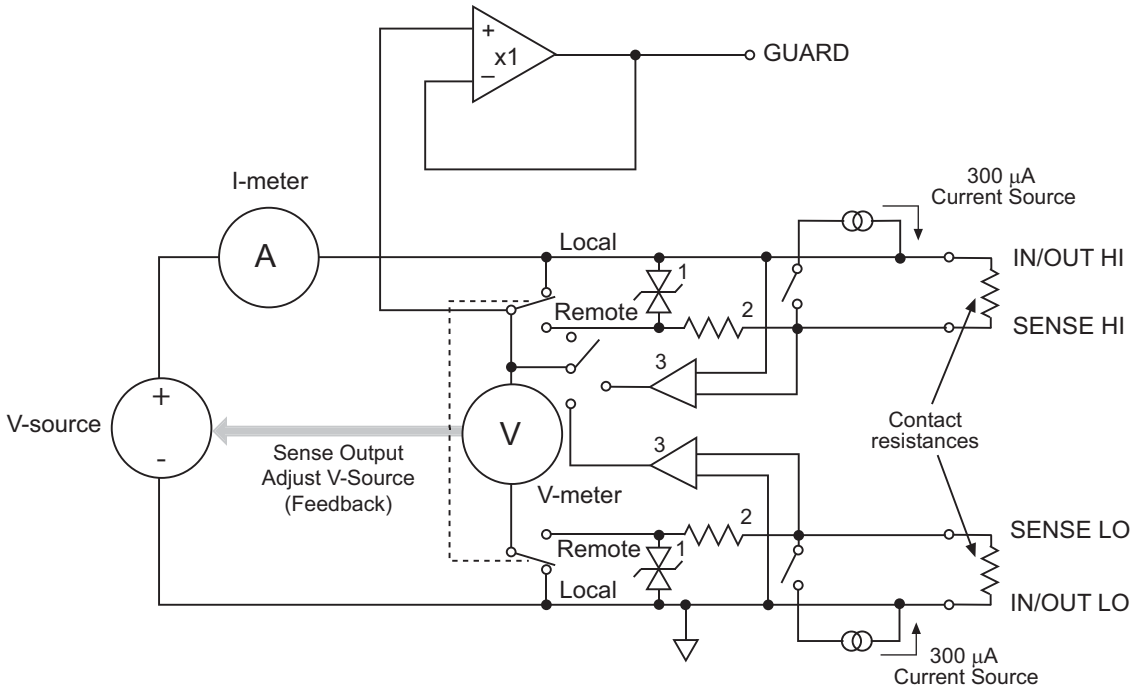


B. Measure current only (uses 2-wire local sense configuration)

Contact check

When a contact check measurement is being performed, two small current sources are switched in 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 80: Contact check circuit 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.
3. High impedance differential amplifier.

Guard

⚠ WARNING

GUARD is at the same potential as output HI. Thus, if hazardous voltages are present at output HI, they are also present at the GUARD terminal.

Guard overview

The driven guard (available at the rear panel GUARD terminals) is always enabled and provides a buffered voltage that is at the same level as the input/output 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 input/output high and low. In the absence of a driven guard, leakage in the external test circuit could be high enough to adversely affect the performance of the High Power System SourceMeter® Instrument.

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

To prevent injury or death, a safety shield must be used to prevent physical contact with a guard plate or guard shield that is at a hazardous potential (>30 V RMS or 42.4 V peak). This safety shield must completely enclose the guard plate or shield and must be connected to safety earth ground. The figure in this topic shows the metal case of a test fixture being used as a safety shield.

NOTE

See Guarding and shielding (on page 2-53) for details about guarded test connections.

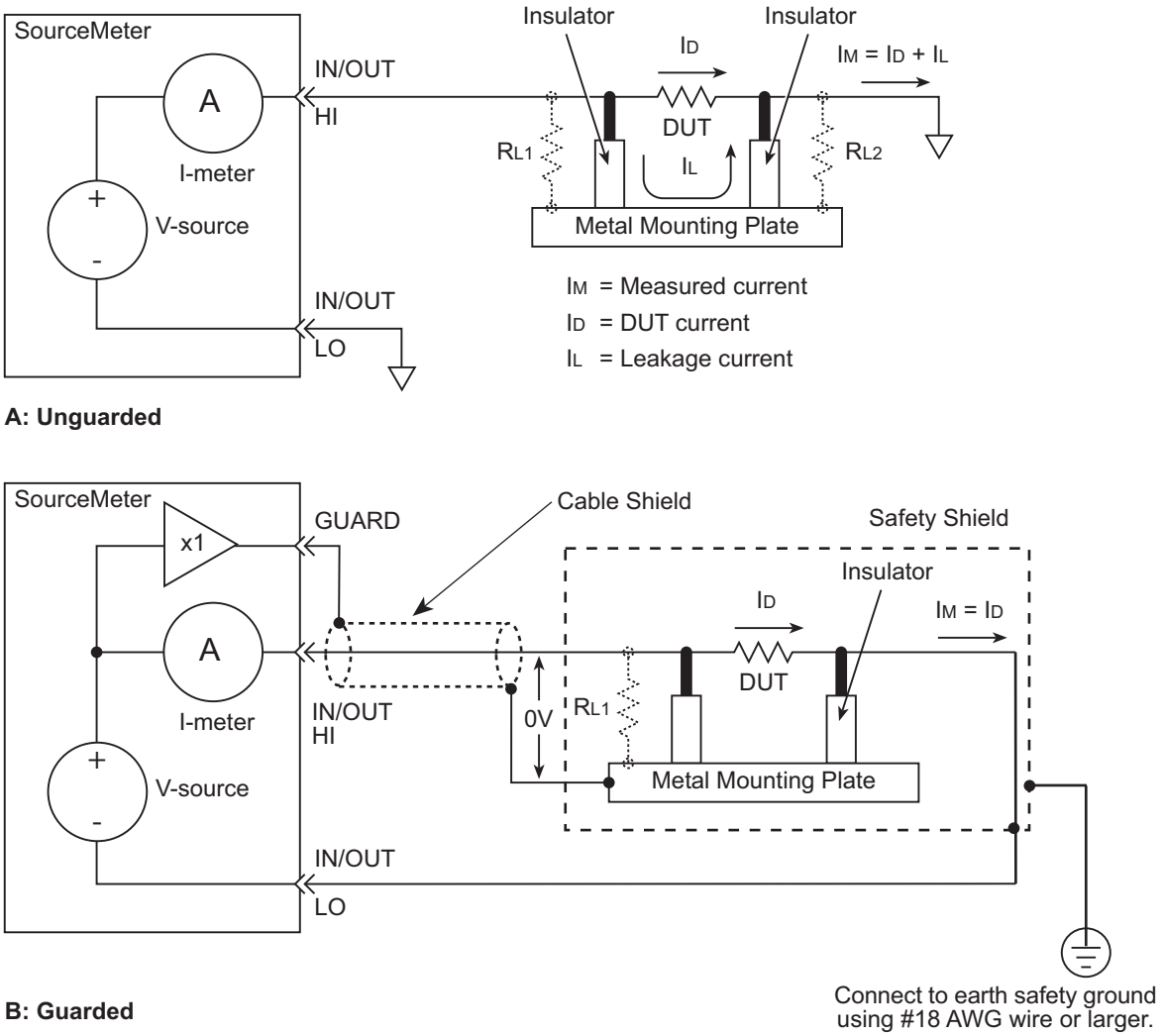
Inside the test fixture, a triaxial cable can be used to extend guard to the device under test (DUT). The center conductor of the cable is used for input/out HI, and the inner shield is used for guard. The outer shield, which is connected to the safety earth ground through the safety shield, is used for input/output LO.

A coaxial cable can be used if the guard potential does not exceed 30 V RMS (42.4 V peak). The center conductor is used for input/output HI, and the outer shield is used for guard. For higher guard potentials, use a triaxial cable.

The figure below shows 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 In/Out LO, adversely affecting the low-current (or high-resistance) measurement of the DUT.

Also in the figure 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 (0V drop), no current can flow through the leakage resistance path. Thus, the SourceMeter instrument only measures the current through the DUT.

Figure 81: Comparison of unguarded and guarded measurements



Cable considerations

The Model 2651A is supplied with a 1 m (3 foot) low inductance, 6 Ω cable. The supplied cable addresses several important considerations:

- Inductance
- Power dissipation
- Voltage drop at high DC current

This cable decreases the inductance as seen from the Model 2651A terminals to the load. This reduction assures specified slew rates and minimal overshoot, which is especially true when operating the Model 2651A in pulse mode at high levels of output current and voltage. The induction equation: V = Ldi/dt illustrates the need for controlling inductance. In this equation, when operating the Model 2651A in source I mode for any given inductance L, the voltage (V) represents the overhead voltage required to be applied across the inductance of the cable and load connections in order to maintain the specified di/dt. With a cable inductance of 5 μH, and a 50 A step in current in 30 μs:

V = Ldi/dt

= (5 μH)(50 A)/30 μs

= 8 V

Therefore 8 V of overhead (4 V for the Hi lead and 4 V for the Lo lead) is required in addition to the device under test (DUT) DC load voltage at 50 A. This 8 V is too high for the remote sensing circuits (3 V Max/lead). In addition, the 10 V range does not have the power supply overhead to provide this 8 V (though the 40 V range does). In this example, the load/cable inductance is therefore too high and must be reduced. Using the supplied Model 2651A 6 Ω cable helps mitigate these problems since it contributes less than 200 nA/meter.

The low resistance cable supplied will dissipate only 3.6 W when operating at 20 A DC; 3.6 W is not excessive for a cable of this size.

The low resistance of the supplied cable will mitigate voltage drop due to operation at high DC currents. This characteristic will leave more overhead voltage available for the actual connection to the DUT.

Be careful to consider each issue outlined above when you make your own connections to your device.

Reduction in gain-bandwidth

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 gain-bandwidth. When the SMU gain-bandwidth is reduced, the settling time of the current source increases.

Use following table to determine the affect of various DUT impedances on the gain-bandwidth when the SMU is operating on each current source range. If the ratio of DUT impedance to current source feedback impedance increases above the indicated 60 kHz ratio, then the settling time will increase beyond the specified times. This means that there is a maximum DUT impedance for each current source range. The settling time on a current source range can increase significantly when measuring DUTs that have an impedance that is higher than that listed in this table.

Current source gain-bandwidth

Range	SMU feedback impedance	60 kHz ratio (DUT / SMU impedance)	Maximum DUT impedance
100 nA	33 MΩ	0.5	33 MΩ
1 µA	396 kΩ	0.5	1.2 MΩ
10 µA	396 kΩ	0.5	396 kΩ
100 µA	3687 Ω	0.5	11 kΩ
1 mA	3687 Ω	0.5	3687 kΩ
10 mA	27 Ω	0.5	81 Ω
100 mA	27 Ω	0.5	27 Ω
1 A	0.045 Ω	0.5	1.1 Ω
5 A	0.045 Ω	0.5	0.23 Ω
10 A	0.005 Ω	0.5	0.065 Ω
20 A	0.005 Ω	0.5	0.065 Ω
50 A	0.005 Ω	0.5	0.065 Ω

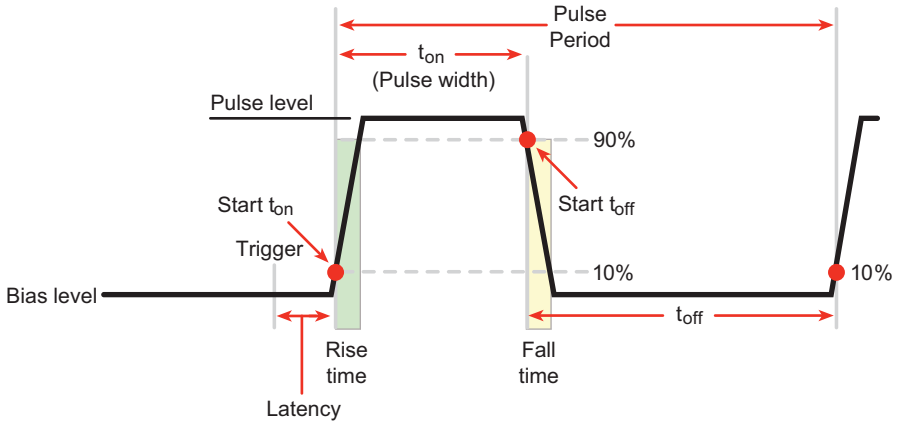
Pulse rise times

The pulse rise time is the leading edge interval for the pulse to go from 10% of maximum value to 90% of maximum value. Pulse fall time is similar but on the trailing edge. For the Model 2651A, pulse rise and fall times can vary depending on the following factors:

- Cable specifications and connection configuration
- Range and setting
- Load and operating mode
- Compliance configuration

Refer to the Model 2651A specifications for details on transient response times, pulse rise times, and source settling times. For latest specifications, go to the Keithley Instruments website (http://www.keithley.com).

Figure 82: Pulse rise and fall times



Cable specifications and connection configuration

Cable length, as well as capacitance and inductance in both the cabling and the test fixture, can affect pulse performance. See Cable considerations (on page 4-20) for pulse related considerations that may affect your connection scheme.

Range and pulse setting

Each range has different specifications for transient response settling times or pulse rise times (and source output settling times). This causes different rise and fall time characteristics depending on the set range. Refer to the Model 2651A specifications for details.

In addition, pulse performance is dependent on the pulse setting as a percent of full scale. For example, a 1 A pulse on the 10 A range (which is 10%) will perform differently than a 10 A pulse on the 10 A range (which is full scale).

Load and operating mode

Settling times for the current source will vary with the resistive load applied, as does the transient response times will vary with the voltage source range. In addition to the load, the times will vary dependent on whether the source-measure unit (SMU) is configured as a voltage source or a current source.

Pulse width

The pulse width is the interval between 10% on the rising (leading) edge to 90% on the falling (trailing) edge. Exceeding the specified pulse width limits can result in short pulses. In addition, the pulse width's jitter can change the pulse width.

NOTE

With respect to pulse width, jitter is the short-term instability of the trailing edge relative to the leading edge.

Refer to the Model 2651A specifications for details on maximum and minimum pulse width limits, pulse width programming resolution, accuracy, and jitter. For latest specifications, go to the Keithley Instruments website (http://www.keithley.com).

Section 5

Remote commands

In this section:

Introduction to remote operation 5-1
About remote commands 5-3
Factory scripts 5-20

Introduction to remote operation

Keithley Instruments Test Script Processor (TSP®) enabled instruments operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you would using any other instrument.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without a controller. You can load a series of instrument control commands into the instrument and 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 particular 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 instrument control commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

Instrument control 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.

Functions

Function-based commands control actions or activities. A function-based command is not always directly related to instrument operation. For example, the `bit.bitand()` function will perform a logical AND operation on two numbers. Each function consists of a function name followed by a set of parentheses ( ). If the function does not have a parameter, the parentheses are left empty. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Example 1

`digio.writeport(15)`	Sets digital I/O lines 1, 2, 3, and 4 high.
`digio.writebit(3, 0)`	Sets line 3 to low (0).
`reset()`	Returns the instrument to its default settings.
`digio.readport()`	Reads the digital I/O port.

Example 2

You can use the results of a function-based command directly or assign to variables to the results for later access. The following code saves the value you enter from the front panel and prints it.

`value = display.inputvalue("+0.00")` `print(value)`	If the operator enters 2.36 from the front panel, the resulting output is: `2.36000e+00`

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, some characteristics of TSP-enabled instruments are the model number (`localnode.model`) and the number of errors in the error queue (`errorqueue.count`).

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

`format.data = 3`	This attribute sets the format of data printed by other commands. Setting this attribute to 3 sets the print format to double precision floating point format.

Example 2: Set an attribute using a constant

`format.data = format.REAL64`	Using the constant `REAL64` instead of 3 also 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 1: 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: `3.00000e+00` This shows that the data format is set to double precision floating point.

Example 2: 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()` command and other related 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: `1.00000e+01` Note that your output might be different if you set your ASCII precision setting to a different value.

Information on scripting and programming

If you need information on using scripts with Model 2651A, see Fundamentals of scripting for TSP (on page 6-1).

If you need information on using the Lua programming language with Model 2651A, see Fundamentals of programming for TSP (on page 6-14).

About remote commands

This section contains an overview of the instrument commands organized into to groups, with a brief description of each group and links to the detailed command descriptions in the Command reference (see "Commands" on page 7-8) in this manual.

Beeper control

The beeper commands allow you to enable or disable and sound the instrument beeper.

beeper.beep() (on page 7-8)
beeper.enable (on page 7-8)

Bit manipulation and logical 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 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. Bitwise logic operations are performed. This means that the logical AND, OR, or XOR operation is performed on Bit B1 of the first number and Bit B1 of the second number. The logical AND, OR, or XOR operation is performed on Bit B2 of the first number and Bit B2 of the second number. 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. These functions can be used 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 of 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 will be truncated. For numbers larger than 4294967295, only the lower 32 bits will be used.

bit.bitand() (on page 7-9)
bit.bitor() (on page 7-9)
bit.bitxor() (on page 7-10)
bit.clear() (on page 7-11)
bit.get() (on page 7-11)
bit.getfield() (on page 7-12)
bit.set() (on page 7-13)
bit.setfield() (on page 7-14)
bit.test() (on page 7-15)
bit.toggle() (on page 7-15)

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 7-38)
dataqueue.CAPACITY (on page 7-39)
dataqueue.clear() (on page 7-39)
dataqueue.count (on page 7-40)
dataqueue.next() (on page 7-41)

Digital I/O

The digital I/O port of the Model 2651A 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 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 7-43)
digio.readport() (on page 7-43)
digio.trigger[N].assert() (on page 7-44)
digio.trigger[N].clear() (on page 7-44)
digio.trigger[N].EVENT_ID (on page 7-45)
digio.trigger[N].mode (on page 7-46)
digio.trigger[N].overrun (on page 7-47)
digio.trigger[N].pulsewidth (on page 7-48)
digio.trigger[N].release() (on page 7-48)
digio.trigger[N].reset() (on page 7-49)
digio.trigger[N].stimulus (on page 7-49)
digio.trigger[N].wait() (on page 7-51)
digio.writebit() (on page 7-51)
digio.writeport() (on page 7-52)
digio.writeprotect (on page 7-53)

Display

display.clear() (on page 7-54)
display.getannunciators() (on page 7-54)
display.getcursor() (on page 7-56)
display.getlastkey() (on page 7-56)
display.gettext() (on page 7-58)
display.inputvalue() (on page 7-59)
display.loadmenu.add() (on page 7-61)
display.loadmenu.catalog() (on page 7-62)
display.loadmenu.delete() (on page 7-63)
display.locallockout (on page 7-64)
display.menu() (on page 7-64)
display.prompt() (on page 7-66)
display.screen (on page 7-67)
display.sendkey() (on page 7-68)
display.setcursor() (on page 7-69)
display.settext() (on page 7-70)
display.trigger.EVENT_ID (on page 7-72)
display.waitkey() (on page 7-74)

Error queue

When errors occur, the error messages are placed in the error queue. Use error queue commands to request error message information.

errorqueue.clear() (on page 7-76)
errorqueue.count (on page 7-76)
errorqueue.next() (on page 7-77)

Event log

You can use the event log to view specific details about LAN triggering events.

eventlog.all() (on page 7-78)
eventlog.clear() (on page 7-78)
eventlog.count (on page 7-79)
eventlog.enable (on page 7-80)
eventlog.next() (on page 7-80)
eventlog.overwritemethod (on page 7-81)

File I/O

You can use the file I/O commands to open, close, 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()`. These commands are responsible for managing file system directories; opening and closing file descriptors; and performing basic I/O operations on a pair of default files, one input and one output.
- Commands that reside in the file descriptors themselves (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

Both slash (/) and backslash (\) are supported as directory separators.

For basic information on navigation and directory listing of files on a flash drive, see File system navigation (on page 2-64).

NOTE

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; as such, the `io.open()` command is 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 7-82)
fileVar:flush() (on page 7-83)
fileVar:read() (on page 7-83)
fileVar:seek() (on page 7-84)
fileVar:write() (on page 7-85)
fs.chdir() (on page 7-88)
fs.cwd() (on page 7-89)
fs.is_dir() (on page 7-89)
fs.mkdir() (on page 7-90)
fs.readdir() (on page 7-91)
fs.rmdir() (on page 7-91)
io.close() (on page 7-99)
io.flush() (on page 7-99)
io.input() (on page 7-100)
io.open() (on page 7-100)
io.output() (on page 7-101)
io.read() (on page 7-101)
io.type() (on page 7-102)
io.write() (on page 7-103)

The following standard I/O commands are not supported at this time:

File	I/O
• *fileVar*:lines()	• io.lines()
• *fileVar*:setvbuf()	• io.popen()

GPIB

This attribute stores the GPIB address.

gpib.address (on page 7-94)

Instrument identification

localnode.description (on page 7-133)
localnode.model (on page 7-134)
localnode.revision (on page 7-138)
localnode.serialno (on page 7-138)

LAN and LXI

The LAN commands has 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 that you must send `lan.applysettings()` in order for the configuration settings to take effect.

The `lan.status.*` commands help you determine the status of the LAN.

The `lan.trigger[N].*` commands allows 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 7-103)
lan.autoconnect (on page 7-104)
lan.config.dns.address[N] (on page 7-104)
lan.config.dns.domain (on page 7-105)
lan.config.dns.dynamic (on page 7-106)
lan.config.dns.hostname (on page 7-107)
lan.config.dns.verify (on page 7-107)
lan.config.duplex (on page 7-108)
lan.config.gateway (on page 7-108)
lan.config.ipaddress (on page 7-109)
lan.config.method (on page 7-110)
lan.config.speed (on page 7-110)
lan.config.subnetmask (on page 7-111)
lan.linktimeout (on page 7-112)
lan.lxidomain (on page 7-113)
lan.nagle (on page 7-113)
lan.reset() (on page 7-114)
lan.restoredefaults() (on page 7-114)
lan.status.dns.address[N] (on page 7-115)
lan.status.dns.name (on page 7-116)
lan.status.duplex (on page 7-117)
lan.status.gateway (on page 7-117)
lan.status.ipaddress (on page 7-118)
lan.status.macaddress (on page 7-118)
lan.status.port.dst (on page 7-119)
lan.status.port.rawsocket (on page 7-119)
lan.status.port.telnet (on page 7-120)
lan.status.port.vxi11 (on page 7-120)
lan.status.speed (on page 7-121)
lan.status.subnetmask (on page 7-121)
lan.timedwait (on page 7-122)
lan.trigger[N].assert() (on page 7-122)
lan.trigger[N].clear() (on page 7-123)
lan.trigger[N].connect() (on page 7-124)
lan.trigger[N].connected (on page 7-124)
lan.trigger[N].disconnect() (on page 7-125)
lan.trigger[N].EVENT_ID (on page 7-126)
lan.trigger[N].ipaddress (on page 7-126)

lan.trigger[N].mode (on page 7-127)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].protocol (on page 7-129)
lan.trigger[N].pseudostate (on page 7-129)
lan.trigger[N].stimulus (on page 7-130)
lan.trigger[N].wait() (on page 7-131)
localnode.description (on page 7-133)
localnode.password (on page 7-134)
localnode.passwordmode (on page 7-135)

Miscellaneous

delay() (on page 7-42)
exit() (on page 7-82)
localnode.autolinefreq (on page 7-132)
localnode.linefreq (on page 7-133)
localnode.password (on page 7-134)
localnode.passwordmode (on page 7-135)
makegetter() (on page 7-140)
makesetter() (on page 7-140)
meminfo() (on page 7-141)
waitcomplete() (on page 7-374)
opc() (on page 7-144)

Parallel script execution

dataqueue.add() (on page 7-38)
dataqueue.CAPACITY (on page 7-39)
dataqueue.clear() (on page 7-39)
dataqueue.count (on page 7-40)
dataqueue.next() (on page 7-41)
node[N].execute() (on page 7-142)
node[N].getglobal() (on page 7-142)
node[N].setglobal() (on page 7-143)
tsplink.group (on page 7-347)
tsplink.master (on page 7-347)
tsplink.node (on page 7-347)

Queries and response messages

The `print()`, `printbuffer()`, and `printnumber()` functions are used to query the instrument and generate response messages. The format attributes control how the data is formatted when those functions are used.

format.asciiprecision (on page 7-86)
format.byteorder (on page 7-87)
format.data (on page 7-87)
localnode.prompts (on page 7-136)
localnode.prompts4882 (on page 7-137)
localnode.showerrors (on page 7-139)
print() (on page 7-144)
printbuffer() (on page 7-145)
printnumber() (on page 7-146)

Reading buffer

bufferVar.appendmode (on page 7-16)
bufferVar.basetimestamp (on page 7-17)
bufferVar.cachemode (on page 7-18)
bufferVar.capacity (on page 7-18)
bufferVar.clear() (on page 7-19)
bufferVar.clearcache() (on page 7-20)
bufferVar.collectsourcevalues (on page 7-21)
bufferVar.collecttimestamps (on page 7-21)
bufferVar.fillcount (on page 7-22)
bufferVar.fillmode (on page 7-23)
bufferVar.n (on page 7-24)
bufferVar.timestampresolution (on page 7-24)
savebuffer() (on page 7-152)
smuX.buffer.getstats() (on page 7-172)
smuX.buffer.recalculatestats() (on page 7-173)
smuX.makebuffer() (on page 7-189)
smuX.nvbufferY (on page 7-208)
smuX.savebuffer() (on page 7-209)

Reset

Resets settings to their default settings.

digio.trigger[N].reset() (on page 7-49)
localnode.reset() (on page 7-137)
reset() (on page 7-151)
timer.reset() (on page 7-333)
smuX.reset() (on page 7-209)
tsplink.reset() (on page 7-349)
trigger.blender[N].reset() (on page 7-335)
trigger.timer[N].reset() (on page 7-343)
tsplink.trigger[N].reset() (on page 7-356)

RS-232

serial.baud (on page 7-163)
serial.databits (on page 7-164)
serial.flowcontrol (on page 7-164)
serial.parity (on page 7-165)
serial.read() (on page 7-166)
serial.write() (on page 7-167)

Saved setups

Use the saved setups commands to manipulate the instrument's configurations. You can restore (or save) configurations from the instrument's nonvolatile memory or an installed USB flash drive. Also, you can use the `setup.poweron` attribute to specify which setup stored in the instrument's nonvolatile memory will be recalled when the unit is turned on.

setup.poweron (on page 7-169)
setup.recall() (on page 7-170)
setup.save() (on page 7-171)

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, and run scripts.

script.anonymous (on page 7-153)
script.delete() (on page 7-154)
script.load() (on page 7-154)
script.new() (on page 7-155)
script.restore() (on page 7-157)
script.user.catalog() (on page 7-158)
scriptVar.autorun (on page 7-159)
scriptVar.list() (on page 7-160)
scriptVar.name (on page 7-160)
scriptVar.run() (on page 7-161)
scriptVar.save() (on page 7-162)
scriptVar.source (on page 7-162)

SMU

localnode.linefreq (on page 7-133)
localnode.autolinefreq (on page 7-132)
smuX.abort() (on page 7-171)
smuX.buffer.getstats() (on page 7-172)
smuX.buffer.recalculatestats() (on page 7-173)
smuX.contact.check() (on page 7-186)
smuX.contact.r() (on page 7-187)
smuX.contact.speed (on page 7-188)
smuX.contact.threshold (on page 7-189)
smuX.makebuffer() (on page 7-189)
smuX.measure.adc (on page 7-190)
smuX.measure.autorangeY (on page 7-191)
smuX.measure.autozero (on page 7-192)
smuX.measure.count (on page 7-194)
smuX.measure.delay (on page 7-195)
smuX.measure.delayfactor (on page 7-196)
smuX.measure.filter.count (on page 7-196)
smuX.measure.filter.enable (on page 7-197)
smuX.measure.filter.type (on page 7-198)
smuX.measure.highcrangedelayfactor (on page 7-199)
smuX.measure.interval (on page 7-199)
smuX.measure.lowrangeY (on page 7-200)
smuX.measure.nplc (on page 7-201)
smuX.measure.overlappedY() (on page 7-201)
smuX.measure.rangeY (on page 7-202)
smuX.measure.rel.enableY (on page 7-204)
smuX.measure.rel.levelY (on page 7-205)
smuX.measure.Y() (on page 7-205)
smuX.measureYandstep() (on page 7-207)
smuX.nvbufferY (on page 7-208)
smuX.reset() (on page 7-209)
smuX.savebuffer() (on page 7-209)
smuX.sense (on page 7-210)
smuX.source.autorangeY (on page 7-211)
smuX.source.compliance (on page 7-213)
smuX.source.delay (on page 7-213)
smuX.source.func (on page 7-214)
smuX.source.highc (on page 7-215)
smuX.source.levelY (on page 7-216)
smuX.source.limitY (on page 7-217)
smuX.source.lowrangeY (on page 7-218)
smuX.source.offlimitY (on page 7-219)
smuX.source.offmode (on page 7-220)
smuX.source.output (on page 7-221)
smuX.source.outputenableaction (on page 7-222)
smuX.source.rangeY (on page 7-223)
smuX.source.settling (on page 7-224)
smuX.source.sink (on page 7-225)
smuX.trigger.arm.count (on page 7-226)
smuX.trigger.arm.set() (on page 7-226)
smuX.trigger.arm.stimulus (on page 7-227)
smuX.trigger.ARMED_EVENT_ID (on page 7-228)

smuX.trigger.autoclear (on page 7-229)
smuX.trigger.count (on page 7-230)
smuX.trigger.endpulse.action (on page 7-230)
smuX.trigger.endpulse.set() (on page 7-231)
smuX.trigger.endpulse.stimulus (on page 7-231)
smuX.trigger.endsweep.action (on page 7-233)
smuX.trigger.IDLE_EVENT_ID (on page 7-233)
smuX.trigger.initiate() (on page 7-234)
smuX.trigger.measure.action (on page 7-235)
smuX.trigger.measure.set() (on page 7-236)
smuX.trigger.measure.stimulus (on page 7-237)
smuX.trigger.measure.Y() (on page 7-238)
smuX.trigger.MEASURE_COMPLETE_EVENT_ID (on page 7-239)
smuX.trigger.PULSE_COMPLETE_EVENT_ID (on page 7-240)
smuX.trigger.source.action (on page 7-241)
smuX.trigger.source.limitY (on page 7-242)
smuX.trigger.source.linearY() (on page 7-243)
smuX.trigger.source.listY() (on page 7-244)
smuX.trigger.source.logY() (on page 7-245)
smuX.trigger.source.set() (on page 7-246)
smuX.trigger.source.stimulus (on page 7-247)
smuX.trigger.SOURCE_COMPLETE_EVENT_ID (on page 7-248)
smuX.trigger.SWEEP_COMPLETE_EVENT_ID (on page 7-249)
smuX.trigger.SWEEPING_EVENT_ID (on page 7-249)

SMU calibration

smuX.cal.adjustdate (on page 7-174)
smuX.cal.date (on page 7-175)
smuX.cal.due (on page 7-175)
smuX.cal.fastadc() (on page 7-176)
smuX.cal.lock() (on page 7-177)
smuX.cal.password (on page 7-177)
smuX.cal.polarity (on page 7-178)
smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.state (on page 7-180)
smuX.cal.unlock() (on page 7-181)
smuX.contact.calibratehi() (on page 7-182)
smuX.contact.calibratelo() (on page 7-183)
smuX.measure.calibrateY() (on page 7-193)
smuX.source.calibrateY() (on page 7-211)

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 7-250)
status.measurement.* (on page 7-252)
status.measurement.buffer_available.* (on page 7-254)
status.measurement.current_limit.* (on page 7-255)
status.measurement.instrument.* (on page 7-256)
status.measurement.instrument.smuX.* (on page 7-257)
status.measurement.reading_overflow.* (on page 7-260)
status.measurement.voltage_limit.* (on page 7-261)
status.node_enable (on page 7-262)
status.node_event (on page 7-264)
status.operation.* (on page 7-266)
status.operation.calibrating.* (on page 7-268)
status.operation.instrument.* (on page 7-269)
status.operation.instrument.digio.* (on page 7-271)
status.operation.instrument.digio.trigger_overrun.* (on page 7-273)
status.operation.instrument.lan.* (on page 7-275)
status.operation.instrument.lan.trigger_overrun.* (on page 7-277)
status.operation.instrument.smuX.* (on page 7-279)
status.operation.instrument.smuX.trigger_overrrun.* (on page 7-281)
status.operation.instrument.trigger_blender.* (on page 7-283)
status.operation.instrument.trigger_blender.trigger_overrun.* (on page 7-284)
status.operation.instrument.trigger_timer.* (on page 7-286)
status.operation.instrument.trigger_timer.trigger_overrun.* (on page 7-287)
status.operation.instrument.tsplink.* (on page 7-289)
status.operation.instrument.tsplink.trigger_overrun.* (on page 7-290)
status.operation.measuring.* (on page 7-292)
status.operation.remote.* (on page 7-293)
status.operation.sweeping.* (on page 7-295)
status.operation.trigger_overrun.* (on page 7-296)
status.operation.user.* (on page 7-298)
status.questionable.* (on page 7-300)
status.questionable.calibration.* (on page 7-301)
status.questionable.instrument.* (on page 7-303)
status.questionable.instrument.smuX.* (on page 7-304)
status.questionable.over_temperature.* (on page 7-306)
status.questionable.unstable_output.* (on page 7-307)
status.request_enable (on page 7-308)
status.request_event (on page 7-310)
status.reset() (on page 7-312)
status.standard.* (on page 7-312)
status.system.* (on page 7-314)
status.system2.* (on page 7-316)
status.system3.* (on page 7-318)
status.system4.* (on page 7-320)
status.system5.* (on page 7-322)

Time

delay() (on page 7-42)
gettimezone() (on page 7-92)
settime() (on page 7-167)
settimezone() (on page 7-168)
timer.measure.t() (on page 7-332)
timer.reset() (on page 7-333)

Triggering

digio.trigger[N].assert() (on page 7-44)
digio.trigger[N].clear() (on page 7-44)
digio.trigger[N].EVENT_ID (on page 7-45)
digio.trigger[N].mode (on page 7-46)
digio.trigger[N].overrun (on page 7-47)
digio.trigger[N].pulsewidth (on page 7-48)
digio.trigger[N].release() (on page 7-48)
digio.trigger[N].reset() (on page 7-49)
digio.trigger[N].stimulus (on page 7-49)
digio.trigger[N].wait() (on page 7-51)
display.trigger.clear() (on page 7-72)
display.trigger.EVENT_ID (on page 7-72)
display.trigger.overrun (on page 7-73)
display.trigger.wait() (on page 7-73)
lan.trigger[N].assert() (on page 7-122)
lan.trigger[N].clear() (on page 7-123)
lan.trigger[N].connect() (on page 7-124)
lan.trigger[N].connected (on page 7-124)
lan.trigger[N].disconnect() (on page 7-125)
lan.trigger[N].EVENT_ID (on page 7-126)
lan.trigger[N].ipaddress (on page 7-126)
lan.trigger[N].mode (on page 7-127)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].protocol (on page 7-129)
lan.trigger[N].pseudostate (on page 7-129)
lan.trigger[N].stimulus (on page 7-130)
lan.trigger[N].wait() (on page 7-131)
smuX.trigger.arm.count (on page 7-226)
smuX.trigger.arm.set() (on page 7-226)
smuX.trigger.arm.stimulus (on page 7-227)
smuX.trigger.ARMED_EVENT_ID (on page 7-228)
smuX.trigger.autoclear (on page 7-229)
smuX.trigger.count (on page 7-230)
smuX.trigger.endpulse.action (on page 7-230)
smuX.trigger.endpulse.set() (on page 7-231)
smuX.trigger.endpulse.stimulus (on page 7-231)
smuX.trigger.endsweep.action (on page 7-233)
smuX.trigger.IDLE_EVENT_ID (on page 7-233)
smuX.trigger.initiate() (on page 7-234)
smuX.trigger.measure.action (on page 7-235)
smuX.trigger.measure.set() (on page 7-236)
smuX.trigger.measure.stimulus (on page 7-237)
smuX.trigger.measure.Y() (on page 7-238)
smuX.trigger.MEASURE_COMPLETE_EVENT_ID (on page 7-239)
smuX.trigger.PULSE_COMPLETE_EVENT_ID (on page 7-240)
smuX.trigger.source.action (on page 7-241)
smuX.trigger.source.limitY (on page 7-242)
smuX.trigger.source.linearY() (on page 7-243)
smuX.trigger.source.listY() (on page 7-244)
smuX.trigger.source.logY() (on page 7-245)
smuX.trigger.source.set() (on page 7-246)
smuX.trigger.source.stimulus (on page 7-247)

smuX.trigger.SOURCE_COMPLETE_EVENT_ID (on page 7-248)
smuX.trigger.SWEEP_COMPLETE_EVENT_ID (on page 7-249)
smuX.trigger.SWEEPING_EVENT_ID (on page 7-249)
trigger.blender[N].clear() (on page 7-333)
trigger.blender[N].EVENT_ID (on page 7-334)
trigger.blender[N].orenable (on page 7-334)
trigger.blender[N].overrun (on page 7-335)
trigger.blender[N].reset() (on page 7-335)
trigger.blender[N].stimulus[M] (on page 7-336)
trigger.blender[N].wait() (on page 7-337)
trigger.clear() (on page 7-338)
trigger.EVENT_ID (on page 7-339)
trigger.timer[N].clear() (on page 7-339)
trigger.timer[N].count (on page 7-339)
trigger.timer[N].delay (on page 7-340)
trigger.timer[N].delaylist (on page 7-341)
trigger.timer[N].EVENT_ID (on page 7-341)
trigger.timer[N].overrun (on page 7-342)
trigger.timer[N].passthrough (on page 7-343)
trigger.timer[N].reset() (on page 7-343)
trigger.timer[N].stimulus (on page 7-344)
trigger.timer[N].wait() (on page 7-345)
trigger.wait() (on page 7-346)
tsplink.trigger[N].assert() (on page 7-350)
tsplink.trigger[N].clear() (on page 7-351)
tsplink.trigger[N].EVENT_ID (on page 7-352)
tsplink.trigger[N].mode (on page 7-352)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].pulsewidth (on page 7-355)
tsplink.trigger[N].release() (on page 7-355)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)

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 7-347)
tsplink.master (on page 7-347)
tsplink.node (on page 7-347)
tsplink.readbit() (on page 7-348)
tsplink.readport() (on page 7-349)
tsplink.reset() (on page 7-349)
tsplink.state (on page 7-350)
tsplink.trigger[N].assert() (on page 7-350)
tsplink.trigger[N].clear() (on page 7-351)
tsplink.trigger[N].EVENT_ID (on page 7-352)
tsplink.trigger[N].mode (on page 7-352)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].pulsewidth (on page 7-355)
tsplink.trigger[N].release() (on page 7-355)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)
tsplink.writebit() (on page 7-359)
tsplink.writeport() (on page 7-359)
tsplink.writeprotect (on page 7-360)

TSP-Net

The TSP-Net module provides a simple socket-like programming interface to Test Script Processor (TSP®) enabled instruments.

tspnet.clear() (on page 7-361)
tspnet.connect() (on page 7-361)
tspnet.disconnect() (on page 7-363)
tspnet.execute() (on page 7-363)
tspnet.idn() (on page 7-364)
tspnet.read() (on page 7-365)
tspnet.readavailable() (on page 7-366)
tspnet.reset() (on page 7-366)
tspnet.termination() (on page 7-367)
tspnet.timeout (on page 7-368)
tspnet.tsp.abort() (on page 7-368)
tspnet.tsp.abortonconnect (on page 7-369)
tspnet.tsp.rbtablecopy() (on page 7-369)
tspnet.tsp.runscript() (on page 7-370)
tspnet.write() (on page 7-371)

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. 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 7-372)
userstring.catalog() (on page 7-372)
userstring.delete() (on page 7-373)
userstring.get() (on page 7-374)

Factory scripts

Introduction

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument 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 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 the Keithley Instruments website (http://www.keithley.com) as a flash firmware upgrade for the Model 2651A. See Upgrading the firmware (on page A-4) for instructions on upgrading the flash firmware of your Model 2651A.

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 "KIPulse"

```
script.factory.scripts.KIPulse()
```

Running a factory script function from the Model 2651A front panel controls

1. Press the **LOAD** key.
2. Select **FACTORY**.
3. Select the function to run and press **RUN** key.

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 loaded back into the Model 2651A 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.

Example:

Retrieve the script listing for a factory script named "KIPulse":

```
script.factory.scripts.KIPulse.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 front panel or the remote interfaces. Details on these functions are provided in the tables that follow. The following functions make up the KISweep factory script:

SweepILinMeasureV() (on page 7-324)
SweepVLinMeasureI() (on page 7-328)
SweepILogMeasureV() (on page 7-326)
SweepVLogMeasureI() (on page 7-330)
SweepIListMeasureV() (on page 7-325)
SweepVListMeasureI() (on page 7-329)

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 will 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 1ms 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 5-23) 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:

Config functions:
Source autorange (I and V) off
Measure autorange (I and V) off
Measure NPLC < ton
Measure autozero OFF or ONCE

InitiatePulseTest functions:
Output on
There is enough free space in the buffer
Buffer append mode is on when pulse train is >1 point

Use the KIPulse simple pulse functions (on page 5-24) 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 7-25)
ConfigPulseVMeasureI() (on page 7-31)
ConfigPulseIMeasureVSweepLin() (on page 7-27)
ConfigPulseVMeasureISweepLin() (on page 7-34)
ConfigPulseIMeasureVSweepLog() (on page 7-29)
ConfigPulseVMeasureISweepLog() (on page 7-36)
QueryPulseConfig() (on page 7-149)
InitiatePulseTest() (on page 7-97)

KIPulse simple pulse functions

PulseIMeasureV() (on page 7-147)
PulseVMeasureI() (on page 7-148)

Advanced features for KIPulse tag parameter pulse functions

Variable off time between pulses in a pulse train

The KIPulse "Configure" functions will 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. The following should be noted:

- If *toff* is passed as a number or only a single value is used in the table, it will be 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 will 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 HighC mode is active. Output is generally at a non-zero voltage prior to 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 7-95)
i_leakage_threshold() (on page 7-96)

KIParlib factory script

The KIParlib factory script is made up of two functions: `gm_vsweep()` and `gm_isweep()`.

gm_vsweep() (on page 7-93)
gm_isweep() (on page 7-92)

KISavebuffer factory script

The KISavebuffer script has one function: `savebuffer()`.

savebuffer() (on page 7-152)

Section 6

Instrument programming

In this section:

Fundamentals of scripting for TSP 6-1
Fundamentals of programming for TSP 6-14
Using Test Script Builder (TSB) 6-37
Working with TSB Embedded 6-37
Advanced scripting for TSP 6-39
TSP-Link system expansion interface 6-47
TSP-Net 6-60

Fundamentals of scripting for TSP

NOTE

The next few sections of the documentation describe scripting and programming features of the instrument. You do not need to review this information if you do not need to use 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. In the instrument, the Test Script Processor (TSP®) scripting engine processes and runs scripts.

Scripts offer several advantages over sending individual commands from the control computer:

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs faster and more efficiently when processing scripts.
- 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.

While 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.

This section describes how to create, load, modify, and run scripts.

What is a script?

A script is a collection of instrument control commands and programming statements. Scripts that you create are referred to as **user scripts**.

Your scripts can be interactive. Interactive scripts display messages on the front panel of the instrument to 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 the nonvolatile memory.

The runtime environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to remember a value as long as the instrument is turned on and the variable has not been assigned a new value. After scripts are loaded into the runtime environment, you can run and manage them from the front panel of the instrument or from a computer.

Nonvolatile memory is where information is stored even when the instrument is turned off. To save a script when the instrument is turned off, you must save it to nonvolatile memory. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

Information in the runtime environment is lost when the instrument is turned off.

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.
- They are loaded into the runtime environment.

For detail on the amount of memory available in the runtime environment, see Memory considerations for the runtime environment (on page 6-46).

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 6-10).

What can be included in scripts?

Scripts can include combinations of commands and Lua code. Commands tell the instrument to do one thing and are described in the Command reference (see "Commands" on page 7-8). Lua is a scripting language that is described in Fundamentals of programming for TSP (on page 6-14).

Commands that cannot be used in scripts

Though an 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

Manage scripts

This section describes how to create scripts by sending commands over the remote interface and using TSB Embedded. Topics include:

- Tools for managing scripts (on page 6-3)
- Create and load a script (on page 6-3)
- Run scripts (on page 6-6)
- Scripts that run automatically (on page 6-6)
- Save the anonymous script as a named script (on page 6-8)

Tools for managing scripts

To manage scripts, you can send messages to the instrument, use your own development tool or program, use the Keithley Test Script Builder Integrated Development Environment (TSB IDE), or use TSB Embedded.

The TSB IDE is a programming tool that is included on the Product Information CD-ROM that came with your Model 2651A. You can use it to create, modify, debug, and store Test Script Processor (TSP®) scripting engine scripts. For more information about using the TSB IDE, see Using Test Script Builder (TSB) (on page 6-37).

TSB Embedded is a tool with a reduced set of features from the complete Keithley TSB IDE. TSB Embedded has both script-building functionality and console functionality (single-line commands). It is accessed from an Internet browser. See Working with TSB Embedded (on page 6-37) for information on script manipulation using TSB Embedded. The Model 2651A User's Manual contains additional information on using TSB Embedded.

Create and load a script

You create scripts by loading them into the instrument's runtime environment. 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 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 other variables that reference it.
- Sending revised scripts with different names will not remove previously loaded scripts.
- Unlike other 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 6-10).

Load a script by sending commands over the remote interface

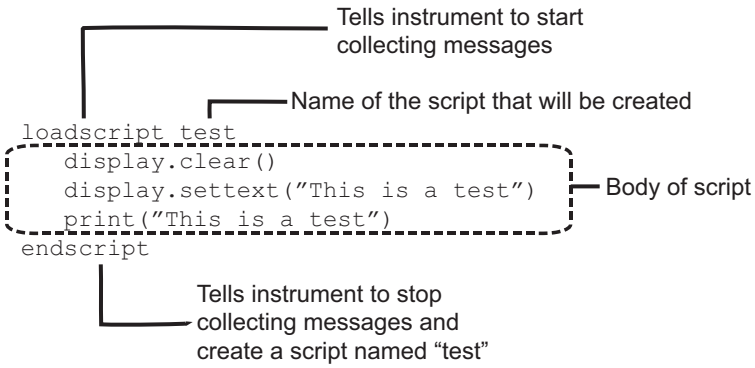
You can send commands over the remote interface instead of using TSB Embedded. To load a script over the remote interface, you can use the `loadscript`, `loadandrunscript`, and `endscript` commands.

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 would run them immediately as individual commands.

The `endscript` command tells the instrument to compile the 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” will be displayed on the instrument and sent to the computer.

Figure 83: 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 (see Lua reserved words (on page 6-16)).
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 6-6).

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`. For information on using TSB Embedded, select the Help button on a web page or the Help option from the navigational pane on the left side.

You can create a script from the instrument web page with TSB Embedded. When you save the script in TSB Embedded, 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. Line numbers are automatically assigned.
3. Click **Save Script**. The name is added to the User Scripts list on the left.

Run scripts

This section describes how to run the anonymous and named scripts.

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 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

Any named script that is in the runtime environment can be run using one of the following commands:

- *`scriptVar`*`()`
- *`scriptVar`*`.run()`

Where: *`scriptVar`* is the user-defined name of the script.

To run a named script from TSB Embedded, select the script from the User Scripts list and click **Run**.

When a script is named, it can be accessed using the global variable *`scriptVar`*.

Example: Run a named script

`test3()`	If the script `test3` is loaded into the runtime environment, the instrument executes `test3`.

Scripts that run automatically

The autorun scripts and the autoexec script run automatically when the instrument is turned on.

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 settings the script's `.autorun` attribute 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 script's `.autorun` attribute 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 6-11) 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 marked 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 6-3).

NOTE

You need to save the autoexec script to nonvolatile memory to save the script when the instrument is turned off. See Save a user script to nonvolatile memory (on page 6-11) for more detail.

Example: Autoexec script with 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: TSB Embedded

`display.clear()` `display.settext("Hello from autoexec")`	In the TSP Script box, enter `autoexec`. Enter the code in the entry box. Click **Save Script**. Creates a new script that clears the display when the instrument is turned on and displays "Hello from autoexec".

Save the anonymous script as a named script

To save the anonymous script as a named script:

1. To name the script, send the following command:

```
script.anonymous.name = "myTest"
```

   Where *myTest* is the name of the script

2. Send the command `script.anonymous.save()` to save *myTest* to nonvolatile memory.

Retrieve a user script

There are several ways to retrieve the source code of a user script:

- Line by line: Use *scriptVar*`.list()` to retrieve the source code line by line.
- Entire script: Use the `print(`*scriptVar*`.source)` command to retrieve the script source code as a single string.
- Use TSB Embedded.

After retrieving a script, see Create and load a script (on page 6-3) for information on recreating the script and loading it back into the instrument.

NOTE

To get a list of scripts that are in nonvolatile memory, see script.user.catalog() (on page 7-158).

Retrieve source code one line at a time

To retrieve the source code one line at a time, use the *scriptVar*`.list()` command. The output for this method 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

`test7.list()`	Retrieve the source of a script named "test7".

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(test1.source)`	Retrieve the source of a script named "test1".

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 6-37) for more information.

Script example: Retrieve the content of scripts

This set of examples:

- Retrieves the source of a script using *`scriptVar`*`.source`
- Retrieves the source of a script using *`scriptVar`*`.list()`

Example: Retrieve the content of a script with scriptVar.source

`print(scriptVarTest.source)`	Request a listing of the source of `scriptVarTest`. The instrument outputs the following (note that the loadscript and endscript commands are not included).
	
	Output:
	`listTones = {100, 400, 800}`
	`for index in listTones do`
	`  beeper.beep(.5, listTones[index])`
	`end`

Example: Retrieve the content of a script with scriptVar.list()

`scriptVarTest.list()`	Request a listing of the source of `scriptVarTest`. The instrument outputs the following (note that the loadscript and endscript commands are included).
	
	Output:
	`loadscript scriptVarTest`
	`listTones = {100, 400, 800}`
	`for index in listTones do`
	`  beeper.beep(.5, listTones[index])`
	`end`
	`endscript`

Working with scripts in nonvolatile memory

The previous section, Fundamentals of scripting for TSP (on page 6-1), described working with scripts, primarily in the runtime environment.

Scripts can also be stored in nonvolatile memory. Information in nonvolatile memory is stored even when the instrument is turned off. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

The runtime environment and nonvolatile memory are separate storage areas in the instrument. The runtime environment is wiped clean 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.

This section describes how to work with the scripts in nonvolatile memory, including how to:

- Save a user script to nonvolatile memory (on page 6-11)
- Retrieve a user script (on page 6-8)
- Restore a script to the runtime environment (on page 6-43)
- Delete user scripts (on page 6-11)

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:

1. Create and load a named script (see Create and load a script (on page 6-3)).
2. Do one of the following:
   - Send the command *scriptVar*.save(), where *scriptVar* is the name of the script.
   - In TSB Embedded, click **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 drive:

NOTE

When you save a script to a USB flash drive, you do not need to specify a file extension; `.tsp` is automatically added. If you do specify a file extension, it must be `.tsp`; an error will occur if you use any other file extension.

1. Load a script (see Create and load a script (on page 6-3)).
2. Send the command: *scriptVar*.save("/usb1/*filename*.tsp"), where: *scriptVar* is the variable referencing the script and *filename*.tsp is a name of the file.

NOTE

You can also use TSB Embedded to save a script to a USB flash drive (or any accessible drive) installed on your computer. From TSB Embedded, load the script and click **Export to PC**.

Delete user scripts

NOTE

These steps remove a script from nonvolatile memory. To completely remove a script from the system, there are additional steps you must take. See Delete user scripts from the instrument (on page 6-45).

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.

If you are using TSB Embedded, to delete a script from nonvolatile memory:

1. In TSB Embedded, select the script from the User Scripts list.
2. Click **Delete**. There is no confirmation message.

Example: Delete a user script from nonvolatile memory

`script.delete("test8")`	Delete a user script named "test8" from nonvolatile memory.

Programming example

Interactive script

An interactive script prompts the operator to input values using the instrument front panel. The following example script uses display messages to prompt the operator to:

- Enter the digital I/O line on which to output a trigger
- Enter the output trigger pulsewidth

After the output trigger occurs, the front display displays a message to the operator.

When an input prompt is displayed, the script waits until the operator inputs the parameter or presses the **ENTER** key.

```
-- Clear the display.
display.clear()

-- Prompt user for digital I/O line on which to output trigger.
myDigioLine = display.menu(
   "Select digio line", "1 2 3 4 5 6 7 8 9")

-- Convert user input to a number.
intMyDigioLine = tonumber(myDigioLine)

-- Prompt user for digital output trigger mode.
myDigioEdge = display.menu(
   "Select digio mode", "Rising Falling")
if myDigioEdge == "Rising" then
   edgeMode = digio.TRIG_RISING
else
   edgeMode = digio.TRIG_FALLING
end

-- Prompt user for output trigger pulsewidth.
myPulseWidth = display.prompt(
   "000.0", "us", "Enter trigger pulsewidth", 10, 10, 100)

-- Scale the entered pulsewidth
myPulseWidth = myPulseWidth * 1e-6

-- Generate the pulse.
digio.trigger[intMyDigioLine].mode = edgeMode
digio.trigger[intMyDigioLine].pulsewidth = myPulseWidth
digio.trigger[intMyDigioLine].assert()

-- Alert the user through the display that the
-- output trigger has occurred.
display.setcursor(1, 1)
display.settext(
   "Trigger asserted $Non digital I/O line " .. intMyDigioLine)

-- Wait five seconds and then return to main screen.
delay(5)
display.screen = display.SMUA
```

Fundamentals of programming for TSP

Introduction

To conduct a test, a computer (controller) is programmed to send sequences of commands to an instrument. The controller orchestrates the actions of the instrumentation. The controller is typically programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

To take advantage of the advanced features of the instrument, you can add programming commands to your scripts. Program statements 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 http://www.lua.org. Another source of useful information is the Lua users group http://lua-users.org, created for and by users of Lua programming language.

Comments

Comments start anywhere outside a string with a double hyphen (`--`). If the text immediately after a double hyphen (`--`) is anything other than double left brackets (`[[`), the comment is a short comment, which continues only until the end of the line. If double left brackets (`[[`) follow the double hyphen (`--`), it is a long comment, which continues until the corresponding double right brackets (`]]`) close the comment. Long comments may continue for several lines and may contain nested `[[ . . . ]]` pairs. The table below shows how to use code comments.

Using code comments

Type of comment	Comment delimiters	Usage	Example

Short comment	`--`	Use when the comment text is short enough that it will not wrap to a second line.	`--Disable the front-panel LOCAL key.`
			
			`display.locallockout = display.LOCK`

Long comment	`--[[ ]]`	Use when the comment text is long enough that it wraps to additional lines. The double left brackets signal the beginning of the multi-line comment, and the double right brackets signal the end of the comment.	`--[[Displays a menu with three menu items. If the second menu item is selected, the selection will be given the value Test2.]]`
			
			`selection = display.menu(`
			`"Sample Menu", "Test1 Test2 Test3")`
			`print(selection)`

Lua reserved words

You cannot use the following 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

In addition to the Lua reserved words, you also cannot use command group names as variable names. Doing so will result in the loss of use of the command group. For example, if you send the command digio = 5, you cannot access the digio.* commands until you cycle the power to the instrument. These groups include:

Command group names	
beeper	lan
bit	localnode
dataqueue	opc
delay	reset
digio	smua
display	status
eventlog	timer
errorqueue	trigger
exit	tsplink
format	tspnet
fs	userstring
gpib	waitcomplete
io	

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 used.

NOTE

Do not create variable names that are the same as the base names of Model 2651A Instrument Control Library (ICL) commands. Doing so will result in the loss of use of those commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until the power to the instrument is turned off and then back on.

Variables can be one of the following types.

Variable types and values

Variable type returned	Value	Notes
`"nil"`	not declared	Nil is the type of the value `nil`, whose main property is to be different from any other value; usually it represents the absence of a useful value.
`"boolean"`	true or false	Boolean is the type of the values `false` and `true`. In Lua, both `nil` and `false` make a condition `false`; any other value makes it `true`.
`"number"`	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers.
`"string"`	sequence of words or characters	
`"function"`	a block of code	Functions can carry out a task or compute and return values.
`"table"`	an array	New tables are created with {} braces. For example, {1, 2, 3.00e0}.

To determine the type of a variable, you can call the `type()` function, as shown in the examples below.

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
1.30000e+02      number
```

Example: Function

```
function add_two(parameter1, parameter2)
   return parameter1 + parameter2
end
print(add_two(3, 4), type(add_two))
```

```
7.00000e+00    function
```

Example: Table

```
atable = {1, 2, 3, 4}
print(atable, type(atable))
print(atable[1])
print(atable[4])
```

Defines a table with four numeric elements.
Note that the "table" value (shown here as `a096cd30`) will vary.

```
table: a096cd30      table
1.00000e+00
4.00000e+00
```

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the runtime environment.

Functions

Lua makes it simple to group commands and statements using the `function` keyword. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple parameters.

Functions can be used to form expressions that calculate and return a value; they also can 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 a global variable in the runtime environment. Like all global variables, the functions persist until they are removed from the runtime environment, are overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or Lua code in the form:

```
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function
- *parameterX*: Parameter values. You can have multiple parameters. Change the value defined by *X* for each parameter and use a comma to separate the values.
- *functionBody* is 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.

Example 1

```
function add_two(parameter1, parameter2)
   return parameter1 + parameter2
end
print(add_two(3, 4))
```

Creates a variable named `add_two` that has a variable type of function.

Output:

```
7.000000000e+00
```

Example 2

```
add_three = function(parameter1,
   parameter2, parameter3)
   return parameter1 + parameter2 +
      parameter3
end
print(add_three(3, 4, 5))
```

Creates a variable named `add_three` that has a variable type of function.

Output:

```
1.200000000e+01
```

Example 3

```
function sum_diff_ratio(parameter1,
   parameter2)
   psum = parameter1 + parameter2
   pdif = parameter1 - parameter2
   prat = parameter1 / parameter2
   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:

```
5.000000000e+00
-1.000000000e+00
6.666666667e-01
```

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. See Load a script by sending commands over the remote interface (on page 6-4).

Steps to create a function using a script

Steps	Example
1. In TSB Embedded, enter a name into the TSP Script box	`MakeMyFunction`

2. Enter the function as the body of the script	This example concatenates two strings:
	`MyFunction = function (who)`
	`  print("Hello " .. who)`
	`end`

3. Click **Save Script**	`MakeMyFunction` now exists 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.
4. Run the script as a function	`MakeMyFunction()` This instructs the instrument to run the script, which creates the `MyFunction` global variable (which is also a function).

5. 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 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	User script created in user's own program

`function myDisplay(name)`
`  display.clear()`
`  display.settext(`
`  name .. "$N is here!")`
`end`
	`loadscript`
	`function myDisplay(name)`
	`  display.clear()`
	`  display.settext(`
	`  name .. " $N is here!")`
	`end`
	`endscript`

Operators

Lua variables and constants can be compared and manipulated 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 this value 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 the `and` and the `or` logical operators use shortcut evaluation, that is, the second operand is evaluated only if necessary.

Example

```
print(10 or errorqueue.next())          1.00000e+01
print(nil or "a")                       a
print(nil and 10)                       nil
print(false and errorqueue.next())      false
print(false and nil)                    false
print(false or nil)                     nil
print(10 and 20)                        2.00000e+01
```

String concatenation

String operators

Operator	Description
`..`	Concatenates two strings. If both operands are strings or number, they are converted to strings according to Lua coercion rules, as follows: • Any arithmetic operation applied to a string tries to convert that string to a number, following the usual rules. • Whenever a number is used where a string is expected, the number is converted to a string, in a reasonable format.

Example: Concatenation

```
print(2 .. 3)                   23
print("Hello " .. "World")      Hello World
```

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority).

Operator precedence

Precedence	Operator
Highest	^ (exponentiation)
.	not, - (unary)
.	*, /
.	+, -
.	.. (concatenation)
.	<, >, <=, >=, ~=, ==
.	and
Lowest	or

You can use parentheses to change the precedences in an expression. The concatenation (". .") and exponentiation ("^") operators are right associative. All other binary operators are left associative. The examples below show equivalent expressions.

Equivalent expressions

`reading + offset < testValue/2+0.5`	=	`(reading + offset) < ((testValue/2)+0.5)`
`3+reading^2*4`	=	`3+((reading^2)*4)`
`Rdg < maxRdg and lastRdg <= expectedRdg`	=	`(Rdg < maxRdg) and (lastRdg <=expectedRdg)`
`-reading^2`	=	`-(reading^2)`
`reading^testAdjustment^2`	=	`reading^(testAdjustment^2)`

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 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
```

```
Zero is true!
```

Example: Comparison

```
x = 1
y = 2
if x and y then
   print("Both x and y are true")
end
```

```
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
```

```
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 less 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 less 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 6-23) 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

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 will exit when `list[element] = nil`.

Output:

```
Count list elements on
   numeric index:
1.000000000e+00   One
2.000000000e+00   Two
3.000000000e+00   Three
4.000000000e+00   Four
5.000000000e+00   Five
6.000000000e+00   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`

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:
1.000000000e+00   One
2.000000000e+00   Two
3.000000000e+00   Three
4.000000000e+00   Four
5.000000000e+00   Five
6.000000000e+00   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.

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:
1.000000000e+00   One
2.000000000e+00   Two
3.000000000e+00   Three
Counting from one to four, in steps of two:
1.000000000e+00   One
3.000000000e+00   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     1.000000000e+00    Sunday
Monday     2.000000000e+00    Monday
Tuesday    3.000000000e+00    Tuesday
Wednesday  4.000000000e+00    Wednesday
Thursday   5.000000000e+00    Thursday
Friday     6.000000000e+00    Friday
Saturday   7.000000000e+00    Saturday
```

Break

The `break` statement can be used to terminate the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A break ends the innermost enclosing loop.

Return and break statements can only be written as the last statement of a block. If it is really necessary to return or break in the middle of a block, then an explicit inner block can be used.

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 i == 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 i =
    3.00000e+00
```

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:

```
0.00000e+00   1.00000e+00
1.00000e+00   1.00000e+00
1.00000e+00   2.00000e+00
2.00000e+00   3.00000e+00
3.00000e+00   5.00000e+00
5.00000e+00   8.00000e+00
8.00000e+00   1.30000e+01
1.30000e+01   2.10000e+01
2.10000e+01   3.40000e+01
3.40000e+01   5.50000e+01
5.50000e+01   8.90000e+01
8.90000e+01   1.44000e+02
1.44000e+02   2.33000e+02
2.33000e+02   3.77000e+02
3.77000e+02   6.10000e+02
```

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"]`.

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:
	`1.00000e+00`
	`2.00000e+00`
	`3.00000e+00`
	`4.00000e+00`

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 http://www.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

This library provides generic functions for string manipulation.

Base library functions

Function	Description
`collectgarbage()` `collectgarbage(`*`limit`*`)`	Sets the garbage-collection threshold to the given limit (in kilobytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, then 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 6-32) for more information.
`gcinfo()`	Returns the number of kilobytes of dynamic memory that the Test Script Processor (TSP®) scripting engine is using, and returns the current garbage collector threshold (also in kilobytes). See Lua memory management (on page 6-32) for more information.
`print(`*`e1, e2, ...`*`)`	Receives any number of arguments, and generates a response message, using the `tostring()` function to convert them to strings (note that numbers are converted to scientific notation using `format.asciiprecision`). The output is not formatted. For formatted output, you can use the `string.format()` command (see String library functions (on page 6-33)). Also see print() (on page 7-144).
`tonumber(`*`x`*`)` `tonumber(`*`x, base`*`)`	Returns *x* converted to a number. If *x* is already a number, or a convertible string, then the number is returned; otherwise, it returns `nil`. An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter `A` (in either upper or lower case) represents 10, `B` represents 11, and so forth, with `Z` representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(`*`x`*`)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(`*`v`*`)`	The possible results of this function are `"nil"` (a string, not the value nil), `"number"`, `"string"`, `"boolean"`, `"table"`, `"function"`, `"thread"`, and `"userdata"`.

Lua does automatic memory management, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua manages memory automatically by occasionally running a garbage collector to collect all objects that are no longer accessible from Lua. All objects in Lua are subject to automatic management, including tables, variables, functions, threads, and strings.

Lua uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory Lua is using; the other is a threshold. When the number of bytes crosses the threshold, Lua runs the garbage collector, which reclaims the memory of all inaccessible objects. The byte counter is adjusted and the threshold is reset to twice the new value of the byte counter.

String library functions

This library provides generic functions for string manipulation, such as finding and extracting substrings. When indexing a string in Lua, the first character is at position 1 (not 0, as in ANSI C). Indices may be negative and are interpreted as indexing backward from the end of the string. Thus, the last character is at position -1, and so on.

String library functions

Function	Description

`string.byte(s)`
`string.byte(s, i)`
`string.byte(s, i, j)`
	Returns the internal numeric codes of the characters `s[i], s[i+1], ···, s[j]`. The default value for `i` is `1`; the default value for `j` is `i`. Note that numeric codes are not necessarily portable across platforms.

`string.char(···)`	Receives zero or more integers. 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. Note that numeric codes are not necessarily portable across platforms.

`string.format(`
`formatstring,`
`...)`
	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the modifiers `*`, `l`, `L`, `n`, `p`, and `h` are not supported and there is an extra option, `q`. The `q` option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written.
	For example, the call:
	`string.format('%q', 'a string with "quotes" and \n new line')`
	will produce the string:
	`"a string with \"quotes\" and \ new line"`
	The options `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `X`, and x all expect a number as argument. `q` and `s` expect a string. This function does not accept string values containing embedded zeros, except as arguments to the `q` option.

`string.len(s)`	Receives a string and returns its length. The empty string `""` has length `0`. Embedded zeros are counted, so `"a\000bc\000"` has length `5`.
`string.lower(s)`	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged. The definition of what an uppercase letter is depends on the current locale.
`string.rep(s, n)`	Returns a string that is the concatenation of `n` copies of the string `s`.
`string.sub(s, i)` `string.sub(s, i, j)`	Returns the substring of `s` that starts at `i` and continues until `j`; `i` and `j` can be negative. If `j` is absent, it is assumed to be equal to `-1` (which is the same as the string length). In particular, the call `string.sub(s, 1, j)` returns a prefix of `s` with length `j`, and `string.sub(s, -i)` returns a suffix of `s` with length `i`.

String library functions

Function	Description
`string.upper(s)`	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged. The definition of what a lowercase letter is depends on the current locale.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions

Function	Description
`math.abs(x)`	Returns the absolute value of *x*.
`math.acos(x)`	Returns the arc cosine of *x*.
`math.asin(x)`	Returns the arc sine of *x*.
`math.atan(x)`	Returns the arc tangent of *x*.
`math.atan2(y, x)`	Returns the arc tangent of *y*/*x*, but uses the signs of both parameters to find the quadrant of the result. (It also handles correctly the case of *x* being zero.)
`math.ceil(x)`	Returns the smallest integer larger than or equal to *x*.
`math.cos(x)`	Returns the cosine of *x*.
`math.deg(x)`	Returns the angle *x* (given in radians) in degrees.
`math.exp(x)`	Returns the value e^x.
`math.floor(x)`	Returns the largest integer smaller than or equal to *x*.
`math.frexp(x)`	Returns `m` and `e` such that *x* = $m2^e$, where `e` is an integer and the absolute value of `m` is in the range `[0.5, 1]` (or zero when *x* is zero).
`math.ldexp(x, n)`	Returns $m2^e$ (`e` should be an integer).
`math.log(x)`	Returns the natural logarithm of *x*.
`math.log10(x)`	Returns the base-10 logarithm of *x*.
`math.max(x, ...)`	Returns the maximum value among its arguments.
`math.min(x, ...)`	Returns the minimum value among its arguments.
`math.pi`	The value of π (3.141592654).
`math.pow(x, y)`	Returns x^y (you can also use the expression `x^y` to compute this value).
`math.rad(x)`	Returns the angle *x* (given in degrees) in radians.

`math.random()`
`math.random(m)`
`math.random(m, n)`
	This function is an interface to the simple pseudorandom generator function `rand` provided by ANSI C.
	When called without arguments, returns a uniform pseudorandom real number in the range `[0,1]`. When called with an integer number *m*, `math.random()` returns a uniform pseudorandom integer in the range `[1, m]`. When called with two integer numbers *m* and *n*, `math.random()` returns a uniform pseudorandom integer in the range `[m, n]`.

`math.randomseed(x)`	Sets *x* as the "seed" for the pseudorandom generator: equal seeds produce equal sequences of numbers.
`math.sin(x)`	Returns the trigonometric sine function of *x*.
`math.sqrt(x)`	Returns the square root of *x*. (You can also use the expression `x^0.5` to compute this value.)
`math.tan(x)`	Returns the tangent of *x*.

Programming example

Script with a for loop

The following script puts a message on the 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

`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`
	`loadscript`
	
	`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`
	`endscript`

Using Test Script Builder (TSB)

Installing the TSB software

To install the TSB software:

1. Close all programs.
2. Place the Test Script Builder Software Suite CD (Keithley Instruments part number KTS-850B01 or greater) into your CD-ROM drive.
3. Follow the on-screen instructions.

If your web browser does not start automatically and display a screen with software installation links, open the installation file (setup.exe) found on the CD to initiate installation.

Project Navigator

The Project Navigator is located on the left side of the workspace. The navigator consists of project folders and the script files (.tsp) created for each project. Each project folder can have one or more script files.

Script Editor

The script is written and modified in the Script Editor. Notice that there is a tab available for each opened script file. A script project is then downloaded to the instrument, where it can be run.

Programming interaction

Up to seven tabs can be displayed in the lower pane of the workspace (the script editor) to provide programming interaction between the Test Script Builder and the instrument. The instrument console shown in Using Test Script Builder (TSB) is used to send commands to the connected instrument. Retrieved data (for example, readings) from commands and scripts appear in the instrument console.

Working with TSB Embedded

TSB Embedded is an alternative to the full version of Test Script Builder (TSB) Suite. The capabilities of TSB Embedded are very similar to TSB. TSB Embedded includes a command line interface that you can use to issue instrument commands, create, modify, and save test scripts to the instrument. For additional information, refer to the Model 2651A User's Manual.

Using instrument commands

NOTE

The response from the instrument appears in the instrument output window.

To send commands from the command line:

1. Type the command in the console and then press the **Enter** key.
2. (Optional) Click **Clear** to clear the instrument output window.

To create a new script:

1. Click in the script editor window and then type the first line of your script. Then press the **Enter** key to advance to line 2.
2. In the TSP Script line, type the name of the script and then click **Save Script**.
   The instrument validates the syntax and then saves the script to the nonvolatile memory.

To remove the code from the script editor:

Click **Clear**.

To run a script:

1. Select the desired script from the User script window.
2. Click **Run**.

To stop a running script:

Click **Abort Script**.

To delete a script from TSB embedded:

NOTE

You cannot retrieve a deleted script.

1. Select the desired script from the user script window.
2. Click **Delete**.

To modify a script:

1. Select the desired script from the **User Scripts** window, and then modify the desired code in the script editor.
2. Click **Save Script** to validate the syntax and save the script.
3. The following message is displayed: **Script <filename> will be overwritten**. Do one of the following:
   - To overwrite the script, click **OK**.
   - To save the script with a new name, click **Cancel** and then type the name of the script in the name field.

To export a script to be saved on an external drive (or to store as a back-up on your computer):

1. To export a script, click on the name of the script in the user script window and then click **Export to PC**. The Save dialogue box displays.
2. Use the drop-down arrow to change folders, and navigate to the desired file or directory.
3. In the File Name field, type the name of the file and then click **Save**.

Advanced scripting for TSP

Global variables and the script.user.scripts table

When working with script commands, it can be 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.

There are several types of scripts in the runtime environment:

- Named scripts
- Unnamed scripts
- The anonymous script

When a named script is loaded into the runtime environment:

- A global variable with the same name is created to reference the script more conveniently.
- An entry for the script is added to the `script.user.scripts` table.

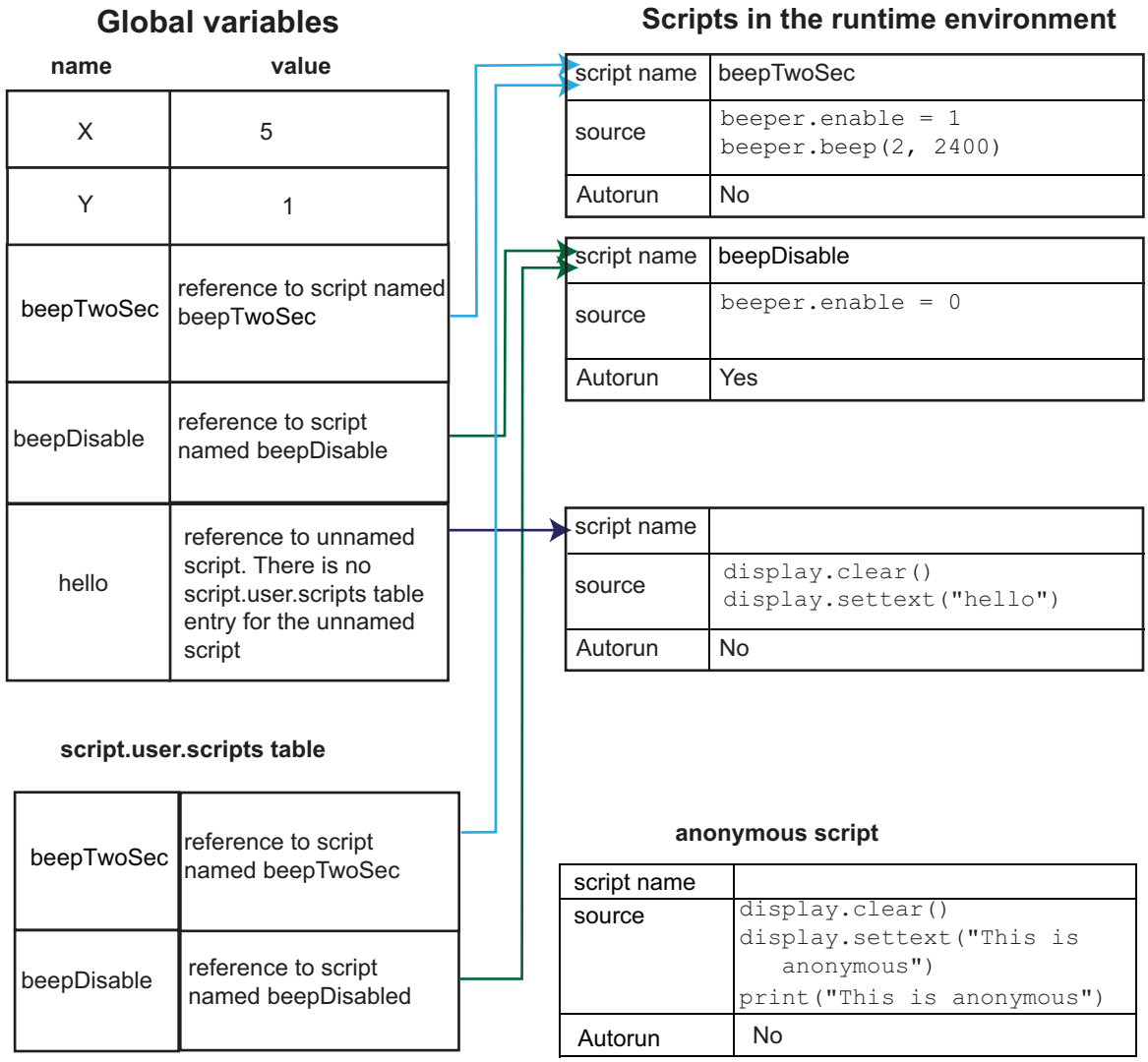
When an unnamed script is loaded using the script.new() (on page 7-155) function, a global variable is added. Nothing is added to the `script.user.scripts` table.

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 84: 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:

scriptVar = Name of the variable created when the script is loaded into the runtime environment
code = Content of the script
name = Name that is added to the `script.user.scripts` table

For example, to set up a two-second beep, you could send the command:

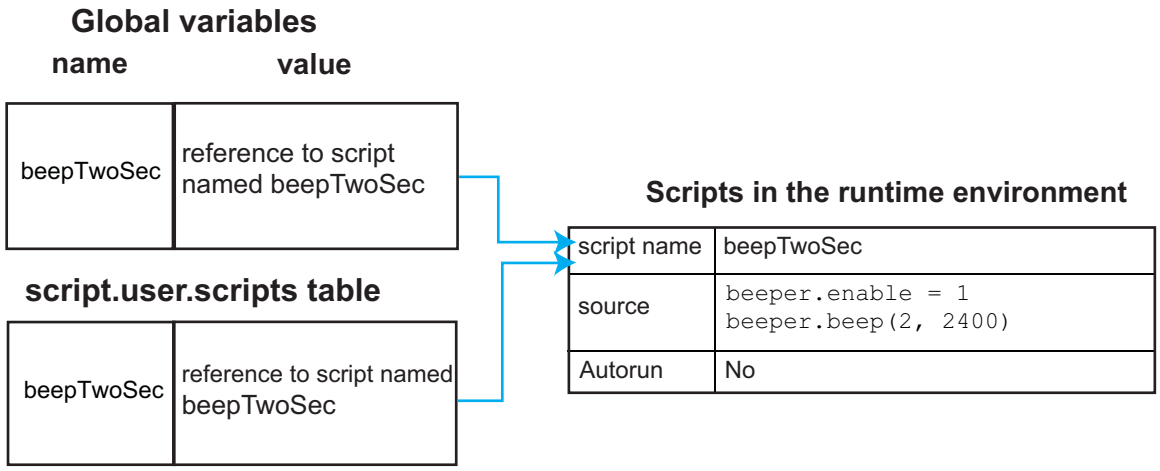
```
beepTwoSec = script.new("beeper.enable = 1 beeper.beep(2, 2400)", "beepTwoSec")
```

To run the new script, send the command:

```
beepTwoSec()
```

When you add `beepTwoSec`, the global variable and `script.user.script` table entries are made to the runtime environment as shown in the following figure.

Figure 85: Runtime environment after creating a script



Create an unnamed script using script.new()

NOTE

Unnamed scripts are not available from the front panel display of the instrument. Only the anonymous script and named scripts are available from the 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, if you sent the command:

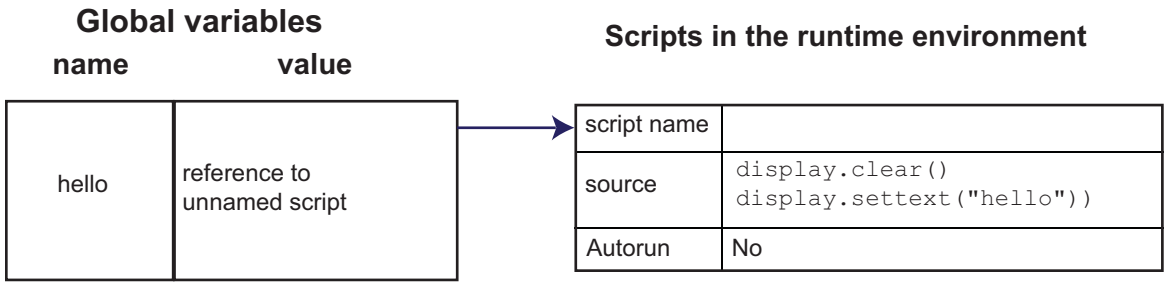
```
hello = script.new('display.clear() display.settext("hello")')
```

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 86: Create an unnamed script



Unnamed scripts are also created if you create a new script with the name attribute of a script that is already in the `script.user.scripts` table. In this case, 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 global variable `beepTwoSec` remains in the runtime environment unchanged (it points to the now unnamed script).
- 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).
- A new script named `beepTwoSec` is added to the runtime environment.

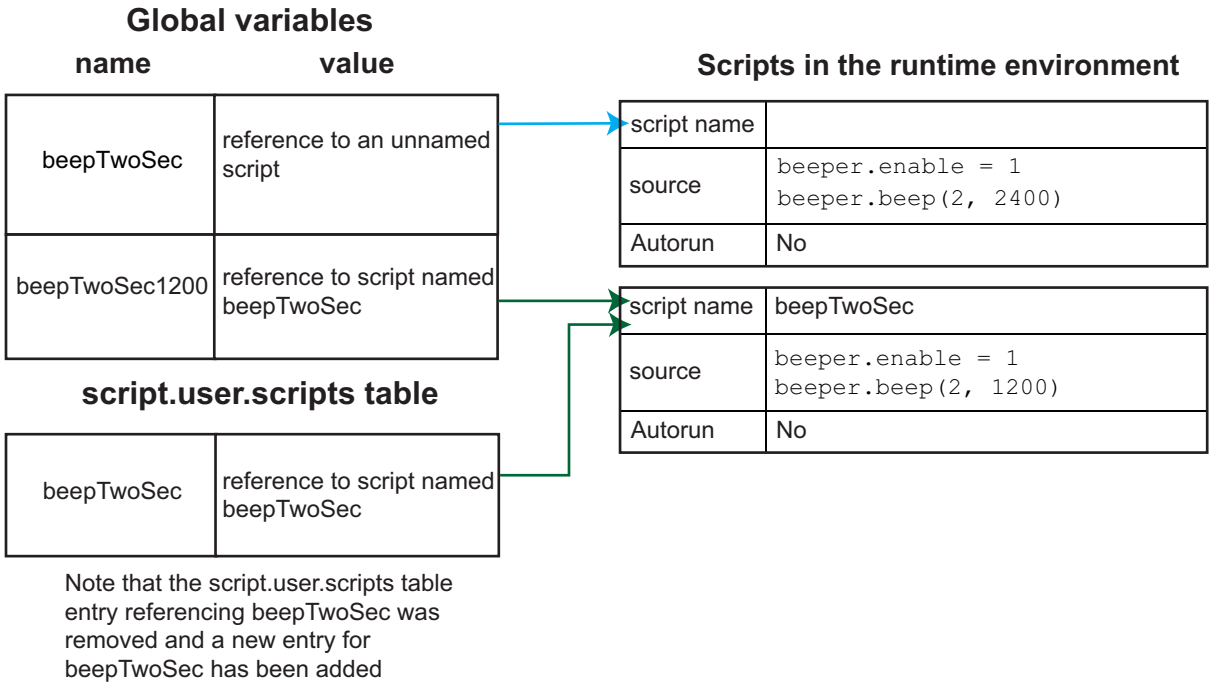
In this example, you can access the new script by sending either of the following commands:

```
beepTwoSec1200()
script.user.scripts.beepTwoSec()
```

To access the unnamed script, you can send the command:

```
beepTwoSec()
```

Figure 87: Change a named script with an unnamed script



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 back into the runtime environment:

```
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")
```

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 autoexecute script.

To change the name of a script, use the command:

```
scriptVar.name = "renamedScript"
```

Where:

scriptVar	=	The global variable name
"renamedScript"	=	The new name of the user script that was referenced by the *scriptVar* 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()
```

`beep2sec` can be run using the command:

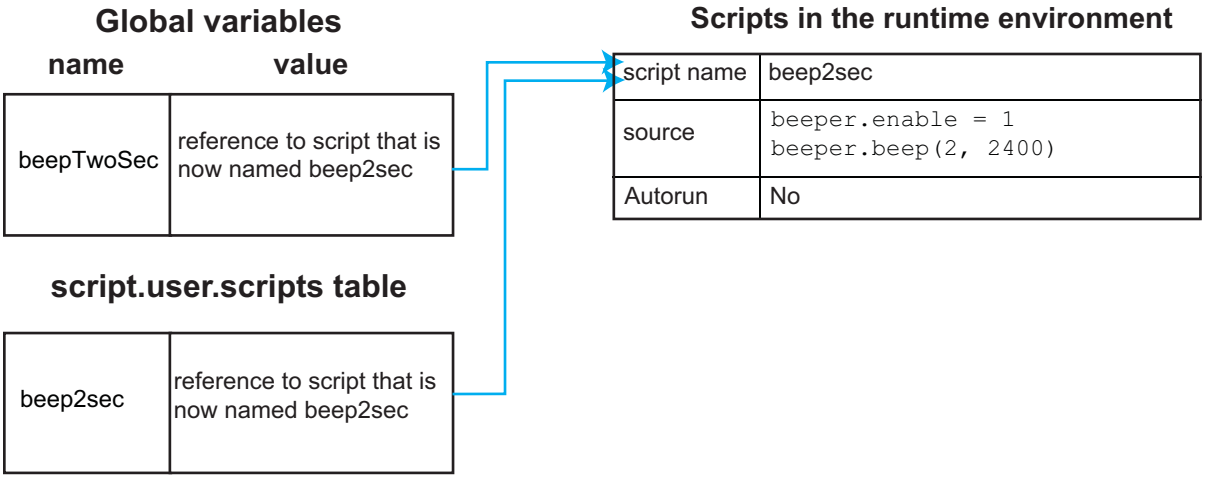
```
script.user.scripts.beep2sec()
```

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 can be found in the `script.user.scripts` table under its new name.

Figure 88: 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 7-155).

Delete user scripts from the instrument

In most cases, you can delete the script using `script.delete()` as described in Delete user scripts (on page 6-11) and then turning the instrument off and then 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 global 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`, you would send:

```
script.delete("beepTwoSec")
```

You can also use TSB Embedded to delete a script from nonvolatile memory. In the TSB Embedded page, select the script from the User Scripts list and click **Delete**.

Memory considerations for the runtime environment

The runtime environment has a fixed amount of memory for storing user scripts and other run-time information.

The Model 2651A 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

The runtime environment, user-created reading buffers, and active sweep configuration must fit within the 24 MB of memory available. The amount of memory used by a reading buffer is approximately 15 bytes for each entry requested. A reading buffer uses a slight amount of these resources, but it 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 2,500. The memory reserved for the reading buffers is calculated as follows:

(1000 * 15) + (2500 * 15) = 52,500 bytes or 52.5 kilobytes.

Note that the dedicated reading buffers do not consume memory 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. You should be 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 these purposes to exceed 24MB. When this occurs, there is a risk that memory allocation errors will be generated and commands will not be executed as expected. If memory allocation errors are encountered, the state of the instrument cannot be guaranteed. After attempting to save any important data, it is recommended that you turn off power to the instrument and turn it back on to return the instrument to a known state. Turning the power on and off resets the runtime environment.

Unsaved scripts and 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%, 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 6-45).
- 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 6-32) 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.

⚠ 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, it is recommended that you turn off power to the instrument and turn it back on to return the instrument to a known state. Cycling power resets the runtime environment. Unsaved scripts will be lost.

TSP-Link system expansion interface

Overview

The TSP-Link® expansion interface allows the Model 2651A 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 Model 2651A instruments to achieve greater currents in both source voltage and source current applications requires specific precautions including configuration settings. Make sure you adequately understand the risks involved and the measures needed to accommodate the combination of two Model 2651A instruments. To prevent damage to the Model 2651A, connected instruments, as well as the device under test, make sure proper procedures are not used. For further information, visit the Keithley Instruments website (http://www.keithley.com) for application notes on combining two Model 2651A instruments' channels.

Although the expanded system can control up to 32 TSP-Link enabled instruments, combining the output of a Model 2651A to increase either voltage or current is only permitted with another Model 2651A. Combining two Model 2651A instruments to achieve greater currents in both source voltage and source current applications requires specific precautions including configuration settings. For further information, visit the Keithley Instruments website (http://www.keithley.com) for application notes on combining two Model 2651A instruments' channels.

Master and subordinates

In a TSP-Link® system, one of the nodes (instruments) is the master node and the other nodes are the subordinate modes.

The master node can control the other nodes (subordinates) in the system. When any node transitions from local operation to remote, 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. For more information about remote and local operations, see Factory scripts (on page 5-20).

TSP-Link system

You can use the TSP-Link® expansion interface to expand your test system to include up to 64 addressable TSP-Link enabled instruments (32 instruments at a time). The expanded system can be stand-alone or computer-based.

Stand-alone system: You can run a script from the front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation (REM indicators turn on). The node running the script becomes the master and can control all of the 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 LAN, GPIB, or RS-232 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 of the 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 or enclosure (node) attached to the TSP-Link bus must be identified. Identify each node by assigning a unique TSP-Link node number.

In test script programs, nodes look like tables. There is one global table named `node` that contains all the actual nodes that are themselves tables. An individual node is accessed as `node[`*N*`]` where *N* is the node number assigned to the node. Each node has certain attributes that can be accessed as elements of its associated table. These are listed as follows:

- `model`: The product model number string of the node.
- `revision`: The product revision string of the node.
- `serialno`: The product serial number string of the node.

There is also an entry for each logical instrument on the node (see Logical instruments (on page 7-3)).

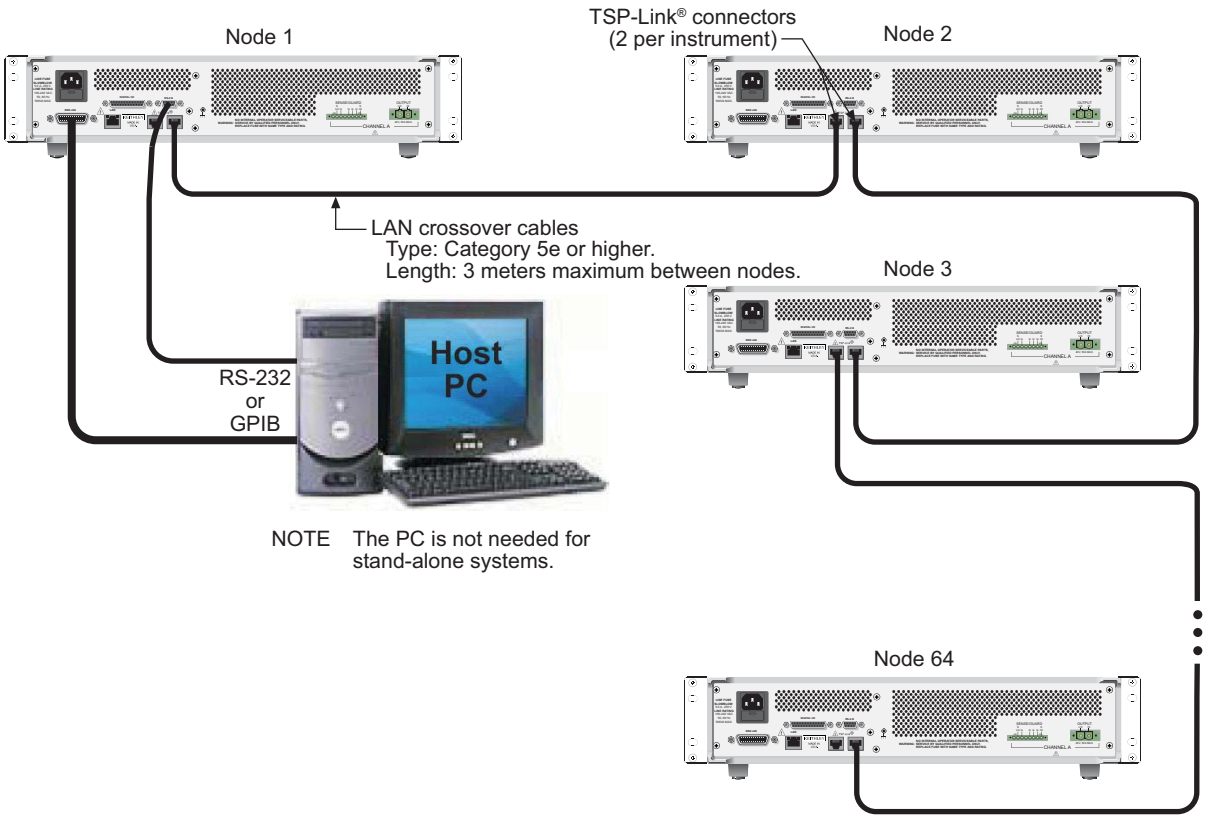
It is not necessary to know the node number of the node running a script. The variable `localnode` is an alias for the node entry the script is running on. For example, if a script is running on node 5, the global variable `localnode` will be an alias for `node[5]`.

Connections

Connections for an expanded system are shown in the following figure. As shown, one unit is optionally connected to the computer using the GPIB, LAN, or RS-232 interface. Details about these computer communication connections are described in Communications interfaces (on page 2-73).

As shown, all the units in the system are connected in a sequence (daisy-chained) using LAN crossover cables.

Figure 89: Series 2650 TSP-Link connections



Initialization

Before a TSP-Link system can be used, 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 Model 2651A instrument is assigned as node 1. The node number for each unit is stored in its nonvolatile memory and will not be lost when the instrument is turned off. You can assign a node number (1 to 64) to a Model 2651A using the front panel or through programming.

To assign a node number from the front panel of the instrument:

1. Press the **MENU** key, then select **TSPLINK > NODE**.
2. Press the navigation wheel ⟳ and select the desired number.
3. Press the **ENTER** key to select the node number.

To assign a node number through programming:

Set the `tsplink.node` attribute of that instrument:

```
tsplink.node = N
```

Where: *N* = 1 to 64

The node number of an instrument can be determined by reading the `tsplink.node` attribute as follows:

```
print(tsplink.node)
```

The above `print` command will output the node number. For example, if the node number is 1, the value `1.000000e+00` will be 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. For initialization to succeed, all units must be turned on when the TSP-Link network reset is performed.

NOTE

If you change the system topology after initialization, you must reinitialize the system by performing a TSP-Link network reset. Changes that affect the system topology include powering down or rebooting any unit in the system, or rearranging or disconnecting the LAN cable connections between units.

Front panel operation

To reset the TSP-Link® network from the front panel:

1. Press the **MENU** key, select **TSPLINK**, and then press the **ENTER** key.
2. 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
`tsplink.reset()`	Initializes the TSP-Link network
`tsplink.state`	"`online`" if the most recent TSP-Link reset was successful; "`offline`" if the reset operation failed

An attempted TSP-Link reset will fail 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 unit performing the reset (only if the expected number of nodes was not provided in the reset call)
- One or more of the units in the system is not powered on
- 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 is successful, `online` will be output to indicate that communications with all nodes have been established.

Using the expanded system

Accessing nodes

A TSP-Link® `reset()` command populates the node table. Each unit in the system corresponds to an entry in this table. Each entry is indexed by the node number of the unit. 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.

Each of these entries is a table holding all of the logical instruments (and associated commands) shared by the corresponding unit (see Logical instruments for more details). Source-measure unit (SMU) A on node 1, therefore, could be accessed as `node[1].smua`.

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, `localnode` can be used instead of `node[1]`.

The following programming examples illustrate how to access instruments in the TSP-Link system (shown in TSP-Link connections):

- Any of the following three commands can be used to 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 will reset the SMU A of node 4, which is a subordinate. The other nodes are not affected.

```
node[4].smua.reset()
```

System behavior

Using the reset () command

While most TSP-Link® system operations target a single node in the system, the `reset()` command affects the system as a whole by resetting all nodes to their default settings:

```
-- Resets all nodes in a TSP-Link system.
reset()
```

`node[N]` and `localnode` can be used with `reset()` to reset only one of the nodes. The other nodes are not affected. The following programming example illustrates this type of reset operation.

```
-- Resets node 1 only.
node[1].reset()
-- Resets node 1 only.
localnode.reset()
-- Resets node 4 only.
node[4].reset()
```

Abort

An `abort` command will terminate an executing script and return all nodes to local operation (REM indicators turn off), dissolving the master/subordinate relationships between nodes. An abort operation is invoked by either issuing an `abort` command to the master node or pressing the EXIT (LOCAL) key on any node in the system.

An abort operation can also be performed 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 synchronization lines that function similarly to the digital I/O synchronization lines. See Digital I/O (on page 3-83) and Triggering (on page 3-32) for more information.

TSP advanced features

Use the Test Script Processor (TSP®) scripting engine's advanced features to run test scripts simultaneously, to manage resources allocated to test scripts running in simultaneously, and to use the data queue to facilitate real-time communication between nodes on the TSP-Link® network.

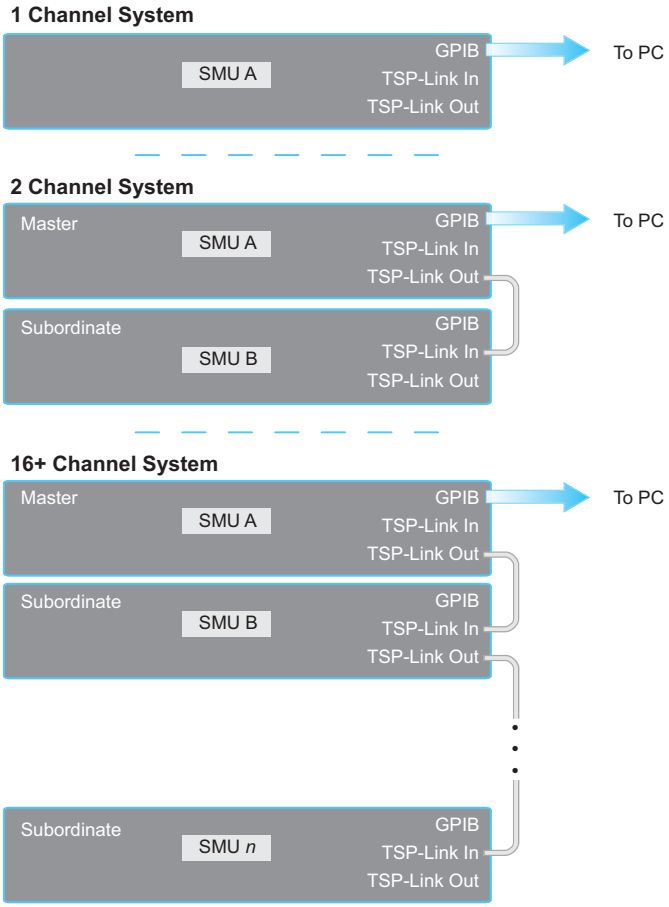
Running test scripts simultaneously improves functional testing, provides higher throughput, and expands system flexibility.

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 GPIB connection to the computer.

Figure 90: Model 2651A multiple TSP-Link networks

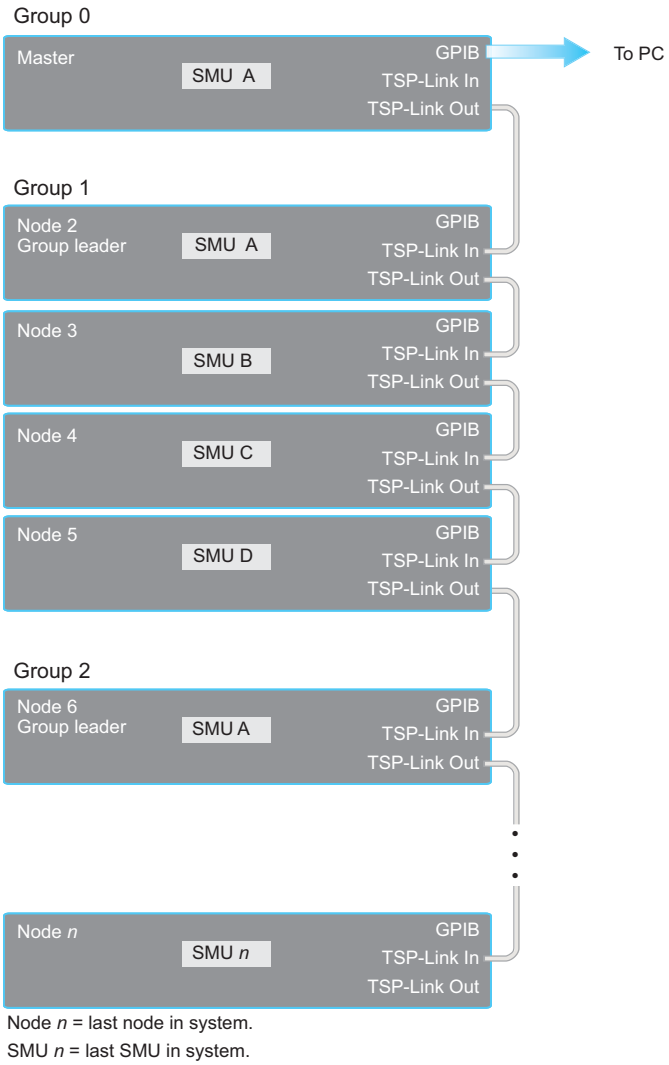


SMU *n* = last SMU in system.

The second method to run parallel test scripts is to use groups with a single TSP-Link network. A group consists of one or more nodes with the same group number. Each group on the TSP-Link network can run different test scripts at the same time (in parallel).

The following figure displays a single TSP-Link network with groups. This method requires one TSP-Link network and a single GPIB connection to the computer.

Figure 91: Model 2651A single TSP-Link network with groups



Node *n* = last node in system.
SMU *n* = last SMU in system.

The following table describes the functions of a single TSP-Link network. Each group in this example runs multiple test scripts at the same time (parallel processing).

TSP-Link network group functions

Group number	Group members	Present function
0	Master node	Initiates and runs a test script on node 2 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 through node 5
	Node 3 through node 5	Performs remote operations initiated by node 2
2	Node 6	Runs the test script initiated by the master node Initiates remote operations through node *n*
	Node *n*	Performs remote operations initiated by node 6

Using groups to manage nodes on TSP-Link network

The primary purpose of a group is to assign each node to run different test scripts at the same time (in parallel). Each node must belong to a group; a group can consist of one or more members. Group numbers are not assigned automatically; you must use remote commands to assign each node to a group.

Master node overview

The master node is always the node that coordinates activity on the TSP-Link® network. All nodes assigned to group 0 belong to the same group as the master node.

The following list describes the functionality of the master node:

- The only node that can send 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
- Sends the `waitcomplete()` command to wait for the local group that the master node belongs to, to wait for a remote group, or to wait for all nodes on the TSP-Link network to complete overlapped operations

Group leader overview

Each group has a dynamic group leader. The last node in a group running any operation initiated by the master node is the group leader.

The following list describes the functionality of the group leader:

- Runs 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 0 (zero) to 64. The default group number is 0. You can change the group number at any time.

Use the following code to dynamically assign nodes to a group.

Note the following:

- Each time the node powers off, the group number for that node changes to 0
- Replace N with the node number
- N represents the node number that runs the test scripts and the Lua code
- Each time the node powers off, the group number for that node changes to 0
- Replace G with the group number

```
-- Assigns the node to a group.
node[N].tsplink.group = G
```

Reassigning groups

Use the following code to change group assignment. You can add or remove a node to a group at anytime.

```
-- Assigns the node to a different group.
node[N].tsplink.group = G
```

Running parallel test scripts

You can send the `execute()` command from the master node to initiate 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 in parallel. Use the following code to send the `execute()` command on a remote node.

Note the following:

- The *N* parameter represents the node number that runs the test script (replace *N* with the node number).

To set the global variable on node N equal to 2.5:

```
node[N].execute("setpoint = 2.5")
```

The following code demonstrates how to run a test script defined on a remote node.

NOTE

For this example, *scriptVar* is defined on the local node.

To run scriptVar on node N:

```
node[N].execute(scriptVar.source)
```

The programming example below demonstrates how to run a test script defined on a remote node.

NOTE

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 6-59) for more information.

Coordinating overlapped operations in remote groups

Errors occur if you send a command to a node in a remote group running an overlapped operation. All nodes in a group must be in the overlapped idle state before the master node can send a command to the group.

Use the `waitcomplete()` command to:

- **Group leader and master node**: To wait for all overlapped operations running in the local group to complete
- **Master node only**: To wait for all overlapped operations running on a remote group to complete on the TSP-Link® network
- **Master node only**: To wait for all groups to complete overlapped operations

For additional information, see waitcomplete() (on page 7-374).

The following code is an example on how to send the `waitcomplete()` command from the master node:

```
-- Waits for each node in group N to complete all overlapped operations.
waitcomplete(N)
-- Waits for all groups on the TSP-Link network to complete overlapped operations.
waitcomplete(0)
```

The 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 send the `waitcomplete()` command:

```
-- Waits for all nodes in a local group to complete all overlapped operations.
waitcomplete()
```

Using the data queue for real-time communication

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. You can use the data queue to retrieve data from any node in a group performing an overlapped operation. In addition, the master node and the group leaders can use the data queue as a way to coordinate activities.

The data queue uses the first-in, first-out (FIFO) structure to store data. Nodes running test scripts in parallel can store data in the data queue for real-time communication. Each Model 2651A has an internal data queue. You can access the data queue from any node at any time.

You can use the data queue to post numeric values, strings, and tables. Tables in the data queue consume one entry. A new copy of the table is created when the table is retrieved from the data queue. The copy of the table does not contain any references to the original table or any subtables.

To add or retrieve values from the data queue and view the capacity, see the Command reference (see "Commands" on page 7-8).

Copying test scripts across the TSP-Link network

To run a large script on a remote node, it is recommended that you copy the test script to the remote node to increase the speed of test script initiation.

Use the code below to copy test scripts across the TSP-Link® network. This example creates a copy of a script on the remote node with the same name.

Note the following:

- Replace *N* with the number of the node that receives a copy of the script
- Replace *scriptName* with the name of the script that you want to copy from the local node

```
-- Adds the source code from scriptName to the data queue.
node[N].dataqueue.add(scriptName.source)
-- Creates a new script on the remote node
-- using the source code from scriptName.
node[N].execute(scriptName.name ..
   "= script.new(dataqueue.next(), [[" .. scriptName.name .. "]])")
```

Removing stale values from the reading buffer

The node that acquires the data 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.

Running Lua code remotely can cause reading buffer data 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.

The following code demonstrates how stale values occur and how to use the `clearcache()` command to clear the cache.

Note the following:

- Replace *N* with the node number
- Replace *G* with the group number

```
-- Creates a reading buffer on a node in a remote group.
node[N].tsplink.group = G
node[N].execute("rbremote = smua.makebuffer(20)" ..
                "smua.measure.count = 20  " ..
                "smua.measure.v(rbremote)")
-- Creates a variable on the local node to
-- access the reading buffer.
rblocal = node[N].getglobal("rbremote")
-- Access data from the reading buffer.
print(rblocal[1])
-- Runs code on the remote node that updates the reading buffer.
node[N].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 will never update. Every time the print command is
-- issued after the first print command, the same data buffer
-- values will print.
print(rblocal[1])
```

TSP-Net

Overview

The TSP-Net® library allows the Model 2651A to control LAN-enabled devices directly through its LAN port. This enables the Model 2651A to communicate directly with a non-TSP® enabled device without the use of a controlling computer.

TSP-Net capabilities

For both Test Script Processor (TSP®) and non-TSP devices, the TSP-Net library permits the Model 2651A to control a remote device through the LAN port. Using TSP-Net library methods, you can transfer string data to and from a remote device, 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 and is not available from the front panel.

You can use TSP-Net commands to communicate with any Ethernet-enabled device. However, specific TSP-Net commands exist for TSP-enabled devices 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 Model 2651A can download a script to another TSP-enabled device and have both devices run scripts independently. The Model 2651A can read the data from the remote device and either manipulate the data or send the data to a different remote device 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 Model 2651A.

Using TSP-Net with any Ethernet-enabled device

NOTE

Refer to Remote Commands (on page 5-1) for more details about the commands presented in this section.

To communicate to a remote Ethernet-enabled device from the Model 2651A:

1. Connect to the remote device through the LAN port;
   - The Model 2651A has Auto-MDIX, so you can use either a LAN crossover cable or a LAN straight-through cable to connect directly from the Model 2651A to an Ethernet-enabled device, or to a hub.
2. Establish a new connection to a remote device at a specific IP address using the `tspnet.connect()` function. For non-TSP® enabled devices, you must also provide the port number, or the Model 2651A assumes the remote device to be TSP-capable and enables TSP prompts and error handling.

   If the Model 2651A is not able to make a connection to the remote device, it generates a timeout error. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 seconds.

NOTE

Set `tspnet.tsp.abortonconnect` to `1` to abort any script currently running on a remote TSP-enabled device.

3. Use `tspnet.write()` or `tspnet.execute()` to send strings to a remote device. Using `tspnet.write()` sends strings to the device exactly as indicated, and you must supply any needed termination characters or other lines. Use `tspnet.termination()` to specify the termination character. If you use `tspnet.execute()` instead, the Model 2651A appends termination characters to all strings sent to the command.

4. Retrieve responses from the remote device using `tspnet.read`. The Model 2651A suspends operation until data is available or a timeout error is generated. You can check if data is available from the remote device using `tspnet.readavailable()`.
5. Disconnect from the remote device using `tspnet.disconnect()`. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote non-TSP® enabled device, and send and receive data from this device:

```
-- Disconnect all existing TSP-Net connections.
tspnet.reset()
-- Set tspnet timeout to 5 seconds.
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 termination character to CRLF. You must do this on a per
-- connection basis after connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device.
tspnet.write(id_instr, "*idn?" .. "\r\n")
-- Read the data available, then print it.
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

TSP-Net versus TSP-Link to communicate with TSP-enabled devices

The TSP-Link® network interface is the preferred communication method when communicating between the Model 2651A and another TSP-enabled instrument. The advantages of using the TSP-Link network interface over using TSP-Net® commands include:

- **Error checking**: When connected to a TSP-enabled device, all errors that occur on the remote device are transferred to the error queue of the Model 2651A. The Model 2651A indicates errors from the remote device by prefacing these errors with “Remote Error”.

  For example, if the remote device generates error number 4909, the Model 2651A generates the error string “Remote Error: (4909) Reading buffer not found within device.”

- **Digital I/O triggering**: TSP-Link connections have three synchronization lines that are available to each device on the TSP-Link network. You can use any one of the synchronization lines to perform hardware triggering between devices on the TSP-Link network. Refer to Hardware trigger modes for more details.

These advantages make using TSP-Link connections to control another TSP-enabled device the best choice for most applications. However, if the distance between the Model 2651A and the TSP-enabled device is longer than 15 feet, use TSP-Net commands.

To establish a remote TSP-Net connection with a TSP-enabled device, use `tspnet.connect()` without specifying a port number. The Model 2651A enables TSP prompt and error handling for the remote device, which allows you to successfully use the commands listed in Instrument commands: TSP-enabled device control (on page 6-63) to load and run scripts and retrieve reading buffers.

Abort any operation on the remote TSP-enabled device using the `tspnet.tsp.abort()` command.

Instrument commands: General device control

The following instrument commands provide general device control:

tspnet.clear() (on page 7-361)
tspnet.connect() (on page 7-361)
tspnet.disconnect() (on page 7-363)
tspnet.execute() (on page 7-363)
tspnet.idn() (on page 7-364)
tspnet.idn() (on page 7-364)
tspnet.read() (on page 7-365)
tspnet.readavailable() (on page 7-366)
tspnet.reset() (on page 7-366)
tspnet.termination() (on page 7-367)
tspnet.timeout (on page 7-368)
tspnet.write() (on page 7-371)

Instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

tspnet.tsp.abort() (on page 7-368)
tspnet.tsp.abortonconnect (on page 7-369)
tspnet.tsp.rbtablecopy() (on page 7-369)
tspnet.tsp.runscript() (on page 7-370)

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 via 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 a data print it to the file
   -- 11.2 is an example of data collected
   cmd = "echo " .. string.format("%g", 11.2) .. " >> datafile.dat\r\n"
   tspnet.write(id, cmd)
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    tspnet.disconnect(id)
end
test_tspnet()
```

Section 7

Command reference

In this section:

Command programming notes 7-1
Using the command reference 7-4
Commands 7-8

Command programming notes

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:

```
gpib.address = address
```

Where:

address is an integer (0 to 30) that you specify. For example, to set this attribute to 15 you would send:

```
gpib.address = 15
```

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 would send:

```
digio.trigger[7].assert()
```

To assert trigger line with a variable as the integer, you would send:

```
triggerline = 7

digio.trigger[triggerline].assert()
```

Example 3:

smu*X*.trigger.measure.*Y*(*rbuffer*)

Where:

X is refers to the 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 will be stored.

For example, to use SMU A to take voltage measurements and store them in buffer vbuffername, you would send:

```
smua.trigger.measure.v(vbuffername)
```

Syntax rules

The following table lists syntax requirements to build well-formed instrument control commands.

Syntax rules for instrument commands

Syntax rule	Details	Examples
Case sensitivity: Instrument commands are case sensitive. For best results, simply match the case shown in the command reference descriptions.	Function and attribute names should be in lowercase characters.	An example of the scriptVar.save() function (where test8 is the name of the script): `test8.save()`
	Parameters can use a combination of lowercase and uppercase characters. Attribute constants use uppercase characters	In the command below, which sets the format of data transmitted from the instrument to double-precision floating point, format.REAL64 is the attribute constant and format.data is the attribute command: `format.data = format.REAL64`

Syntax rules for instrument commands

Syntax rule	Details	Examples

White space: Not required in a function.	Functions can be sent with or without white spaces.	The following functions, which set digital I/O line 3 low, are equivalent:
		`digio.writebit(3,0)`
		`digio.writebit (3, 0)`

Function parameters: All functions are required to have a set of parentheses () immediately following the function.	You can specify the function parameters by placing them between the parentheses. Note that the parentheses are required even when there are no parameters specified.	The following function specifies all overlapped commands in the nodes in group G must complete before commands from other groups can execute:
		`waitcomplete(G)`
		
		The command below reads the value of the local time zone (no parameters are needed):
		`timezone = localnode.gettimezone()`

Multiple parameters: Must be separated by commas (,).	Some commands require multiple parameters, which must be separated by commas (,).	This command sets the beeper to emit a double-beep at 2400 Hz, with a beep sequence of 0.5 seconds on, 0.25 seconds off, and then 0.5 seconds on:
		`beeper.beep(0.5, 2400)`
		`delay(0.250)`
		`beeper.beep(0.5, 2400)`

Logical instruments

You would normally refer to all instrumentation in one enclosure or node as a single instrument. In the context of Test Script Processor (TSP®) scripting engine and instrument commands, it is useful to think of each individual subdivision in an enclosure, such as a card slot or the channels, as a stand-alone instrument. To avoid confusion, all subdivisions of the instrumentation in an enclosure are referred to as "logical instruments."

Each logical instrument is given a unique identifier in a system. These identifiers are used as part of all commands that control a given logical instrument. The logical instruments are:

- beeper
- bit
- digio
- display
- eventlog
- errorqueue
- format
- gpib
- lan
- status
- timer
- trigger
- tsplink

NOTE

Do not create variable names that are the same as names of logical instruments. Doing so will result in the loss of use of the logical instrument and its associated commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until you turn off the power to the instrument, and then turn it on again.

Time and date values

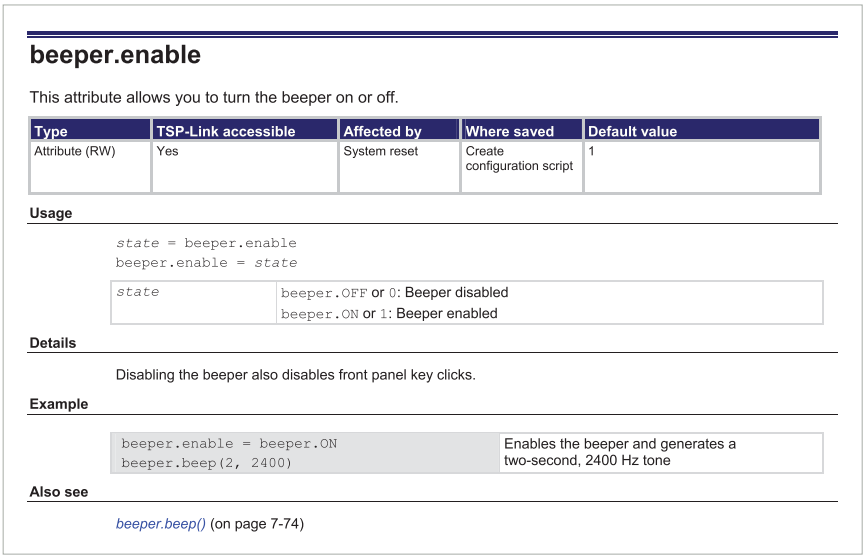
Time and date values are represented as the number of seconds since some base. Representing time as a number of seconds is referred to as "standard time format." There are three time bases:

- **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 command reference

The command reference contains detailed descriptions of each of the commands you can use to control your Model 2651A. Each command description is broken into several standardized subsections. The figure below shows an example of a command description.

Figure 92: Example instrument control library 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	System reset	Create configuration script	1

Usage

```
state = beeper.enable
beeper.enable = state
```

state	`beeper.OFF` or `0`: Beeper disabled `beeper.ON` or `1`: Beeper enabled

Details

Disabling the beeper also disables 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 7-74)

Each command listing is divided into five major categories of information about the command:

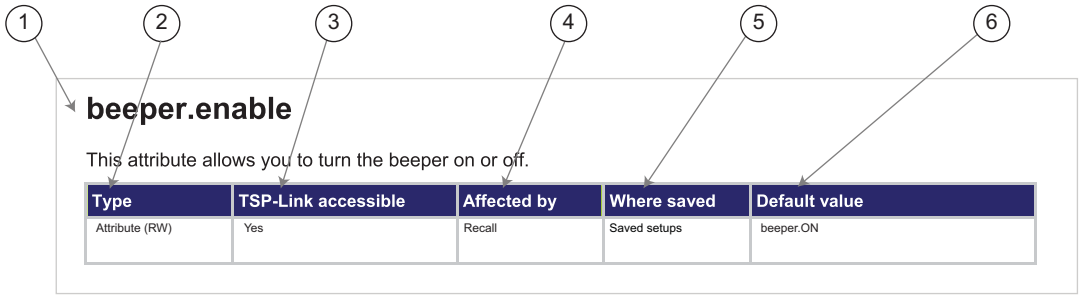
- Command name and standard parameters table
- Usage
- Details
- Example
- Also see

The content of each of these categories is described in the following topics.

Command name and standard parameters summary

The beginning of each instrument command description starts with the command name, followed by a table with relevant information for each command. Definitions for the numbered items in the figure below are listed following the figure.

Figure 93: Command name and standard parameters summary



1. **Instrument command name**. Signals the beginning of the command description and is followed by a brief description of what the command does.
2. **Type of command**. Options are:
   - **Function**. Function-based commands control actions or activities, but are not always directly related to instrument operation. Function names are always followed by a set of parentheses, for example, `digio.writeport(15)`**.** If the function does not need a parameter, the parentheses set remains empty., for example, `exit()`.
   - **Attribute (R), (RW),** or **(W)**. Attribute-based commands set or read the characteristics of an instrument feature or operation by defining a value. For example, a characteristic of a TSP-enabled instrument is the model number (`localnode.model`); another characteristic is the number of errors in the error queue (`errorqueue.count`). For many attributes, the defined value is a number or predefined constant. Attributes can be read-only (R), read-write (RW), or write-only (W), and can be used as a parameter of a function or assigned to another variable.
3. **TSP-Link accessible**. **Yes** or **No**; indicates whether or not the command can accessed through a TSP-Link network.
4. **Affected by**. Commands or actions that have a direct effect on the instrument control command.
   - **LAN restore defaults**
   - **Recall**
   - **Reset:** This command has varied effects depending on how it is used. Reset actions include:

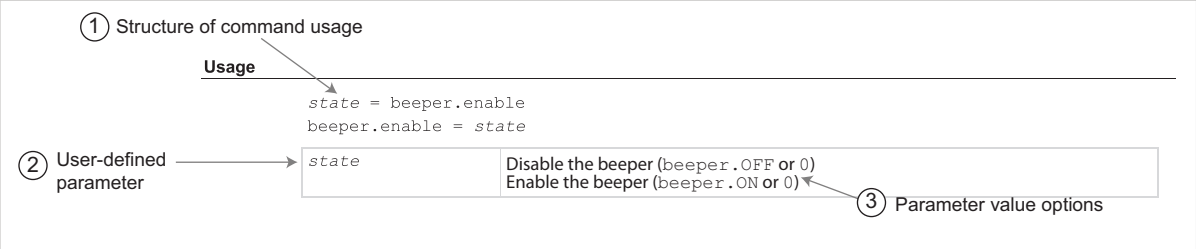
     - Digital I/O trigger reset
     - Instrument reset
     - LAN reset
     - LAN restore defaults
     - Local node reset
     - SMU reset
     - Status reset
     - Trigger blender reset
     - Trigger timer reset
     - TSP-Link trigger reset
     - SMU calibration lock
     - SMU calibration restore

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.
   - **System settings**
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 instrument control command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage; all possible command usage options are shown here.

Figure 94: Command usage section



1. **Structure of command usage:** Shows how the parts of the command should be organized.
2. **User-defined 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. For example, `beeper.beep(2, 2400)` generates a two-second, 2400 Hz tone.

NOTE

If there are optional parameters, they must be entered in the order presented in usage. You cannot leave out any parameters that precede the optional parameter, but you can leave off subsequent optional parameters. Optional parameters are shown as separate lines in usage, showing each permutation of the command. For example:

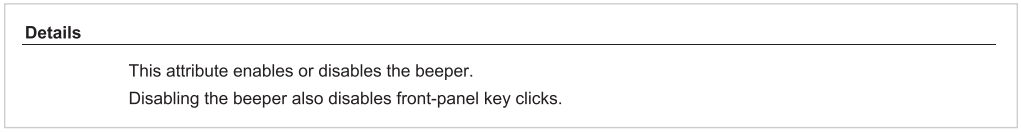
```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, columnStart)
text = display.gettext(embellished, row, columnStart, columnEnd)
```

3. **Parameter value options:** Displayed in order, from left to right, as they are presented in the Usage example.

Command details

This section lists additional information you need to know to successfully use the instrument control command.

Figure 95: Details section of command listing



Example section

The Example section of the instrument control command description shows some simple examples of how the command can be used.

Figure 96: Code examples in command listings

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 the output of the code.

Related commands and information

The **Also see** section of the instrument control command description lists commands that are related to the command being described.

Figure 97: Links to related commands and information

Commands

beeper.beep()

This function generates an audible tone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

duration	The amount of time to play the tone in seconds; the allowable range is 0.1 s to 100 s
frequency	The frequency of the tone in Hertz (Hz)

Details

The beeper will not sound if it is disabled. It can be disabled or enabled with the `beeper.enable` attribute, or through the front-panel Main Menu.

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 7-8)

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	Saved setups	beeper.ON

Usage

```
state = beeper.enable
beeper.enable = state
```

state	Disable the beeper (`beeper.OFF` or `0`) Enable the beeper (`beeper.ON` or `1`)

Details

Disabling the beeper also disables 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 7-8)

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*)

result	Result of the logical AND operation
value1	Operand for the logical AND operation
value2	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.bitor() (on page 7-9)
bit.bitxor() (on page 7-10)
Logical operators (on page 6-22)

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)
```

result	Result of the logical OR operation
value1	Operand for the logical OR operation
value2	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.bitand() (on page 7-9)
bit.bitxor() (on page 7-10)
Logical operators (on page 6-22)

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)
```

result	Result of the logical XOR operation
value1	Operand for the logical XOR operation
value2	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: 3.00000e+00

Also see

bit.bitand() (on page 7-9)
bit.bitor() (on page 7-9)
Logical operators (on page 6-22)

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)
```

result	Result of the bit manipulation
value	Specified number
index	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.get() (on page 7-11)
bit.set() (on page 7-13)
bit.test() (on page 7-15)
bit.toggle() (on page 7-15)
Logical operators (on page 6-22)

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)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* 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 will be 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.clear() (on page 7-11)
bit.set() (on page 7-13)
bit.test() (on page 7-15)
bit.toggle() (on page 7-15)
Logical operators (on page 6-22)

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)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get (1 to 32)
width	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:
	`6.00000e+00`

Also see

bit.get() (on page 7-11)
bit.set() (on page 7-13)
bit.setfield() (on page 7-14)
Logical operators (on page 6-22)

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*)

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* 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.clear() (on page 7-11)
bit.get() (on page 7-11)
bit.getfield() (on page 7-12)
bit.setfield() (on page 7-14)
bit.test() (on page 7-15)
bit.toggle() (on page 7-15)
Logical operators (on page 6-22)

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)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to set (1 to 32)
width	The number of bits to include in the field (1 to 32)
fieldValue	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.get() (on page 7-11)
bit.set() (on page 7-13)
bit.getfield() (on page 7-12)
Logical operators (on page 6-22)

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*)

result	Result of the bit manipulation
value	Specified number
index	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.clear() (on page 7-11)
bit.get() (on page 7-11)
bit.set() (on page 7-13)
bit.toggle() (on page 7-15)
Logical operators (on page 6-22)

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)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* 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.clear() (on page 7-11)
bit.get() (on page 7-11)
bit.set() (on page 7-13)
bit.test() (on page 7-15)
Logical operators (on page 6-22)

bufferVar.appendmode

This attribute sets the state of the reading buffer's append mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	0 (disabled)

Usage

```
state = bufferVar.appendmode
bufferVar.appendmode = state
```

state	The reading buffer append mode; set to one of the following: • `0`: Append mode off; new measure data overwrites the previous buffer content • `1`: Append mode on; appends new measure data to the present buffer content
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

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.

With append mode on, the first new measurement will be stored at rb[n+1], where n is the number of readings stored in buffer rb.

See the smu*X*.nvbuffer*Y* attribute for details on accessing dedicated reading buffers.

Example

`smua.nvbuffer1.appendmode = 1`	Append new readings to contents of dedicated reading buffer 1.

Also see

smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

bufferVar.basetimestamp

This attribute contains the timestamp of when the first reading was stored in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	0

Usage

basetime = *bufferVar*.basetimestamp

basetime	The timestamp of the first stored reading
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as smua.nvbuffer1)

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.

See the smu*X*.nvbuffer*Y* 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: 1.2830e+09
	The above output indicates that the timestamp is 1,283,000,000 seconds (which is Saturday, August 28, 2010 at 12:53:20 PM).

Also see

smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

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
```

cacheMode	The reading buffer cache mode; set to one of the following: • 0: Cache mode disabled (off) • 1: Cache mode enabled (on)
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as smua.nvbuffer1)

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 smu*X*.FILL_WINDOW. To avoid this, make sure to 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

`smua.nvbuffer1.cachemode = 1`	Enables reading buffer cache of dedicated reading buffer 1.

Also see

bufferVar.clearcache() (on page 7-20)
bufferVar.fillmode (on page 7-23)

bufferVar.capacity

This attribute contains the capacity of the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
maxNumber = bufferVar.capacity
```

maxNumber	The maximum number of readings the buffer can store
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

Details

This read-only attribute reads the number of readings that can be stored in the buffer.

The buffer's capacity 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

```
maxNumber = smua.nvbuffer1.capacity

print(capacity)
```

Reads the capacity of dedicated reading buffer 1.

Output: `1.49789e+05`

The above output indicates that the buffer can hold 149789 readings.

Also see

smuX.makebuffer() (on page 7-189)
smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

bufferVar.clear()

This function clears the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar.clear()
```

bufferVar	The reading buffer; this can be a dynamically allocated buffer (user defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

Details

This function clears all readings from the specified buffer.

Example

`smua.nvbuffer1.clear()`	Clears dedicated reading buffer 1.

Also see

smuX.nvbufferY (on page 7-208)
Reading buffers (on page 3-6)

bufferVar.clearcache()

This function clears the cache.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar`.clearcache()`

bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

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:

- initiating successive sweeps without reconfiguring the sweep measurements. Watch for this when running test script language (TSL) code remotely on more than one node, because values in the reading buffer cache may change while the TSL code is running.
- overwriting data in the reading buffer by setting the *bufferVar*`.fillmode` attribute to `smu`*X*`.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 dedicated reading buffer 1. cache.

Also see

bufferVar.fillmode (on page 7-23)
smuX.nvbufferY (on page 7-208)
Reading buffers (on page 3-6)
Removing stale values from the reading buffer (on page 6-60)

bufferVar.collectsourcevalues

This attribute sets whether or not source values will be stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	0 (disabled)

Usage

```
state = bufferVar.collectsourcevalues
bufferVar.collectsourcevalues = state
```

state	Source value collection status; set to one of the following: • 0: Source value collection disabled (off) • 1: Source value collection enabled (on)
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as smua.nvbuffer1)

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.

When on, source values will be stored with readings in the buffer. This requires four extra bytes of storage per reading. Turning on additional collection items, such as source values (this attribute) and timestamps, 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.

This value, off or on, can only be changed when the buffer is empty. Empty the buffer using the *bufferVar*.clear() function.

Example

`smua.nvbuffer1.collectsourcevalues = 1`	Include source values with readings for dedicated reading buffer 1.

Also see

bufferVar.clear() (on page 7-19)
smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

bufferVar.collecttimestamps

This attribute sets whether or not timestamp values will be stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	0 (disabled)

Usage

```
state = bufferVar.collecttimestamps
bufferVar.collecttimestamps = state
```

state	Timestamp value collection status; set to one of the following: • 0: Timestamp value collection disabled (off) • 1: Timestamp value collection enabled (on)
bufferVar	The reading buffer; this can be a dynamically allocated buffer (user defined), or a dedicated reading buffer (such as smua.nvbuffer1)

Details

Assigning a value to this attribute enables or disables the storage of timestamps. Reading this attribute returns the state of timestamp collection. This value can only be changed with an empty buffer. Empty the buffer using the *bufferVar*.clear() function.

When on, timestamp values will be stored with readings in the buffer. This requires four extra bytes of storage per 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.

This value, off or on, can only be changed when the buffer is empty. Empty the buffer using the *bufferVar*.clear() function.

Example

`smua.nvbuffer1.collecttimestamps = 1`	Include timestamps with readings for dedicated reading buffer 1.

Also see

bufferVar.clear() (on page 7-19)
smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

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	Not applicable	0

Usage

```
fillCount = bufferVar.fillcount
bufferVar.fillcount = fillCount
```

fillCount	The reading buffer fill count
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as smua.nvbuffer1)

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 will overwrite the reading at index 1, the reading after that will overwrite the reading at index 2, and so on.

This attribute is only used when the *bufferVar*.fillmode attribute is set to smu*X*.FILL_WINDOW.

Example

`smua.nvbuffer1.fillcount = 50`	Sets fill count of dedicated reading buffer 1 to 50.

Also see

bufferVar.fillmode (on page 7-23)

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	Not applicable	smuX.FILL_ONCE

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

fillMode	The reading buffer fill mode; set to one of the following: • 0 or smu*X*.FILL_ONCE: Do not overwrite old data • 1 or smu*X*.FILL_WINDOW: New readings restart at index 1 after acquiring reading at index *bufferVar*.fillcount
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as smua.nvbuffer1)

Details

When this attribute is set to smu*X*.FILL_ONCE, the reading buffer will not overwrite readings. If the buffer fills up, new readings will be discarded.

When this attribute is set to smu*X*.FILL_WINDOW, new readings will be added after existing data until the buffer holds *bufferVar*.fillcount elements. Continuing the sequence, the next reading will overwrite the reading at index 1, the reading after that will overwrite the reading at index 2, and so on.

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 7-22)

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	Not applicable	Not saved	Not applicable

Usage

```
numberOfReadings = bufferVar.n
```

numberOfReadings	The number of readings stored in the buffer
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

Details

This read-only attribute contains the number of readings presently stored in the buffer.

Example

```
numberOfReadings = smua.nvbuffer1.n

print(numberOfReadings)
```

Reads the number of readings stored in dedicated reading buffer 1.

Output: `1.250000+02`

The above output indicates that there are 125 readings stored in the buffer.

Also see

smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

bufferVar.timestampresolution

This attribute contains the timestamp's resolution.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	1e-6

Usage

```
resolution = bufferVar.timestampresolution
bufferVar.timestampresolution = resolution
```

resolution	Timestamp resolution in seconds
bufferVar	The reading buffer; this can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

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. This value can be increased for very long tests.

The value specified when setting this attribute will be rounded to an even power of 2 µs.

Example

`smua.nvbuffer1.timestampresolution = 0.000008`	Sets the timestamp resolution of dedicated reading buffer 1 to 8 µs.

Also see

bufferVar.clear() (on page 7-19)
smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.Y() (on page 7-238)
Reading buffers (on page 3-6)

ConfigPulseIMeasureV()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in amperes
level	Pulse level in amperes
limit	Voltage limit (for example, compliance) in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train will wait for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train will generate a trigger output immediately before the start of `ton` (this parameter is optional)
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s.
sync_in_abort	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`.

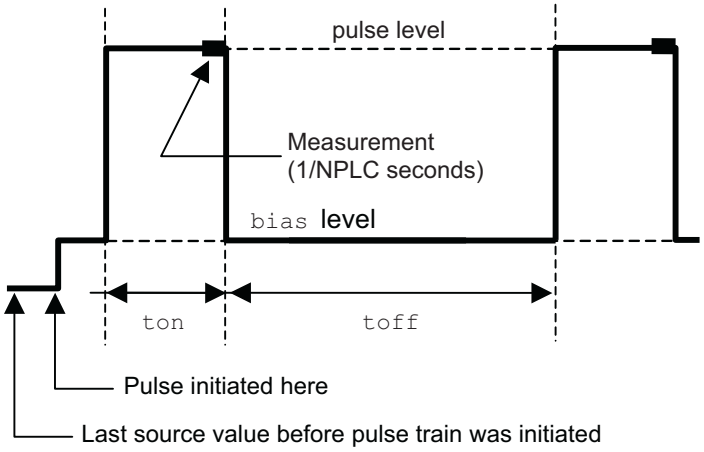
Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the *buffer* input parameter.

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 of the pulse dimensions are achievable, and if they are, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid `tag`.

Figure 98: ConfigPulseIMeasureV()



Example

`ConfigPulseIMeasureV(smua, 0, 5, 10, 0.001, 0.080, 1, smua.nvbuffer1, 1)`	Set up a pulse train that uses System SourceMeter® instrument channel A. The pulse amplitude will be 5 A and will return to 0 A after 1 ms. The pulse will remain at 0 A for 80 ms and the voltage limit will be 10 V during the pulse. The pulse train will consist of only 1 pulse, and this pulse will be assigned a *tag* index of 1.

Also see

InitiatePulseTest() (on page 7-97)

ConfigPulseIMeasureVSweepLin()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag; this flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, `msg` will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in amperes
start	Pulse sweep start level in volts
stop	Pulse sweep stop level in volts
limit	Voltage limit (for example, compliance) in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train will wait for a trigger input before executing each pulse (this parameter is optional)
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train will generate a trigger output immediately before the start of `ton` (this parameter is optional)
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	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.

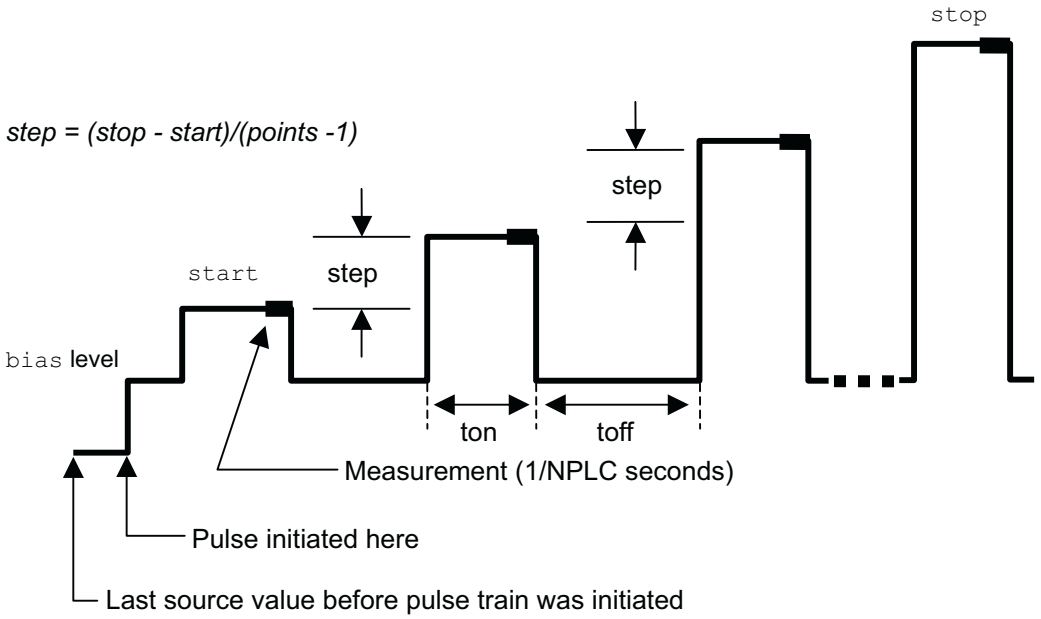
Configures a linear pulsed current sweep with a voltage measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse will be *start* amperes; the magnitude of the last pulse will be *stop* amperes. The magnitude of each pulse in between will be *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 simply checks to see if all of the pulse dimensions are achievable, and if they are, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid `tag`.

Figure 99: ConfigPulseIMeasureVSweepLin()



Example

`ConfigPulseIMeasureVSweepLin(smua, 0, 0.01, 0.05, 1, 1e-3, 0.1, 20, smua.nvbuffer2, 3)`	Set up a pulsed sweep that will use SourceMeter instrument channel A. The pulsed sweep will start at 10 mA, end at 50 mA, and return to a 0 mA bias level between pulses. Each pulsed step will be on for 1 ms, and then at the bias level for 100 ms. The voltage limit will be 1 V during the entire pulsed sweep. The pulse train will be comprised of 20 pulsed steps and the pulse train will be assigned a *tag* index of 3.

Also see

InitiatePulseTest() (on page 7-97)

ConfigPulseIMeasureVSweepLog()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag; this flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the `f` flag is `false`, *msg* will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in amperes
start	Pulse sweep start level in amperes
stop	Pulse sweep stop level in amperes
limit	Voltage limit (for example, compliance) in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train will wait for a trigger input before executing each pulse (this parameter is optional)
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train will generate a trigger output immediately before the start of *ton* (this parameter is optional)
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	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.

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 will be *start* amperes; the magnitude of the last pulse will be *stop* amperes. The magnitude of each pulse in between will be LogStep_n amperes larger than the previous pulse, where:

LogStepSize = (log10(stop) - log10(start)) / (points -1)

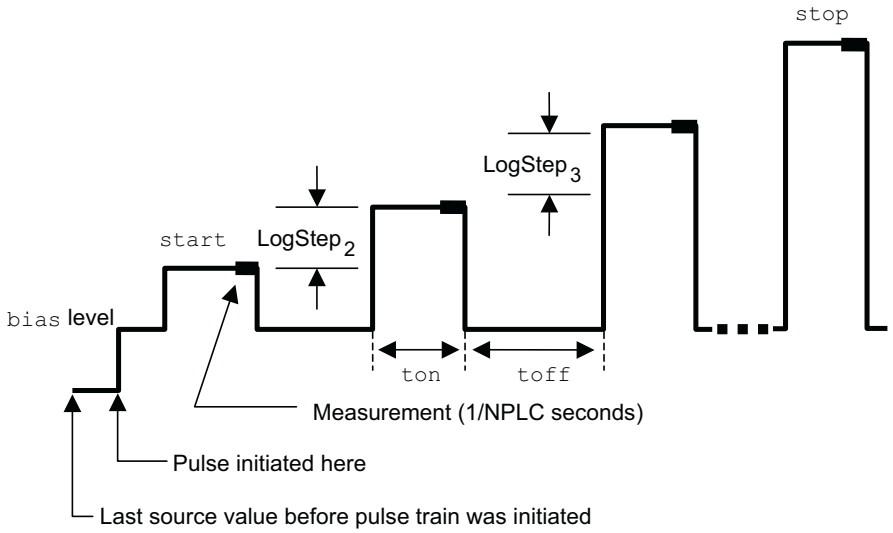
*LogStepn = (n - 1) * (LogStepSize), where* n = [1, points]

*SourceStepLeveln = antilog(LogStepn) * 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 are achievable, and if they are, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid *tag*.

Figure 100: ConfigPulseIMeasureVSweepLog()

LogStepSize = (log10(stop) - log10(start)) / (points - 1)
*$LogStep_n$ = (n - 1) * (LogStepSize)* *where* n = [1, *points*]
*$SourceStepLevel_n$ = antilog ($LogStep_n$) * start*



Example

`ConfigPulseIMeasureVSweepLog(smua, 0, 1e-3, 0.01, 1, 1e-3, 10e-3, 10, smua.nvbuffer1, 5)`	Set up a pulsed logarithmic sweep that uses SourceMeter instrument channel A. The pulsed sweep will start at 1 mA, end at 10 mA, and return to a 0 A bias level between pulses. Each pulsed step will be on for 1 ms, and then at the bias level for 10ms. The voltage limit will be 1 V during the entire pulsed sweep. The pulse train will be comprised of 10 pulsed steps, and the pulse train will be assigned a *tag* index of 5.

Also see

InitiatePulseTest (see "InitiatePulseTest()" on page 7-97)

ConfigPulseVMeasureI()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag; this flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the `f` flag is `false`, *msg* will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in volts
level	Pulse level in volts
limit	Voltage limit (for example, compliance) in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train will wait for a trigger input before executing each pulse (this parameter is optional)
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train will generate a trigger output immediately before the start of `ton` (this parameter is optional)
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	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`

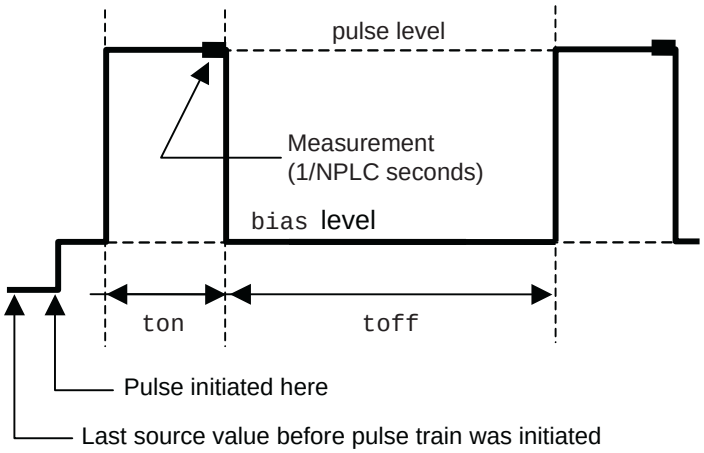
Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

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 simply checks to see if all of the pulse dimensions are achievable, and if they are, assigns the indicated *tag* or index to the pulse train.The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid `tag`.

Figure 101: ConfigPulseVMeasureI()



Example 1

```
ConfigPulseVMeasureI(smua, 0, 20, 1,
   0.001, 0.080, 10, smua.nvbuffer1, 2)
```

Set up a pulse train that uses SourceMeter instrument channel A. The pulse amplitude will be 20 V and will return to 0 V after 1 ms. The pulse will remain at 0 V for 80 ms, and the current limit will be 1 A during the pulse. The pulse train will consist of 10 pulses, and the pulse train will be 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 7-97)

ConfigPulseVMeasureISweepLin()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, `msg` will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in volts
start	Pulse sweep start level in volts
stop	Pulse sweep stop level in volts
limit	Voltage limit (for example, compliance) in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train will wait for a trigger input before executing each pulse (this parameter is optional)
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train will generate a trigger output immediately before the start of `ton` (this parameter is optional)
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	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 current measurements are stored in the reading buffer specified by the *buffer* input parameter.

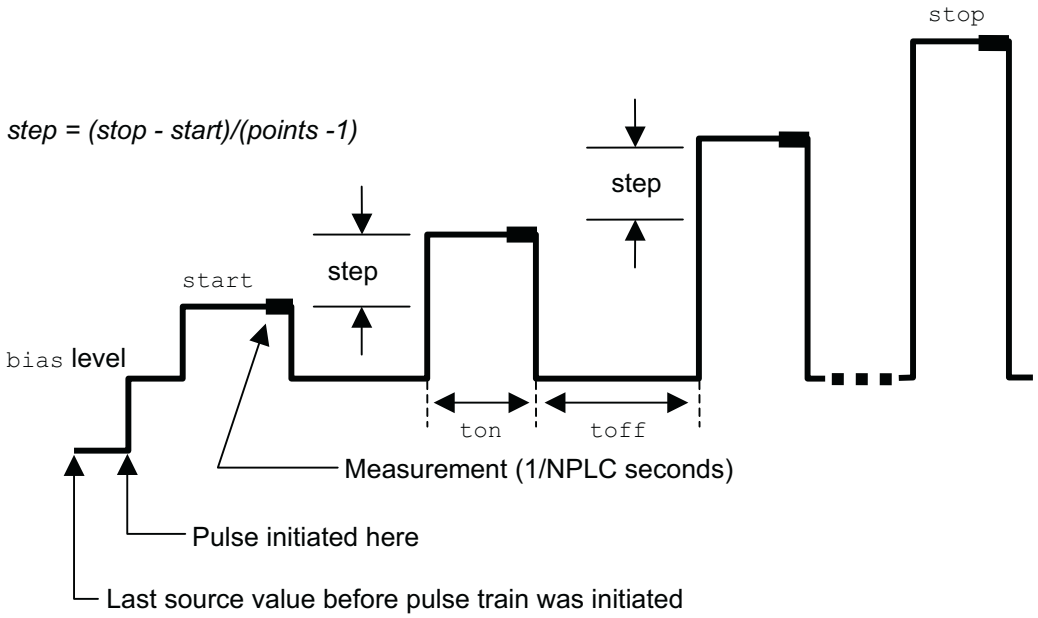
Configures a linear pulsed voltage sweep with a current measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse will be *start* volts; the magnitude of the last pulse will be *stop* volts. The magnitude of each pulse in between will be *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 simply checks to see if all of the pulse dimensions are achievable, and if they are, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(tag)` function is used to initiate a pulse train assigned to a valid `tag`.

Figure 102: ConfigPulseVMeasureISweepLin()



Example

`ConfigPulseVMeasureISweepLin(smua, 0, 1, 10, 1, 10e-3, 20e-3, 16, smua.nvbuffer1, 4)`	Set up a pulsed sweep that will use System SourceMeter instrument channel A. The pulsed sweep will start at 1 V, end at 10 V, and return to a 0 V bias level between pulses. Each pulsed step will be on for 10 ms, and then at the bias level for 20 ms. The current limit will be 1 A during the entire pulsed sweep. The pulse train will be comprised of 16 pulsed steps, and the pulse train will be assigned a *tag* index of 4.

Also see

InitiatePulseTest() (on page 7-97)

ConfigPulseVMeasureISweepLog()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the `f` flag is `false`, `msg` will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in volts
start	Pulse sweep start level in volts
stop	Pulse sweep stop level in volts
limit	Voltage limit (for example, compliance) in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train will wait for a trigger input before executing each pulse (this parameter is optional)
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train will generate a trigger output immediately before the start of `ton` (this parameter is optional)
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	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 current measurements are stored in the reading buffer specified by the *buffer* input parameter.

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 will be *start* volts; the magnitude of the last pulse will be *stop* volts. The magnitude of each pulse in between will be $LogStep_n$ volts larger than the previous pulse, where:

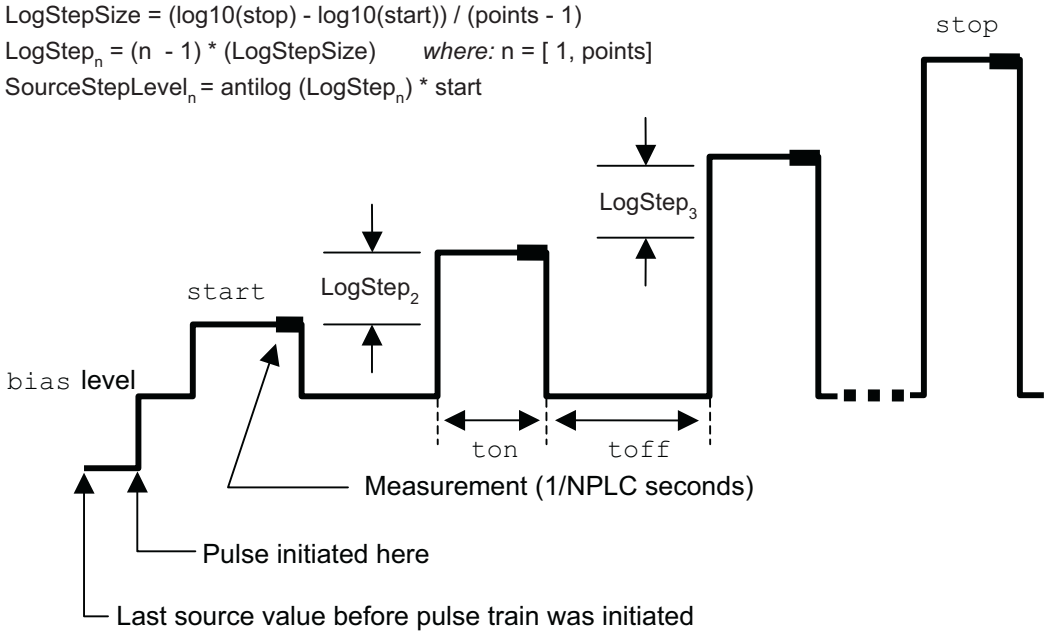
LogStepSize = (log10(stop) - log10(start)) / (points -1)

$LogStep_n$ *= (n - 1) * (LogStepSize), where n = [1, points]*

$SourceStepLevel_n$ *= antilog(*$LogStep_n$*) * 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 are achievable, and if they are, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid `tag`.

Figure 103: ConfigPulseVMeasureISweepLog()



Example

`ConfigPulseVMeasureISweepLog(smua, 0, 1, 10, 1, 10e-3, 20e-3, 10, smua.nvbuffer1, 6)`	Set up a pulsed logarithmic sweep that will use SourceMeter instrument channel A. The pulsed sweep will start at 1 V, end at 10 V, and return to a 0 V bias level between pulses. Each pulsed step will be on for 10 ms, and then at the bias level for 20 ms. The current limit will be 1 A during the entire pulsed sweep. The pulse train will be comprised of 10 pulsed steps, and the pulse train will be assigned a *tag* index of 6.

Also see

InitiatePulseTest() (on page 7-97)

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = dataqueue.add(value)
result = dataqueue.add(value, timeout)
```

result	The resulting value of `true` or `false` based on the success of the function
value	The data item to add; *value* can be of any type
timeout	The maximum number of seconds to wait for space in the data queue

Details

You cannot use the *timeout* value when accessing the data queue from a remote node (you can only use the *timeout* value while adding data to the local data queue).

The *timeout* value is ignored if the data queue is not full.

The `dataqueue.add()` function returns `false`:

- If the timeout expires before space is available in the data queue
- If the data queue is full and a *timeout* value is not specified

If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
dataqueue.add(10)
dataqueue.add(11, 2)
result = dataqueue.add(12, 3)
if result == false then
   print("Failed to add 12 to the dataqueue")
end
print("The dataqueue contains:")
while dataqueue.count > 0 do
   print(dataqueue.next())
end
```

Clear the data queue.

Each line adds one item to the data queue.

Output:

```
The dataqueue contains:
1.00000e+01
1.10000e+01
1.20000e+01
```

Also see

dataqueue.CAPACITY (on page 7-39)
dataqueue.clear() (on page 7-39)
dataqueue.count (on page 7-40)
dataqueue.next() (on page 7-41)

dataqueue.CAPACITY

This constant is the maximum number of entries that you can store in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
count = dataqueue.CAPACITY
```

count	The variable assigned the value of `dataqueue.CAPACITY`

Details

This constant always returns the maximum number of entries that can be stored in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

Add items to the data queue until it is at capacity.

Output:

```
There are 128 items in the data
   queue
```

Also see

dataqueue.add() (on page 7-38)
dataqueue.clear() (on page 7-39)
dataqueue.count (on page 7-40)
dataqueue.next() (on page 7-41)

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.

The function 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 7-38)
dataqueue.CAPACITY (on page 7-39)
dataqueue.count (on page 7-40)
dataqueue.next() (on page 7-41)

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	Power cycle Dataqueue add Dataqueue clear Dataqueue next	Not saved	Not applicable

Usage

```
count = dataqueue.count
```

count	The number of items in the data queue

Details

The count gets updated as entries are added and read from the data queue. It is also updated when the dataqueue is cleared.

The number of items in the data queue are controlled through `dataqueue.add()`, `dataqueue.next()`, and `dataqueue.clear()`, with a maximum of `dataqueue.CAPACITY` items.

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")
```

Add items to the data queue until it is at capacity.

Output:

```
There are 128 items in the data queue
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 7-38)
dataqueue.CAPACITY (on page 7-39)
dataqueue.clear() (on page 7-39)
dataqueue.next() (on page 7-41)

dataqueue.next()

This function removes the next entry from the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dataqueue.next()
value = dataqueue.next(timeout)
```

value	The next entry in the data queue
timeout	The number of seconds to wait for data in the queue

Details

If the data queue is empty, the function waits up to the *timeout* value.

If data is not available in the data queue before the *timeout* expires, the return value is `nil`.

The entries in the data queue are removed in first-in, first-out (FIFO) order.

If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
for i = 1, 10 do
   dataqueue.add(i)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")

while dataqueue.count > 0 do
   x = dataqueue.next()
   print(x)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

Clears the data queue, adds ten entries, then reads the entries from the data queue.

Output:

```
There are 10 items in the data
    queue
1.00000e+00
2.00000e+00
3.00000e+00
4.00000e+00
5.00000e+00
6.00000e+00
7.00000e+00
8.00000e+00
9.00000e+00
1.00000e+01
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 7-38)
dataqueue.CAPACITY (on page 7-39)
dataqueue.clear() (on page 7-39)
dataqueue.count (on page 7-40)

delay()

This function delays the execution of the commands that follow it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
delay(seconds)
```

seconds	The number of seconds to delay, maximum 100,000

Details

It is impossible to delay for zero seconds.

The system delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the system to delay 5 µs to 10 µs (typical) more than the requested delay.

Example 1

```
beeper.beep(0.5, 2400)
delay(0.250)
beeper.beep(0.5, 2400)
```

Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

```
dataqueue.clear()
dataqueue.add(35)
timer.reset()
delay(0.5)
dt = timer.measure.t()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, then delay 0.5 seconds before reading it.

Output:

```
Delay time was 0.500099
3.50000e+01
```

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)
```

data	A custom variable that stores the state of the I/O line
N	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:
`1.00000e+00`

Also see

Digital I/O port
digio.readport() (on page 7-43)
digio.writebit() (on page 7-51)
digio.writeport() (on page 7-52)

digio.readport()

This function reads the digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readport()
```

data	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the 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: `1.70000e+02` This is binary 10101010

Also see

Digital I/O port

digio.readbit() (on page 7-43)
digio.writebit() (on page 7-51)
digio.writeport() (on page 7-52)

digio.trigger[N].assert()

This function asserts a trigger 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 set pulsewidth determines how long the trigger is asserted.

Example

`digio.trigger[2].assert()`	Asserts a trigger on digital I/O line 2.

Also see

digio.trigger[N].pulsewidth (on page 7-48)

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()
```

N	Digital I/O trigger line (1 to 14)

Details

The event detector of a trigger recalls if a trigger event has been detected since the last `digio.trigger[N].wait()` command. This function clears the event detector of the specified trigger line, discards the previous history of the trigger line, and clears the `digio.trigger[N].overrun` attribute.

Example

`digio.trigger[2].clear()`	Clears the trigger event detector on I/O line 2.

Also see

digio.trigger[N].overrun (on page 7-47)
digio.trigger[N].wait() (on page 7-51)

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

eventID	The trigger event number
N	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 other object's stimulus attribute to the value of this constant.

Example 1

`digio.trigger[5].stimulus = digio.trigger[3].EVENT_ID`	Uses a trigger event on digital I/O trigger line 3 to be the stimulus for digital I/O trigger line 5.

Example 2

`smua.trigger.arm.stimulus = digio.trigger[3].EVENT_ID`	Uses a trigger event on digital I/O trigger line 3 to be the trigger stimulus for the SMU.

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 N reset	Not saved	digio.TRIG_BYPASS

Usage

```
triggerMode = digio.trigger[N].mode
digio.trigger[N].mode = triggerMode
```

triggerMode	The trigger mode; see **Details** for values
N	Digital I/O trigger line (1 to 14)

Details

Set *triggerMode* to one of the following values:

Trigger mode values

triggerMode	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 behaves similar to digio.TRIG_RISINGA. If the programmed state of the line is low, the digio.TRIG_RISING mode behaves similar to digio.TRIG_RISINGM. This setting should only be used 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, as digio.TRIG_SYNCHRONOUS is provided for compatibility with the digital I/O and TSP-Link triggering on other Keithley Instruments products.

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].reset() (on page 7-49)
digio.writebit() (on page 7-51)
digio.writeport() (on page 7-52)
Sweep operation (on page 3-19)

digio.trigger[N].overrun

Use this attribute to read the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Digital I/O trigger N clear Digital I/O trigger N reset	Not saved	Not applicable

Usage

overrun = digio.trigger[*N*].overrun

overrun	Trigger overrun state (`true` or `false`)
N	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].reset() (on page 7-49)

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 N reset	Not saved	10e-6

Usage

```
width = digio.trigger[N].pulsewidth
digio.trigger[N].pulsewidth = width
```

width	The pulse width (seconds)
N	Digital I/O trigger line (1 to 14)

Details

Setting *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 7-44)
digio.trigger[N].reset() (on page 7-49)
digio.trigger[N].release() (on page 7-48)

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()
```

N	Digital I/O trigger line (1 to 14)

Details

Releases a trigger that was asserted with an indefinite pulse width time, as well as a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line (*N*) is affected.

Example

```
digio.trigger[4].release()
```

Releases digital I/O trigger Line 4.

Also see

digio.trigger[N].pulsewidth (on page 7-48)

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()
```

N	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`.

Also see

digio.trigger[N].mode (on page 7-46)
digio.trigger[N].overrun (on page 7-47)
digio.trigger[N].pulsewidth (on page 7-48)
digio.trigger[N].stimulus (on page 7-49)

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	Not saved	0

Usage

```
triggerStimulus = digio.trigger[N].stimulus
digio.trigger[N].stimulus = triggerStimulus
```

triggerStimulus	The event identifier for the triggering event
N	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.

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
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the 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 measure 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 `N`
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.

* 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 7-44)
digio.trigger[N].clear() (on page 7-44)
digio.trigger[N].reset() (on page 7-49)

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)
```

triggered	`true` if a trigger is detected, or `false` if no triggers are detected during the timeout period
N	Digital I/O trigger line (1 to 14)
timeout	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `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 re-armed. 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 7-44)

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)
```

N	Digital I/O trigger line (1 to 14)
data	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 7-43)
digio.readport() (on page 7-43)
digio.trigger[N].mode (on page 7-46)
digio.writeport() (on page 7-52)
digio.writeprotect (on page 7-53)

digio.writeport()

This function writes to all digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writeport(data)
```

data	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 7-43)
digio.readport() (on page 7-43)
digio.writebit() (on page 7-51)
digio.writeprotect (on page 7-53)

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	Saved setups	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 7-51)
digio.writeport() (on page 7-52)

display.clear()

This function clears all lines of the 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 display.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped, nonblocking commands. That is, 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 update the physical display as soon as processing time becomes available.

Also see

display.setcursor() (on page 7-69)
display.settext() (on page 7-70)

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()
```

annunciators	The bitmasked value showing 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_TRIG
* (star)	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

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


rem = bit.bitand(myAnnunciators, 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 7-9)

display.getcursor()

This function reads the present position of the cursor on the front panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

row	The row where the cursor is: `1` (top row); `2` (bottom row)
column	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
style	Visibility of the cursor: `0` (invisible cursor); `1` (blinking cursor)

Details

This function switches the display to the user screen (the text set by `display.settext()`), and then returns values to indicate the cursor's row and column position and cursor style.

Columns are numbered from left to right on the display.

Example 1

```
myRow, myColumn = display.getcursor()
print(myRow, myColumn)
```

This example reads the cursor position into local variables and prints them. Example output:

```
1.00000e+00     1.00000e+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:

```
1.00000e+00   3.00000e+00
    0.00000e+00
```

Also see

display.gettext() (on page 7-58)
display.screen (on page 7-67)
display.setcursor() (on page 7-69)
display.settext() (on page 7-70)

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()
```

keyCode	A returned value that represents the last front-panel key pressed. See **Details** for more information.

Details

A history of the key code for the last pressed 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, `display.locallockout` must be used.

The table below lists the *keyCode* value for each front-panel action.

Key codes

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

NOTE

When using this function, use built-in constants such as `display.KEY_RIGHT` (rather than the numeric value of `103`). This will allow for better forward compatibility with firmware revisions.

The OUTPUT ON/OFF control cannot be tracked by this function.

Example

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

On the front panel, press the **MENU** key and then send the code to the left. This retrieves the key code for the last pressed key.

Output:

```
6.800000e+01
```

Also see

display.locallockout (on page 7-64)

display.sendkey() (on page 7-68)

display.gettext()

This function reads the text displayed on the instrument 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)
```

text	The returned value, which contains the text that is presently displayed
embellished	Indicates type of returned text: `false` (simple text); `true` (text with embedded character codes)
row	Selects the row from which to read the text: `1` (row 1); `2` (row 2). If *row* is not included, both rows of text are read
columnStart	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
columnEnd	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 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 7-70).

The display is not switched to the user screen (the screen set using `display.settext()`). Text will be read from the active screen.

Example 1

```
display.clear()
display.setcursor(1, 1)
display.settext("ABCDEFGHIJ$DKLMNOPQRST")
display.setcursor(2, 1)
display.settext("abcdefghijklm$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:

```
ABCDEFGHIJKLMNOPQRST$Nabcdefghijklmnopqrstuvwxyz123456
$RABCDEFGHIJ$DKLMNOPQRST$N$Rabcdefghijklm$Bnopqrstuvwxyz$F123456
abcdefghijklmnopqrstuvwxyz123456
$Rijklm$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 7-54)
display.getcursor() (on page 7-56)
display.setcursor() (on page 7-69)
display.settext() (on page 7-70)

display.inputvalue()

This function displays a formatted input field on the instrument 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)
```

format	A string that defines how the input field is formatted. See **Details** for more information
default	The default value for the input value
minimum	The minimum input value
maximum	The maximum input value

Details

format uses 0s, the decimal point, polarity sign, and exponents to define how the input field is formatted. *format* 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. You can use up to six 0s.	+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.

The *minimum* and *maximum* parameters can be used to limit the values that can be entered. When + is 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 ±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 ("+" sign used for the value field and/or 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 will also toggle 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, value 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 7-66)
display.setcursor() (on page 7-69)
display.settext() (on page 7-70)

display.loadmenu.add()

This function adds an entry to the User menu, which can be accessed by pressing the **LOAD** key on the instrument 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)
```

displayName	The name that is added to the User menu
code	The code that is run from the User menu
memory	Determines if code is saved to nonvolatile memory: `0` or `display.DONT_SAVE`: Does not save the code to nonvolatile memory `1` or `display.SAVE`: Saves the code to nonvolatile memory (default)

Details

After adding code to the load menu, you can run it from the front panel by pressing the **LOAD** key, then selecting **USER** to select from the available code to load. Pressing the **RUN** key will then run 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 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 will be lost when the Model 2651A is turned off. See **Example 2** below.

If you do not make a selection for *memory*, the code is automatically saved to nonvolatile memory.

Quick Tip

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 front panel.

Example 1

`display.loadmenu.add("Test9", "Test9()")`	Assume a user script named "Test9" has been 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" has already been 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 TESTS 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 front panel USER TESTS 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 again run from the 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 front panel "USER TESTS" load menu for the code `testpart([[Part1]], 5.0)`, and saves it in nonvolatile memory.

Also see

display.loadmenu.delete() (on page 7-63)

display.loadmenu.catalog()

This function creates an iterator for the user menu items accessed using the LOAD key on the instrument 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
```

displayName	The name displayed in the menu
code	The code associated with the *displayName*
body	The body of the code to process the entries in the loop

Details

Each time through the loop, *displayName* and *code* will take on the values in the User menu.

The list is traversed is 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 7-61)
display.loadmenu.delete() (on page 7-63)

display.loadmenu.delete()

This function removes an entry from the User menu, which can be accessed using the **LOAD** key on the instrument front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.loadmenu.delete(displayName)
```

displayName	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 script named "Test9"

Output:

```
Test   DUT1() beeper.beep(2, 500)
Part1   testpart([[Part1]], 5.0)
```

Also see

display.loadmenu.add() (on page 7-61)
display.loadmenu.catalog() (on page 7-62)

display.locallockout

This attribute describes whether or not the EXIT (LOCAL) key on the instrument front panel is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	display.UNLOCK

Usage

```
lockout = display.locallockout
display.locallockout = lockout
```

lockout	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/remote operation.

Example

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

Disables the front-panel EXIT (LOCAL) key.

Also see

None

display.menu()

This function presents a menu on the front panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
selection = display.menu(name, items)
```

selection	Name of the variable that holds the menu item selected
name	Menu name to display on the top line
items	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 will not abort the script while the menu is displayed, but it will 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 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	Saved setups	display.ENABLE

Usage

numericKeypad = display.numpad
display.numpad = *numericKeypad*

numericKeypad	Enable the numeric keypad feature (`1` or `display.ENABLE`) Disable the numeric keypad feature (`0` or `display.DISABLE`)

Details

The numeric keypad feature is only available when editing a numeric value at the same time that the EDIT indicator is lit.

Example

`display.numpad = display.ENABLE`	Turn on the numeric keypad feature.

Also see

Setting a value (on page 2-20)

display.prompt()

This function prompts the user to enter a parameter from the 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)
```

format	A string that defines how the input field is formatted; see **Details** for more information
units	Set the units text string for the top line (eight characters maximum); this indicates the units (for example, "V" or "A") for the value
help	Text string to display on the bottom line (32 characters maximum)
default	The value that is shown when the value is first displayed
minimum	The minimum input value that can be entered
maximum	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
```

format uses 0s, the decimal point, polarity sign and exponents to define how the input field is formatted.

format can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially. Include a "+" sign for positive/negative exponent entry. Do not include the "+" 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 "+" 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 0s.	`+00.0000E+00`
.	The decimal point where needed for the value.	`+0.00`

minimum and *maximum* can be used to limit the values that can be entered. When + is selected for *format*, the minimum limit must be more than or equal to zero. When limits are used, the operator cannot enter values above or below these limits.

The input value is limited to ±1e37.

After sending this command, script execution pauses for the operator to enter a value and press **ENTER**.

For positive and negative entry ("+" sign used for the value field and/or the exponent field), polarity of a non-zero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel ⟳. Polarity will also toggle 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 exectuing 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 7-59)

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	Saved setups	display.SMUA

Usage

```
displayID = display.screen
display.screen = displayID
```

displayID	One of the following values: • `0` or `display.SMUA`: Displays source-measure and compliance • `3` or `display.USER`: Displays the user screen

Details

Setting this attribute selects the display screen for the front panel. This performs the same action as pressing the **DISPLAY** key on the 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 the SMU.

Also see

display.settext() (on page 7-70)

display.sendkey()

This function sends a code that simulates the action of a 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 to virtually press. See **Details** for more information

Details

This command simulates the pressing of a front panel key or navigation wheel, or the turning the navigation wheel one click to the left or right.

Key codes

Value	Key list	Value	Key list
0	display.KEY_NONE	82	display.KEY_ENTER
65	display.KEY_RANGEUP	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	92	display.KEY_TRIG
72	display.KEY_DISPLAY	93	display.KEY_LIMITA
73	display.KEY_AUTO	94	display.KEY_SPEEDA
75	display.KEY_EXIT	95	display.KEY_LOAD
77	display.KEY_FILTERA	97	display.WHEEL_ENTER
78	display.KEY_STORE	103	display.KEY_RIGHT
79	display.KEY_SRCA	104	display.KEY_LEFT
80	display.KEY_CONFIG	107	display.WHEEL_LEFT
81	display.KEY_RANGEDOWN	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

`display.sendkey(display.KEY_RUN)`	Simulates pressing the RUN key.

Also see

Front panel (on page 2-1)

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)
```

row	The row number for the cursor (1 or 2)
column	The active column position to set; row 1 has columns 1 to 20, row 2 has columns 1 to 32
style	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, nonblocking commands. That is, 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 update the physical 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 the message "Hello World" on the instrument front panel, approximately center. Note that the top line of text is larger than the bottom.

Also see

display.clear() (on page 7-54)
display.getcursor() (on page 7-56)
display.gettext() (on page 7-58)
display.screen (on page 7-67)
display.settext() (on page 7-70)

display.settext()

This function displays text on the user screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.settext(text)
```

text	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, nonblocking commands. That is, 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 update the physical 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 is left at the end of the line.

The text remains on the display until replaced or cleared.

The following character codes can be also be included in the text string:

Display character codes

Character Code	Description
`$N`	Newline, starts text on the next line. If the cursor is already on line 2, text will be ignored after the `$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 $BBlinking$N")
display.settext("$DDim $FBackgroundBlink"
   .. "$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 7-54)
display.getcursor() (on page 7-56)
display.gettext() (on page 7-58)
display.screen (on page 7-67)
display.setcursor() (on page 7-69)

display.smuX.digits

This attribute sets the display resolution of the selected measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall	Saved setups	display.DIGITS_5_5

Usage

```
digits = display.smuX.digits
display.smuX.digits = digits
```

digits	Set digits to one of the following values: • Select 4-1/2 digit resolution (`4` or `display.DIGITS_4_5`) • Select 5-1/2 digit resolution (`5` or `display.DIGITS_5_5`) • Select 6-1/2 digit resolution (`6` or `display.DIGITS_6_5`)
X	SMU channel (`smua`)

Details

This attribute sets the display resolution.

Example

`display.smua.digits = display.DIGITS_5_5`	Select 5-1/2 digit resolution for SMU A.

Also see

Display resolution (on page 3-72)

display.smuX.measure.func

This attribute specifies the type of measurement being displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall	Saved setups	display.MEASURE_DCVOLTS

Usage

```
func = display.smuX.measure.func
display.smuX.measure.func = func
```

func	One of the following values: • 0 or `display.MEASURE_DCAMPS`: Selects current measure function • 1 or `display.MEASURE_DCVOLTS`: Selects volts measure function • 2 or `display.MEASURE_OHMS`: Selects ohms measure function • 3 or `display.MEASURE_WATTS`: Selects power measure function
X	Source-measure unit (SMU) channel

Details

Selects the displayed measurement function: Amperes, volts, ohms, or watts.

Example

```
display.smua.measure.func =
   display.MEASURE_DCAMPS
```

Selects the current measure function for SMU A.

Also see

None

display.trigger.clear()

This function clears the 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's 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 7-73)
display.trigger.wait() (on page 7-73)

display.trigger.EVENT_ID

This constant is the event ID of the event generated when the front-panel TRIG key is pressed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = display.trigger.EVENT_ID
```

eventID	The trigger event number

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to front panel trigger 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	Not saved	false

Usage

```
overrun = display.trigger.overrun
```

overrun	The trigger overrun state

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 7-72)

display.trigger.wait()

This function waits for the TRIG key on the front panel to be pressed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = display.trigger.wait(timeout)
```

triggered	`true`: Trigger was detected `false`: The operation timed out
timeout	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 5 seconds, the output is `true`. If not, the output is `false`.

Also see

display.trigger.clear() (on page 7-72)

display.waitkey()

This function captures the key code value for the next 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 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 front-panel action, the script is not aborted.

The table below lists the *keyCode* values for each front panel action.

Key codes

Value	Key (or action)	Value	Key (or action)
0	display.KEY_NONE	82	display.KEY_ENTER
65	display.KEY_RANGEUP	85	display.KEY_RECALL
68	display.KEY_MENU	86	display.KEY_MEASA
69	display.KEY_MODEA	86	display.KEY_DIGITSA
70	display.KEY_RELA	88	display.KEY_OUTPUTA
71	display.KEY_RUN	92	display.KEY_TRIG
72	display.KEY_DISPLAY	93	display.KEY_LIMITA
73	display.KEY_AUTO	94	display.KEY_SPEEDA
75	display.KEY_EXIT	95	display.KEY_LOAD
77	display.KEY_FILTERA	97	display.WHEEL_ENTER
78	display.KEY_STORE	103	display.KEY_RIGHT
79	display.KEY_SRCA	104	display.KEY_LEFT
80	display.KEY_CONFIG	107	display.WHEEL_LEFT
81	display.KEY_RANGEDOWN	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 will allow 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 3-81)
display.getlastkey() (on page 3-82, on page 7-56)
display.sendkey() (on page 7-68)
display.settext() (on page 7-70)

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 the Reading error messages and Status model topics for additional information on the error queue.

Also see

errorqueue.count (on page 7-76)
errorqueue.next() (on page 7-77)
Reading errors (on page 8-2)
Status model (on page E-1)

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

errorqueue.clear() (on page 7-76)
errorqueue.next() (on page 7-77)

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 functions reads the oldest entry and removes it from the queue.

Error codes and messages are listed in the Error and status messages (on page 2-80) topics.

If there are no entries in the queue, code 0, "Queue is Empty" is returned

Returned severity levels are described in the following table.

Severity level descriptions

Number	Level	Description
0	**Informational**	Indicates that there are no entries in the queue.
10	**Informational**	Indicates a status message or minor error.
20	**Recoverable**	Indicates possible invalid user input; operation continues but action should be taken to correct the error.
30	**Serious**	Indicates a serious error that may require technical assistance, such as corrupted data.
40	**Fatal**	Indicates that the Model 2651A is nonoperational and requires service. Contact information for service is provided at the front of this manual. Examples: “Bad SMU AFPGA image size,” “SMU is unresponsive,” and “Communication Timeout with DFPGA.”

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

errorqueue.clear() (on page 7-76)
errorqueue.count (on page 7-76)
Error and status messages (on page 2-80)
Status model (on page E-1)

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()
```

logString	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())
```

Output:

```
17:26:35.690 10 Oct 2007, LAN0, 192.168.1.102, LXI, 0, 1192037132,
   1192037155.733269000, 0, 0x0
17:26:39.009 10 Oct 2007, LAN5, 192.168.1.102, LXI, 0, 1192037133,
   1192037159.052777000, 0, 0x0
```

Also see

eventlog.clear() (on page 7-78)
eventlog.count (on page 7-79)
eventlog.enable (on page 7-80)
eventlog.next() (on page 7-80)
eventlog.overwritemethod (on page 7-81)

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 function erases any messages contained in the event log.

Also see

eventlog.all() (on page 7-78)
eventlog.count (on page 7-79)
eventlog.enable (on page 7-80)
eventlog.next() (on page 7-80)
eventlog.overwritemethod (on page 7-81)

eventlog.count

This attribute gets the number of events contained 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 contained the Model 2651A event log. Output looks similar to: `3.00000e00`

Also see

eventlog.all() (on page 7-78)
eventlog.clear() (on page 7-78)
eventlog.enable (on page 7-80)
eventlog.next() (on page 7-80)
eventlog.overwritemethod (on page 7-81)

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	Not saved	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 Model 2651A event log.

Output:

```
1.00000e00
0.00000e00
```

Also see

eventlog.all() (on page 7-78)
eventlog.clear() (on page 7-78)
eventlog.count (on page 7-79)
eventlog.next() (on page 7-80)
eventlog.overwritemethod (on page 7-81)

eventlog.next()

This function returns the oldest message from the event log and removes it from the 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())
```

Output:

```
17:28:22.085 10 Oct 2009, LAN2, 192.168.1.102, LXI, 0, 1192037134, <no time>, 0,
    0x0
```

Example 2

```
print(eventlog.next())
```

Output:

```
nil
```

Also see

eventlog.all() (on page 7-78)
eventlog.clear() (on page 7-78)
eventlog.count (on page 7-79)
eventlog.enable (on page 7-80)
eventlog.overwritemethod (on page 7-81)

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	Not saved	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 be 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 will ignore new entries.

Also see

eventlog.all() (on page 7-78)
eventlog.clear() (on page 7-78)
eventlog.count (on page 7-79)
eventlog.enable (on page 7-80)
eventlog.next() (on page 7-80)

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 7-374)

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()
```

fileVar	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.

Also see

fileVar:flush() (on page 7-83)
fileVar:read() (on page 7-83)
fileVar:seek() (on page 7-84)
fileVar:write() (on page 7-85)
io.close() (on page 7-99)
io.open() (on page 7-100)

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()
```

fileVar	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 this function to flush this data. Using this function removes the need to close a file after writing to it, allowing it 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.

Also see

fileVar:write() (on page 7-85)
io.open() (on page 7-100)
io.write() (on page 7-103)

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)
```

data1	First data read from the file
data2	Second data read from the file.
datan	Last data read from the file
fileVar	The descriptor of the file to be read
format1	A string or number indicating the first type of data to be read
format2	A string or number indicating the second type of data to be read
formatn	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 will perform 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.

Also see

fileVar:write() (on page 7-85)
io.open() (on page 7-100)

fileVar:seek()

This function sets and gets a file's current position.

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)
```

position	The new file position, measured in bytes from the beginning of the file
errorMsg	A string containing the error message
fileVar	The file descriptor variable
whence	A string indicating the base against which `offset` is applied; the *whence* attribute is optional (default is `"cur"`)
offset	The intended new position, measured in bytes from a base indicated by *whence*; (optional, 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.

Also see

io.open() (on page 7-100)
Reading errors (on page 8-2)

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)
```

fileVar	The file descriptor variable
data	Write all data to the file
data1	The first data to write to the file
data2	The second data to write to the file
datan	The last data to write to the file

Details

This function may buffer data until a flush (*fileVar*`:flush()` or `io.flush()`) or close (*fileVar*`:close()` or `io.close()`) operation is performed.

Also see

fileVar:close() (on page 7-82)
fileVar:flush() (on page 7-83)
io.close() (on page 7-99)
io.open() (on page 7-100)

format.asciiprecision

This attribute sets the precision (number of digits) for all numbers printed with the ASCII format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset	Not saved	6

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

precision	A number representing the number of digits to be printed for numbers printed with the `print()`, `printbuffer()`, and `printnumber()` functions; must be a number 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 between 1 and 16.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x)
```

Output:

```
2.540000000e+00

2.54e+00
```

Also see

format.byteorder (on page 7-87)
format.data (on page 7-87)
print() (on page 7-144)
printbuffer() (on page 7-145)
printnumber() (on page 7-146)

format.byteorder

This attribute sets the binary byte order for data printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset	Not saved	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 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)
```

Output depends on the terminal program you use, but will look something like:

```
#0¤p??
#0??p¤
```

Also see

format.asciiprecision (on page 7-86)
format.data (on page 7-87)
printbuffer() (on page 7-145)
printnumber() (on page 7-146)

format.data

This attribute sets the data format for data printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset	Not saved	format.ASCII

Usage

```
value = format.data
format.data = value
```

value	The format to use for data, set to one of the following values: • ASCII format: `1` or `format.ASCII` • Single-precision IEEE Std 754 binary format: `2`, `format.SREAL`, or `format.REAL32` • Double-precision IEEE Std 754 binary format: `3`, `format.REAL`, `format.REAL64`, or `format.DREAL`

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 each for single-precision values and eight bytes each 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:

```
3.141592650e+00
#0ñÔÈSû!      @
```

Also see

format.asciiprecision (on page 7-86)
format.byteorder (on page 7-87)
printbuffer() (on page 7-145)
printnumber() (on page 7-146)

fs.chdir()

This function sets the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
workingDirectory = fs.chdir(path)
```

workingDirectory	Returned value containing the working path
path	A string indicating the new working directory path

Details

The new working directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist.

Example

```
testPath = fs.chdir("/usb1/")
```
Change the working directory to `usb1`.

Also see

None

fs.cwd()

This function returns the absolute path of the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.cwd()
```

path	The absolute path of the current working directory

Also see

None

fs.is_dir()

This function tests whether or not the specified path refers to a directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_dir(path)
```

status	Whether or not the given path is a directory (`true` or `false`)
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

An error is logged to the error queue if the given path does not exist.

Also see

fs.is_file() (on page 7-90)

fs.is_file()

Tests whether the specified path refers to a file (as opposed to a directory).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_file(path)
```

status	`true` if the given path is a file; otherwise, `false`
path	The path of the file system entry to test; this path may be absolute or relative to the current working directory

Details

An error is logged to the error queue if the given path does not exist.

Also see

fs.is_dir() (on page 7-89)

fs.mkdir()

This function creates a directory at the specified path.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.mkdir(newPath)
```

path	The returned path of the new directory
newpath	Location (path) of where to create the new directory

Details

The directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

Also see

fs.rmdir() (on page 7-91)

fs.readdir()

This function returns a list of the file system entries in the directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
files = fs.readdir(path)
```

files	A table containing the names of all the file system entries in the specified directory
path	The directory path

Details

The directory path may be absolute or relative to the current working directory.

This command is nonrecursive. For example, entries in subfolders are not returned.

An error is logged to the error queue if the given path does not exist or does not represent a directory.

Also see

None

fs.rmdir()

This function removes a directory from the file system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
fs.rmdir(path)
```

path	The path of the directory to remove

Details

This path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist, or does not represent a directory, or if the directory is not empty.

Also see

fs.mkdir() (on page 7-90)

gettimezone()

This function retrieves the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timeZone = gettimezone()
```

timeZone	The local timezone of the instrument

Details

See `settimezone()` for additional details on the time zone format and a description of the fields. *timeZone* can be in either of the following formats:

- If one argument was used with `settimezone()`, the format used is:
  `GMThh:mm:ss`
- If four arguments 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 timezone.

Also see

settimezone() (on page 7-168)

gm_isweep()

This KIParlib factory script (on page 5-24) function performs a linear current sweep and calculates the transconductance (Gm) 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)
```

gm_array	A Lua table containing the calculated Gm values at each point
vbuf	A reading buffer containing the measured voltage at each point
ibuf	A reading buffer containing the measured current at each point
smu	System SourceMeter® instrument channel (set to `smua`)
start_i	Starting current level of the sweep
stop_i	Ending current level of the sweep
points	Number of measurements between *start_i* and *stop_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 (Gm) at each point using the central difference method. It can return an array of Gm values, a reading buffer with the measured voltages, and a reading buffer with the measured currents.

Example

`gm_array = gm_isweep(smua, 0, 0.01, 20)`	Source-measure unit (SMU) A returns Gm values only.
`gm_array, vbuf = gm_isweep(smua, 0, 0.01, 20)`	SMU A returns Gm and reading buffer with measured voltages.
`gm_array, vbuf, ibuf = gm_isweep(smua, 0, 0.01, 20)`	SMU A returns Gm and reading buffers with measured voltages and currents.

Also see

gm_vsweep() (on page 7-93)

gm_vsweep()

This KIParlib factory script (on page 5-24) function performs a linear voltage sweep and calculates the transconductance (Gm) 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*)

gm_array	A Lua table containing the calculated Gm values at each point
ibuf	A reading buffer containing the measured current at each point
vbuf	A reading buffer containing the measured voltage at each point
smu	System SourceMeter® instrument channel (set to `smua`)
start_v	Starting voltage level of the sweep
stop_v	Ending voltage level of the sweep
points	Number of measurements between *start_v* and *stop_v* (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 (Gm) at each point using the central difference method. It can return an array of Gm values, a reading buffer with the measured currents, and a reading buffer with the measured voltages.

Example

`gm_array = gm_vsweep(smua, 0, 5, 20)`	SMU A returns Gm values only.
`gm_array, ibuf = gm_vsweep(smua, 0, 5, 20)`	SMU A returns Gm and reading buffer with measured currents.
`gm_array, ibuf, vbuf = gm_vsweep(smua, 0, 5, 20)`	SMU A returns Gm and reading buffers with measured currents and voltages.

Also see

gm_isweep() (on page 7-92)

gpib.address

This attribute contains the GPIB address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	16

Usage

```
address = gpib.address
gpib.address = address
```

address	GPIB address (0 to 30)

Details

A new GPIB address takes affect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow ample time for the command to be processed before attempting to communicate with the instrument again. After sending this command, make sure to use the new address to communicate with the instrument.

The GPIB address is stored in nonvolatile memory. The reset function has no affect on the address.

Example

```
gpib.address = 26
address = gpib.address
print(address)
```

Sets the GPIB address and reads the address.

Output:

```
2.600000e+01
```

Also see

GPIB (see "GPIB operation" on page 2-76)

i_leakage_measure()

This KIHighC factory script (on page 5-24) 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(smu, levelv, limiti, sourcedelay, measurei, measuredelay)
```

imeas	The measured current
smu	System SourceMeter® instrument channel (set to `smua`)
levelv	Voltage level to step to when this function is called
limiti	Current limit setting for the voltage step
sourcedelay	Delay to wait before lowering the current limit for measurement
measurei	Current limit (and measure range); note the current limit is lower at this level and because high-capacitance mode is active, the measure range will follow
measuredelay	Delay to wait after lowering the current limit before taking the measurement

Details

This function is intended to be used when high-capacitance mode is active.

When the instrument is in high-capacitance mode, 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, a measurement is taken and returned as *imeas*.

When measuring leakage current:

- Charge the capacitor before calling this function (the instrument's output is usually at a non-zero voltage prior to 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, .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 will depend on the value and type of capacitor used.

Also see

i_leakage_threshold() (on page 7-96)
High-capacitance mode (on page 3-65)

i_leakage_threshold()

This KIHighC factory script (on page 5-24) function measures the current and compares it to a threshold 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(smu, levelv, limiti, sourcedelay, measurei, measuredelay,
    threshold, timeout)
```

`f`	A Boolean flag; this flag will be `true` when the current is below the threshold, `false` if threshold is not reached before timeout expires
`smu`	System SourceMeter® instrument channel (set to `smua`)
`levelv`	Voltage level to step to when this function is called
`limiti`	Current limit setting for the voltage step
`sourcedelay`	Delay to wait before lowering the current limit for measurement
`measurei`	Current limit (and measure range); note the current limit is lower at this level and because high-capacitance mode is active, the measure range will follow
`measuredelay`	Delay before the first measurement after measure range is changed
`threshold`	The specified current that establishes the test limit
`timeout`	Amount of time (in seconds) to wait for the current to drop to *`threshold`* after all the delays have occurred

Details

This function is intended to be used when high-capacitance mode is active.

When the instrument is in high-capacitance mode, 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 taken at a rate determined by the smu*X*.measure.nplc setting

The function returns `true` if threshold is reached; the function returns `false` if threshold is not reached before *timeout* expires.

When testing the leakage current threshold:

- Charge the capacitor before calling this function (the instrument's output is usually at a non-zero voltage prior to calling this function; when measuring leakage, this function does not charge the capacitor).
- If testing the device's leakage current threshold, 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 will 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

i_leakage_measure() (on page 7-95)
High-capacitance mode (on page 3-65)

InitiatePulseTest()

This KIPulse factory script (on page 5-23) 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)
```

f	A Boolean flag; this flag will be `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* will contain an error message; if it is `true`, `msg` will contain a string indicating successful configuration
tag	Numeric identifier of the pulse configuration to be initiated

Details

This function will only initiate 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 System SourceMeter® instrument channel A to generate a pulse train. If no errors are encountered, initiate the pulse train. Channel A will pulse voltage from a bias level of 0 V to a pulse level of 5 V. The pulse level will be present for 2 ms and the bias level for 200 ms, with a 1 A limit setting. A total of 10 pulses will be generated, and the measurement data will be stored in `smua.nvbuffer1`. This pulse train will be assigned to `tag = 1`.

Also see

ConfigPulseIMeasureV() (on page 7-25)
ConfigPulseVMeasureI() (on page 7-31)
ConfigPulseIMeasureVSweepLin() (on page 7-27)
ConfigPulseVMeasureISweepLin() (on page 7-34)
ConfigPulseIMeasureVSweepLog() (on page 7-29)
ConfigPulseVMeasureISweepLog() (on page 7-36)

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)
```

file	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.

Also see

io.open() (on page 7-100)

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 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 file open and not close it), flush the file after writing to it.

Also see

fileVar:flush() (on page 7-83)
fileVar:write() (on page 7-85)
io.write() (on page 7-103)

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)
```

fileVar	The descriptor of the input file or an error message (if the function fails)
newfile	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 will be returned.

Also see

io.open() (on page 7-100)
io.output() (on page 7-101)

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)
```

fileVar	The descriptor of the opened file
errorMsg	Indicates whether an error was encountered while processing the function
path	The path of the file to open
mode	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 an error is encountered, the command returns `nil` and an error string.

Example

```
testFile, testError = io.open("testfile.txt","w")
if testError == nil then
   testFile:write("This is my test file")
end
io.close(testFile)
```

Also see

io.close() (on page 7-99)

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)
```

fileVar	The descriptor of the output file or an error message (if the function fails)
newfile	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 will be returned.

Also see

io.input() (on page 7-100)
io.open() (on page 7-100)

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)
```

data1	The data read from the file
data2	The data read from the file
dataN	The data read from the file; the number of return values matches the number of format values given
format1	A string or number indicating the type of data to be read
format2	A string or number indicating the type of data to be read
formatN	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 current position; returns an empty string if it is at the end of 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 (0); returns `nil` if the current 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.

If no format parameters are provided, the function will perform as if the function was passed the value `"*l"`.

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)
```

type	Indicates whether the object is an open file handle
obj	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.

Also see

io.open() (on page 7-100)

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)
```

data1	The data to be written
data2	The data to be written
dataN	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 drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

Also see

io.flush() (on page 7-99)

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 current configuration settings.

This function initiates a background operation. LAN configuration could be a lengthy operation. Although the function returns immediately, the LAN initialization will continue 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 DHCP or 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	lan.ENABLE

Usage

```
state = lan.autoconnect
lan.autoconnect = state
```

state	LAN link monitoring state: `1` or `lan.ENABLE`: Enables automatic link reconnection and monitoring `0` or `lan.DISABLE`: 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 configuration after the LAN link is established.

Also see

lan.linktimeout (on page 7-112)
lan.restoredefaults() (on page 7-114)

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
```

dnsAddress	DNS server IP address
N	Entry index (1 or 2)

Details

This attribute is an array of DNS (domain name system) 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.

Unused entries will be returned as "0.0.0.0" when read. *dnsAddress* must be a string specifying the DNS server's IP address in dotted decimal notation. To disable an entry, set its value to "0.0.0.0" or the empty string "".

Although only two address may be manually specified here, the instrument will use 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. The IP address obtained from the DHCP server takes priority for all DNS lookups.

Example

`dnsaddress = "164.109.48.173"` `lan.config.dns.address[1] = dnsaddress`	Write a DNS address of 164.109.48.173 as address 1. Set the DNS address to `dnsaddress`

Also see

lan.config.dns.domain (on page 7-105)
lan.config.dns.dynamic (on page 7-106)
lan.config.dns.hostname (on page 7-107)
lan.config.dns.verify (on page 7-107)
lan.restoredefaults() (on page 7-114)

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
```

domain	Dynamic DNS registration domain; use a string of 255 characters or fewer

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 would be: `Matrix`

Also see

lan.config.dns.dynamic (on page 7-106)
lan.config.dns.hostname (on page 7-107)
lan.config.dns.verify (on page 7-107)
lan.restoredefaults() (on page 7-114)

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	lan.ENABLE

Usage

```
state = lan.config.dns.dynamic
lan.config.dns.dynamic = state
```

state	The dynamic DNS registration state. It may be one of the following values: `1` or `lan.ENABLE`: Enabled `0` or `lan.DISABLE`: 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 7-107)
lan.restoredefaults() (on page 7-114)

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	LAN restore defaults	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 255 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 format 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-707B-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 255 characters can be entered here, care must be taken to be sure the combined length is also no more than 255 characters.

Example

`print(lan.config.dns.hostname)`	Outputs the present dynamic DNS host name.

Also see

lan.config.dns.dynamic (on page 7-106)
lan.restoredefaults() (on page 7-114)

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	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: `1.00000e+00`

Also see

lan.config.dns.hostname (on page 7-107)
lan.restoredefaults() (on page 7-114)

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	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 current operating state of the LAN.

Also see

lan.restoredefaults() (on page 7-114)

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 gateway's IP address 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 currently in effect. Use the `lan.status.gateway` attribute to determine the current 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: `010.060.008.001`

Also see

lan.status.gateway (on page 7-117)
lan.restoredefaults() (on page 7-114)

lan.config.ipaddress

This attribute 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
```

ipAddress	LAN IP address; must be a string specifying the IP address in dotted decimal notation

Details

This attribute 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 currently in effect. Use the `lan.status.ipaddress` attribute to determine the current operating state of the LAN.

Example

`ipaddress = lan.config.ipaddress`	Retrieves the presently set LAN IP address.

Also see

lan.restoredefaults() (on page 7-114)
lan.status.ipaddress (on page 7-118)

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	lan.AUTO

Usage

```
method = lan.config.method
lan.config.method = method
```

method	The method for configuring LAN settings; it can be one of the following values: `0` or `lan.AUTO`: Selects automatic sequencing of configuration methods `1` or `lan.MANUAL`: 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 current method. For example: `1.00000e+00`

Also see

lan.restoredefaults() (on page 7-114)

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

Usage

```
speed = lan.config.speed
lan.config.speed = speed
```

speed	LAN speed setting in Mbps (10 or 100)

Details

This attribute does not indicate the actual setting currently in effect. Use the `lan.status.speed` attribute to determine the current operating state of the LAN.

This attribute stores the speed that will be used if the LAN is restarted for manual configuration operation.

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 7-114)
lan.status.speed (on page 7-121)

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
```

mask	LAN subnet mask value string specifying the subnet mask in dotted decimal notation

Details

This attribute specifies the LAN subnet mask to use 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 currently in effect. Use the `lan.status.subnetmask` attribute to determine the current operating state of the LAN.

Example

`print(lan.config.subnetmask)`	Outputs the LAN subnet mask, such as: `255.255.255.000`

Also see

lan.restoredefaults() (on page 7-114)
lan.status.subnetmask (on page 7-121)

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

Usage

```
timeout = lan.linktimeout
lan.linktimeout = timeout
```

timeout	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 7-104)
lan.restoredefaults() (on page 7-114)

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 will be generated with this domain number. All inbound LXI packets will be ignored unless they have this domain number.

Example

```
print(lan.lxidomain)
```
Displays the LXI domain.

Also see

lan.restoredefaults() (on page 7-114)

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	lan.ENABLE

Usage

```
state = lan.nagle
lan.nagle = state
```

state	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 7-114)

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()`.

Also see

lan.applysettings() (on page 7-103)
lan.restoredefaults() (on page 7-114)

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	""
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.ENABLE
lan.timedwait	20 (seconds)

The lan.restoredefaults() function does not reset the LAN password. The localnode.password attribute controls the web password, which can be reset separately.

This command is run when lan.reset() is sent.

Example

lan.restoredefaults()	Restores the LAN defaults.

Also see

lan.reset() (on page 7-114)
localnode.password (on page 7-134)

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*]

dnsAddress	DNS server IP address
N	Entry index (1, 2, or 3)

Details

This attribute is an array of DNS server addresses. The system 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 here, 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, such as: `164.109.48.173`

Also see

lan.status.dns.name (on page 7-116)

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
```

hostName	Fully qualified DNS host name that can be used to connect to the instrument

Details

A fully qualified domain name (FQDN), sometimes referred to as an absolute domain name, is a domain name that 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 7-104)
lan.config.dns.hostname (on page 7-107)

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

Details

Example

`print(lan.status.duplex)`	Outputs the present LAN duplex mode, such as: `1.00000e+00`

Also see

None

lan.status.gateway

This attribute contains the LAN default gateway address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
gatewayAddress = lan.status.gateway
```

gatewayAddress	LAN default gateway address

Details

The value of *gatewayAddress* is a string that indicates the IP address of the default gateway in dotted decimal notation.

Example

`print(lan.status.gateway)`	Outputs the gateway address, such as: `10.60.8.1`

Also see

lan.config.gateway (on page 7-108)

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
```

ipAddress	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:

```
10.60.8.83
```

Also see

lan.config.ipaddress (on page 7-109)

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
```

macAddress	The instrument MAC address

Details

The MAC address is a character string representing the instrument's MAC address 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:

```
00:60:1A: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
```

port	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

Stores 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 Model 2651A 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 instrument's transmission speed 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
```

mask	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 will return 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 7-111)

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

Usage

```
timeout = lan.timedwait
lan.timedwait = timeout
```

timeout	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.

Also see

lan.restoredefaults() (on page 7-114)

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()
```

N	The LAN event number (1 to 8)

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudostate to the appropriate state.

The following indexes provide the listed events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- …
- 8:LAN7

Example

`lan.trigger[5].assert()`	Creates a trigger with LAN packet 5.

Also see

lan.trigger[N].clear() (on page 7-123)
lan.trigger[N].mode (on page 7-127)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].stimulus (on page 7-130)
lan.trigger[N].wait() (on page 7-131)

lan.trigger[N].clear()

This function clears the event detector for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.trigger[N].clear()
```

N	The LAN event number to clear (1 to 8)

Details

A trigger's event detector remembers if an event has been detected since the last call. This function clears a trigger's event detector and discards the previous 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 7-122)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].stimulus (on page 7-130)
lan.trigger[N].wait() (on page 7-131)

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()
```

N	The LAN event number (1 to 8)

Details

Prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes for this event are 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 7-122)
lan.trigger[N].ipaddress (on page 7-126)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].protocol (on page 7-129)
lan.trigger[N].stimulus (on page 7-130)
lan.trigger[N].wait() (on page 7-131)

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
```

connected	The LAN event connection state: • `true`: Connected • `false`: Not connected
N	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 7-124)
lan.trigger[N].ipaddress (on page 7-126)
lan.trigger[N].protocol (on page 7-129)

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()
```

N	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 7-126)
lan.trigger[N].protocol (on page 7-129)

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
```

N	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	Not saved	"0.0.0.0"

Usage

```
ipAddress = lan.trigger[N].ipaddress
lan.trigger[N].ipaddress = ipAddress
```

ipAddress	The LAN address for this attribute as a string in dotted decimal notation
N	A number specifying 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 7-124)

lan.trigger[N].mode

This attribute sets the trigger operation and detection mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset	Not saved	lan.TRIG_EITHER

Usage

```
mode = lan.trigger[N].mode
lan.trigger[N].mode = mode
```

mode	A number representing the trigger mode (0 to 7); see the **Details** section for more information
N	A number representing the LAN event number (1 to 8)

Details

This attribute controls the mode in which 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` over `lan.TRIG_SYNCHRONOUS` is preferred, as `lan.TRIG_SYNCHRONOUS` is provided for compatibility with 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 5-5)
TSP-Link (on page 5-19)

lan.trigger[N].overrun

This attribute contains the event detector's overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger N clear LAN trigger N reset	Not applicable	Not applicable

Usage

overrun = lan.trigger[*N*].overrun

overrun	The trigger overrun state for the specified LAN packet (`true` or `false`)
N	The LAN event number (1 to 8)

Details

This attribute 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 LAN4 and outputs the value, such as: `false`

Also see

lan.trigger[N].assert() (on page 7-122)
lan.trigger[N].clear() (on page 7-123)
lan.trigger[N].stimulus (on page 7-130)
lan.trigger[N].wait() (on page 7-131)

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	Not saved	lan.TCP

Usage

```
protocol = lan.trigger[N].protocol
lan.trigger[N].protocol = protocol
```

protocol	The protocol to use for the trigger's messages: • 0 or lan.TCP • 1 or lan.UDP • 2 or lan.MULTICAST
N	The LAN event number (1 through 8)

Details

The LAN trigger listens for trigger messages from either protocol 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].ipaddress (on page 7-126)
lan.trigger[N].connect() (on page 7-124)

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	Not saved	1

Usage

```
pseudostate = lan.trigger[N].pseudostate
lan.trigger[N].pseudostate = pseudostate
```

pseudostate	The simulated line state (0 or 1)
N	The LAN event number (1 to 8)

Details

This attribute can be set to initialize the pseudo state to a known value.

Setting this attribute will 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	Not saved	0

Usage

triggerStimulus = lan.trigger[*N*].stimulus
lan.trigger[*N*].stimulus = *triggerStimulus*

triggerStimulus	The LAN event identifier used to trigger the event
N	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set *triggerStimulus* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.SWEEPING_EVENT_ID	Occurs when the SMU transitions from the idle state to the arm layer of the trigger model
smu*X*.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measure action

Trigger event IDs*	
Event ID	**Event description**
`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 `N`
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.
* 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 7-122)
lan.trigger[N].clear() (on page 7-123)
lan.trigger[N].connect() (on page 7-124)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].wait() (on page 7-131)

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)
```

triggered	Trigger detection indication
N	The trigger packet over LAN to wait for (1 to 8)
timeout	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `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 7-122)
lan.trigger[N].clear() (on page 7-123)
lan.trigger[N].overrun (on page 7-128)
lan.trigger[N].stimulus (on page 7-130)

localnode.autolinefreq

This attribute enables or disables automatic 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` or `false`

Details

Set *flag* to one of the following values:

`true`: Enable automatic line frequency detection at start-up.

`false`: Disable automatic line frequency detection at start-up.

When this attribute is set to `true`, the power line frequency is detected automatically the next time the Model 2651A powers up. After the power line frequency is automatically detected at power-up, the `localnode.linefreq` attribute will be set automatically to `50` or `60`.

If the `localnode.linefreq` attribute is explicitly set, `localnode.autolinefreq` will be automatically set to `false`.

Also see

localnode.linefreq (on page 7-133)

localnode.description

This attribute stores a user-defined description 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
```

description	User-defined description of the instrument

Details

This attribute stores a string that contains a description of the instrument. This value appears on instrument's LXI home page.

This attribute's default value contains `Keithley` *MMMM* `#`*SSSSSSSS*, where: *MMMM* is the instrument's four-digit model number, and #*SSSSSSSS* is the instrument's eight-digit serial number. You can change it to a value that makes sense for your system.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].description`.

Example

```
description = "System in Lab 05"
localnode.description = description
```

Set `description` equal to "System in Lab 05".
Set the LXI home page of this instrument to display "System in Lab 05".

Also see

Using the web interface in the Model 2651A User's manual

localnode.linefreq

This attribute contains the line frequency setting used for NPLC calculations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	60

Usage

```
frequency = localnode.linefreq
localnode.linefreq = frequency
```

frequency	An integer representing the instrument's detected or specified line frequency

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, `localnode` should be replaced 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

`frequency = localnode.linefreq`	Reads line frequency setting.

Example 2

`localnode.linefreq = 60`	Sets the line frequency to 60 Hz.

Also see

localnode.autolinefreq (on page 7-132)

localnode.model

This attribute stores the model number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

model = localnode.model

model	The model number of the instrument

Example

`print(localnode.model)`	Outputs the model number.

Also see

localnode.description (on page 7-133)
localnode.revision (on page 7-138)
localnode.serialno (on page 7-138)

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	String containing the remote interface password

Details

This write-only attribute (cannot be read) stores the password set for any remote interface. When password usage is enabled (`localnode.passwordmode`), supply this password to change the configuration, or to control an instrument from a web page or other 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.

The password can be reset by resetting the LAN from the front panel or by assigning an empty string to this attribute.

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

None

localnode.passwordmode

This attribute stores the remote access password enable mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.PASSWORD_WEB

Usage

```
mode = localnode.passwordmode
localnode.passwordmode = mode
```

mode	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

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].passwordmode`.

Example

`mode = localnode.PASSWORD_WEB` `localnode.passwordmode = mode`	Sets value of `mode` to `PASSWORD_WEB`. Allows use of passwords on the web interface only.

Also see

localnode.password (on page 7-134)

localnode.prompts

This attribute sets and reads the local node prompting state (enabled or disabled).

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 state (`0` to disable or `1` to enable)

Details

This attribute controls prompting. When set to `1`, prompts are issued after each command message is processed by the instrument. When set to `0`, prompts are not issued.

The command messages do not generate prompts. The instrument generates prompts in response to command messages.

When the prompting mode is enabled, the instrument generates prompts in response to command messages. There are three prompts that might be generated:

- **TSP>** is the standard prompt. This prompt indicates that everything is normal and the command is done processing.
- **TSP?** is issued if there are entries in the error queue when the prompt is issued. Like the TSP> prompt, it indicates the command is done processing. It does not mean the previous command generated an error, only that there are still errors in the queue when the command was done processing.
- **>>>>** is the continuation prompt. This prompt is used when downloading scripts. When downloading scripts, many command messages must be sent as a unit. The continuation prompt indicates that the instrument is expecting more messages as part of the current command.

NOTE

Test Script Builder requires prompts. It sets the prompting mode behind the scenes. If you disable prompting, using Test Script Builder will cause the instrument to stop responding because it is waiting for the prompt that lets it know that the command is done executing. Do not disable prompting when using Test Script Builder.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].prompts`.

Example

`localnode.prompts = 1`	Enable prompting.

Also see

localnode.showerrors (on page 7-139)
localnode.prompts4882 (on page 7-137)

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

Details

When set to `1`, the IEEE Std 488.2 common commands generate prompts if prompting is enabled with the `localnode.prompts` attribute. If set to `1`, limit the number of `*trg` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When set to `0`, IEEE Std 488.2 common commands will not generate prompts. When using the `*trg` command with a script that executes `trigger.wait()` repeatedly, set `localnode.prompts4882` to `0` to avoid problems associated with the command interface input queue filling.

This attribute resets to the default value each time the instrument power is cycled.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].prompts4882`.

Example

`localnode.prompts4882 = 0`	Disables IEEE Std 488.2 common command prompting.

Also see

localnode.prompts (on page 7-136)

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.

Example

`localnode.reset()`	Resets the local node.

Also see

reset() (on page 7-151)
smuX.reset() (on page 7-209)

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 firmware revision number currently running in the instrument.

Example

`print(localnode.revision)`	Outputs the present revision level. Sample output: `1.0.0`

Also see

localnode.description (on page 7-133)
localnode.model (on page 7-134)
localnode.serialno (on page 7-138)

localnode.serialno

This attribute stores the instrument's serial number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
serialno = localnode.serialno
```

serialno	The serial number of the instrument

Details

This read-only attribute indicates the instrument serial number.

Example

```
display.clear()
display.settext(localnode.serialno)
```

Clears the unit's display.
Places the unit's serial number on the top line of its display.

Also see

localnode.description (on page 7-133)
localnode.model (on page 7-134)
localnode.revision (on page 7-138)

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
```

errorMode	Enables (1) or disables (0) show-errors state

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, `localnode` should be replaced with the node reference, for example `node[5].showerrors`.

Example

```
localnode.showerrors = 1
```

Enables sending of generated errors.

Also see

localnode.prompts (on page 7-136)

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)
```

getter	The return value
table	Read-only table where the attribute is located
attributeName	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 faster 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 7-140)

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)
```

setter	Function that sets the value of the attribute
table	Read-only table where the attribute is located
attributeName	The string name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the *setter* function will execute faster 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 7-140)

meminfo()

This function returns the current 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()`

freeMem	The amount of free dynamically allocated memory available
totalMem	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 currently used.

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)
```

N	The node number of this instrument
scriptCode	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 will not wait for the script to finish execution.

This function cannot be used from the local node. 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.

Example 2

```
node[3].execute("x = 5")
```

Runs script code in string constant (`"x = 5"`) to set x equal to 5 on node 3.

Example 3

```
node[32].execute(TestDut.source)
```

Runs the test script stored in the variable `TestDut` (previously stored on the master node) on node 32.

Also see

tsplink.group (on page 7-347)
TSP advanced features (on page 6-53)

node[N].getglobal()

This function returns the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
value = node[N].getglobal(name)
```

value	The value of the variable
N	The node number of this instrument
name	The global variable name

Details

Use this function to have the remote master node retrieve the value of a global variable from this node's runtime environment.

Do not use this command to retrieve the value of a global variable from the local node (access the global variable directly). This command should only be used from a remote master when controlling this instrument over a TSP-link®.

Example

`print(node[5].getglobal("test_val"))`	Retrieves and outputs the value of the global variable named `test_val` from node 5.

Also see

TSP advanced features (on page 6-53)

node[N].setglobal() (on page 7-143)

node[N].setglobal()

This function sets the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
node[N].setglobal(name, value)
```

N	The node number of this instrument
name	The global variable name to set
value	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-link®.

Example

`node[3].setglobal("x", 5)`	Sets the global variable `x` on Node 3 to the value of 5.

Also see

TSP advanced features (on page 6-53)

node[N].getglobal() (on page 7-142)

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 not actively performing overlapped commands will set their bits immediately. All remaining nodes will set their own bits as they complete their own overlapped commands.

Also see

waitcomplete() (on page 7-374)
Status Model (on page E-1)

print()

This function generates a response message.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments are 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:

```
1.00000e+01
```

Note that your output might be different if you set your ASCII precision setting to a different value.

Example 2

```
x = 10
print(tostring(x))
```

Example of an output response message:

```
10
```

Also see

format.asciiprecision (on page 7-86)

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, buffer1)
printbuffer(startIndex, endIndex, buffer1, buffer2)
printbuffer(startIndex, endIndex, buffer1, ..., bufferN)
```

startIndex	Beginning index of the buffer to print
endIndex	Ending index of the buffer to print
buffer1	First table or reading buffer subtable to print
buffer2	Second table or reading buffer subtable to print
bufferN	The last table or reading buffer subtable to print
...	One or more tables or reading buffer subtables separated with commas

Details

The correct usage of this function for a buffer containing n elements is:

$1 \leq startIndex \leq endIndex \leq n$

Where n refers to the index of the last entry in the tables to be printed.

If $endIndex < startIndex$ or $n < startIndex$, no data is printed. 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 desired 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 will have 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 will be 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.

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

format.asciiprecision (on page 7-86)
format.byteorder (on page 7-87)
format.data (on page 7-87)
print() (on page 7-144)
printnumber() (on page 7-146)

printnumber()

This function prints numbers using the configured format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printnumber(value1)
printnumber(value1, value2)
printnumber(value1, ..., valueN)
```

value1	First value to print in the configured format
value2	Second value to print in the configured format
valueN	Last value to print in the configured format
...	One or more values separated with commas

Details

There are multiple ways to use this function, depending on how many numbers are to be printed.

This function prints the given numbers using the data format specified by `format.data` and `format.asciiprecision`.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x, 2.54321, 3.1)
```

Output:

```
2.540000000e+00
2.54e+00, 2.54e+00, 3.10e+00
```

Also see

format.asciiprecision (on page 7-86)
format.byteorder (on page 7-87)
format.data (on page 7-87)
print() (on page 7-144)
printbuffer() (on page 7-145)

PulseIMeasureV()

This KIPulse factory script (on page 5-23) 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*`)`

smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in amperes
level	Pulse level in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles

Details

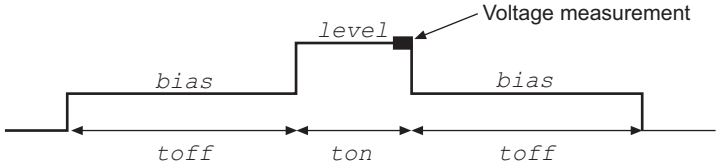
Data for pulsed voltage measurements, current levels, and timestamps are stored in `smu`*X*`.nvbuffer1`.

If any parameters are omitted or `nil`, the operator will be prompted to enter them using the front panel.

Performs a specified number of pulse I, measure V cycles:

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 104: PulseIMeasureV



Example

`PulseIMeasureV(smua, 0.001, 1.0, 20E-3, 40E-3, 10)`	SMU A will output 1 mA and dwell for 40 ms, output 1 A and dwell for 20 ms. The voltage measurement occurs during the 20 ms dwell periods. After the measurement, the output will return to 1 mA and dwell for 40 ms. This pulse-measure process will repeat nine more times.

Also see

None

PulseVMeasureI()

This KIPulse factory script (on page 5-23) 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`*`)`

smu	System SourceMeter® instrument channel (set to `smua`)
bias	Bias level in volts
level	Pulse level in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles

Details

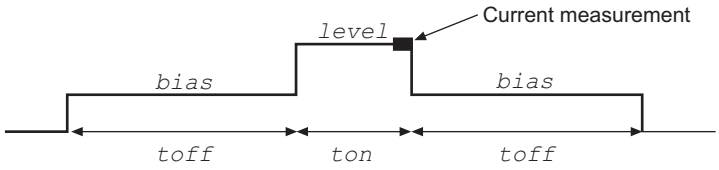
If any parameters are omitted or `nil`, the operator will be prompted to enter them using the front panel.

Data for pulsed current measurements, voltage levels, and timestamps are stored in `smuX.nvbuffer1`.

Performs a specified number of pulse V, measure I cycles:

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 105: PulseVMeasureI()



Example 1

`smua.measure.nplc = 0.001` `PulseVMeasureI(smua, -1, 1, 1E-3, 2E-3, 20)`	SMU A will output -1 V and dwell for 2 ms, output 1 V and dwell for 1 ms. The current measurement occurs during the dwell period. After the measurement, the output will return to -1 V and dwell for 2 ms. This pulse-measure process will repeat 19 more times.

Also see

None

QueryPulseConfig()

This KIPulse factory script (on page 5-23) function allows you to inspect the settings of the pre-configured pulse train assigned to *tag*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

tbl = QueryPulseConfig(*tag*)

tag	Numeric identifier to be assigned to the defined pulse train
tbl	Returned table

Details

Once a pulse train has been configured and assigned to a *tag*, it is often desirable to inspect the settings of this preconfigured pulse train. The `QueryPulseConfig()` command can be used for this purpose.

This function will return a table containing the various settings associated with the *tag* input parameter.

Return values:	
`tostring()`	A function that returns most elements in a string convenient for printing
`tag`	Identifying tag for this pulse train
`smu`	The SMU configured for pulsing
`func`	Pulse function: `smuX.OUTPUT_DCAMPS` or `smuX.OUTPUT_DCVOLTS`
`bias`	Pulse bias level
`level`	Pulse level for non sweeping pulses
`start`	Starting level for sweep pulses
`stop`	Ending level for sweep pulses
`limit`	Limit value
`ton`	On time in seconds
`toff`	Off time in seconds
`points`	The number of points in this pulse train
`buf`	Reference to buffer containing measurement data
`sync_in`	The `sync_in` digio line, if used
`sync_out`	The `sync_out` digio line, if used
`sourcevalues`	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 will pulse voltage from a bias level of 0 V to a pulse level of 5 V. The pulse level will be present for 2 ms, and the bias level for 200 ms with a 1 A limit setting. A total of 10 pulses will be generated, and the measurement data will be stored in `smua.nvbuffer1`. This pulse train will be 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

ConfigPulseIMeasureV() (on page 7-25)
ConfigPulseVMeasureI() (on page 7-31)
ConfigPulseIMeasureVSweepLin() (on page 7-27)
ConfigPulseVMeasureISweepLin() (on page 7-34)
ConfigPulseIMeasureVSweepLog() (on page 7-29)
ConfigPulseVMeasureISweepLog() (on page 7-36)

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	`true`: If the node is the master, the entire system is reset `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. The `localnode.reset()` command is used for the local instrument. The `node[X].reset()` command is used to reset an instrument on a subordinate node.

When no value is specified for *system*, the default value is `true`.

Resetting the entire system using `reset(true)` is permitted only 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 7-137)

savebuffer()

This KISavebuffer factory script (on page 5-24) function saves reading buffers 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)
```

buffer	The reading buffer to save
formatType	A string indicating which file type to use: `.csv` or `.xml`
fileName	The file name of the saved buffer

Details

Use this function to save the specified buffer to a USB flash drive.

This function will only save to a USB flash drive.

You are not required to qualify the path to the USB flash drive, but if you wish to, add `/usb1/` before the *fileName* (see Example 2).

Example 1

`savebuffer(smua.nvbuffer1, "csv", "mybuffer.csv")`	Save `smua` nonvolatile reading buffer 1 as a `.CSV` file named `mybuffer.csv`.

Example 2

`savebuffer(smua.nvbuffer1, "csv", "/usb1/mybuffer.csv")`	Save `smua` nonvolatile reading buffer 1 to an installed USB flash drive as a `.CSV` file named `mybuffer.csv`.

Also see

smuX.savebuffer() (on page 7-209)

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
```

scriptVar	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 `loadandrunscript` 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 6-4)
scriptVar.autorun (on page 7-159)
scriptVar.list() (on page 7-160)
scriptVar.name (on page 7-160)
scriptVar.run() (on page 7-161)
scriptVar.save() (on page 7-162)
scriptVar.source (on page 7-162)

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)
```

scriptName	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 (on page 6-11)
Delete user scripts from the instrument (on page 6-45)
scriptVar.save() (on page 7-162)

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)
```

scriptVar	The created script. This is nil if an error is encountered.
file	The path and file name of the script file to load.
name	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 `loadandrunscript` 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 7-155)

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*)

scriptVar	The name of the variable that will reference the script
code	A string containing the body of the script
name	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 given, an empty string will be used, and the script will be unnamed. If the name already exists in `script.user.scripts`, the existing script's `name` attribute 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 will not be available from the instrument front panel.

You must save the new script into nonvolatile memory to keep it when the instrument is turned off.

Example 1: Create new script

```
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: Create new autoexec script

```
autoexec = script.new(
   "display.clear() display.settext('Hello from autoexec')", 'autoexec')
```

Creates a new 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 6-40)
Global variables and the script.user.scripts table (on page 6-39)
Named scripts (on page 6-4)
scriptVar.save() (on page 7-162)
script.newautorun() (on page 7-157)

script.newautorun()

This function is identical to the `script.new()` function, but it creates a script with the autorun attribute set to `"yes"`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.newautorun(code)
scriptVar = script.newautorun(code, name)
```

scriptVar	The name of the variable that will reference the script
code	A string containing the body of the script
name	The name of the script

Details

The `script.newautorun()` function is identical to the `script.new()` function, except that the autorun attribute of the script is set to `"yes"`.

Also see

Create a script using the script.new() command (on page 6-40)
Global variables and the script.user.scripts table (on page 6-39)
Named scripts (on page 6-4)
scriptVar.save() (on page 7-162)
script.new() (on page 7-155)

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)
```

name	The name of the script to be restored

Details

This command copies the script from nonvolatile memory back into the runtime environment, and it 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 7-154)

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 7-153)

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
```

name	String representing the name of the script
body	Code that implements the body of the `for` loop to process the names in the catalog

Details

Accessing the catalog of scripts stored in nonvolatile memory allows you to process all scripts in nonvolatile memory. The entries will be 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
```

scriptVar	The name of the variable that references the script
state	Whether or not the script runs automatically when powered on: • `"yes"` (script runs automatically) • `"no"` (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 `loadandrunscript` or `script.newautorun()`.

NOTE

Make sure to save the script in nonvolatile memory after setting the `autorun` attribute so that the instrument will retain 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()
```

scriptVar	The name of 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

```
beep.list()
```

Generates output for the content of the script named "`beep`."

Also see

Load a script by sending commands over the remote interface (on page 6-4)
Retrieve source code one line at a time (on page 6-8)

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
```

scriptVar	Name of the variable that references the script
scriptName	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 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 will be removed from the run-time 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 will still reference the script, but the names of the script and variables may not match.

Example: Create a new script with no name and then rename it

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
test7()
print(test7.name)

test7.name = "test7"
print(test7.name)

test7.save()
```

Use `script.new()` to create a script with no name, and then run the script.
Name the script "`test7`", and then save the script in nonvolatile memory.

Also see

script.new() (on page 7-155)
scriptVar.save() (on page 7-162)
Rename a script (on page 6-43)

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

scriptVar`.run()`
scriptVar`()`

scriptVar	The name of variable that references the script

Details

The *scriptVar*`.run()` function runs the script; 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 desired 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)
```

scriptVar	The name of variable that references the script
filename	The file name to use when saving the script to a USB flash drive

Details

The `scriptVar.save()` function only saves a script to nonvolatile memory or a USB flash drive.

You are not required to qualify the path to the USB flash drive, but if you would like to, add `/usb1/` before the *fileName* (see Example 2).

You are not required to add the file extension, but if you would like to, 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 flash drive

Also see

Save a user script (on page 6-10)

scriptVar.source

This attribute holds the source code of a user 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
```

scriptVar	The name of the variable that references the script that contains the source code
code	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 keeps the source for all scripts 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

`print(test1.source)`	Gets source code for a script named `test1`.

Also see

scriptVar.list() (on page 7-160)

serial.baud

This attribute configures the baud rate for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	9600

Usage

```
baud = serial.baud
serial.baud = baud
```

baud	The baud rate (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 or 115200)

Details

A new baud rate setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set the baud rate from one of the other command interfaces or from the front panel.

The reset function has no effect on data bits.

Example

`serial.baud = 1200`	Sets the baud rate to 1200.

Also see

RS-232 interface operation (on page 2-81)
serial.databits (on page 7-164)
serial.flowcontrol (on page 7-164)
serial.parity (on page 7-165)

serial.databits

This attribute configures character width (data bits) for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	8

Usage

```
bits = serial.databits
serial.databits = bits
```

bits	An integer representing the character width (7 or 8)

Details

A new data width setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set the character width from one of the other command interfaces or from the front panel.

The reset function has no effect on data bits.

Example

`serial.databits = 8`	Sets data width to 8.

Also see

RS-232 interface operation (on page 2-81)
serial.baud (on page 7-163)
serial.flowcontrol (on page 7-164)
serial.parity (on page 7-165)

serial.flowcontrol

This attribute configures flow control for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	serial.FLOW_NONE

Usage

```
flow = serial.flowcontrol
serial.flowcontrol = flow
```

flow	A string representing flow control configuration; set to: • "none" or serial.FLOW_NONE (selects no flow control) • "hardware" or serial.FLOW_HARDWARE (selects hardware flow control)

Details

A new flow control setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set the flow control from one of the other command interfaces or from the front panel.

The reset function has no effect on flow control.

Example

serial.flowcontrol = serial.FLOW_NONE	Sets flow control to none.

Also see

serial.baud (on page 7-163)
serial.databits (on page 7-164)
serial.parity (on page 7-165)

serial.parity

This attribute configures parity for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	serial.PARITY_NONE

Usage

```
parity = serial.parity
serial.parity = parity
```

parity	Set parity to one of the following values: • Select no parity ("none" or serial.PARITY_NONE) • Select even parity ("even" or serial.PARITY_EVEN) • Select odd parity ("odd" or serial.PARITY_ODD)

Details

A new parity setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set parity from one of the other command interfaces or from the front panel.

The reset function has no effect on parity.

Example

`serial.parity = serial.PARITY_NONE`	Sets parity to `none`.

Also see

RS-232 interface operation (on page 2-81)
serial.baud (on page 7-163)
serial.databits (on page 7-164)
serial.flowcontrol (on page 7-164)

serial.read()

This function reads available characters (data) from the serial port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

data = serial.read(*maxchars*)

maxchars	An integer that specifies the maximum number of characters to read
data	A string consisting of all data read from the serial port

Details

This function will read available characters from the serial port. It will not wait for new characters to arrive. As long as *maxchars* is less than 200 characters, all characters received by the serial port (before the `serial.read()` command is executed) are returned. If too many characters are received between calls to this function, the RS-232 buffers will overflow and some characters may be lost.

Call this function as many times as necessary to receive the required number of characters. For optimal performance, use a small delay between repeated calls to this function.

The data returned is the raw data stream read from the port. No characters, such as control characters or terminator characters, are interpreted nor will the data stream be altered.

This function cannot be used if the serial port is enabled as a command interface; a settings conflict error will be generated.

Example

`data = serial.read(200)`
 	
`print(data)`
	Read data from the serial port.
	
	Output:
	`John Doe`
	The above output indicates that the string "John Doe" was read from the serial port.

Also see

serial.write() (on page 7-167)

serial.write()

This function writes data to the serial port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
serial.write(data)
```

data	A string representing the data to write

Details

This function writes the specified string to the serial port, where it can be read by connected equipment (for example, a component handler).

No terminator characters are added to the data, and data is written exactly as specified by the `data` parameter.

Example

`serial.write("1 2 3 4")`	Write data string `"1 2 3 4"` to the serial port.

Also see

serial.read() (on page 7-166)

settime()

This function sets the real-time clock (sets current time of the system).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
settime(time)
```

time	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).

To use the Lua `os.time()` function to generate a time value in UTC time, use the following format:

```
os.time({year = year, month = month, day = day, hour = hour, min = min, sec = sec})
```

Where:

year = A full year (2006 or later)
month = The desired month (1 to 12)
day = The desired day (1 to 31)
hour = The desired hour (00 to 23)
min = The desired minute (00 to 59)
sec = The desired second (00 to 59)

When you are using the `os.time()` function, make sure that you include the *year*, *month*, and *day* parameters, which are mandatory (the rest are optional). If the other parameters are not used, they default to noon for that day.

Set the time zone before calling the `os.time()` function.

Example

```
systemTime = os.time({year = 2010,
       month = 3,
       day = 31,
       hour = 14,
       min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2010 at 2:25 pm.

Also see

gettimezone() (on page 7-92)
settimezone() (on page 7-168)

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)
```

offset	String representing offset from UTC
dstOffset	String representing daylight savings offset from UTC
dstStart	String representing when daylight savings time starts
dstEnd	String representing when daylight savings time ends

Details

The time zone is only used when converting between local time and UTC time when using 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 minutes and seconds fields are optional.

The UTC-5 time zone would be specified with the string `"5"` because UTC-5 is 5 hours behind UTC and one must add 5 hours to the local time to get UTC time. The time zone UTC4 would be specified as `"-4"` because UTC4 is 4 hours ahead of UTC and 4 hours must be subtracted from the local time to get 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 within 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")`	Sets `offset` to equal +8 hours, +1 hour for DST, starts on Mar 14 at 2:00 a.m, ends on Nov 7 at 2:00 a.m.
`settimezone(offset)`	Sets local time zone to `offset`.

Also see

gettimezone() (on page 7-92)
settime() (on page 7-167)

setup.poweron

This attribute specifies which saved setup to recall when the unit is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0

Usage

```
n = setup.poweron
setup.poweron = n
```

n	An integer representing the specified setup to recall when system power is turned on (0 to 5)

Details

When *n* = `0`, the instrument uses the factory default setup when turned on; when *n* = `1` to `5`, it uses a user-saved setup.

Example

`setup.poweron = 0`	Set the instrument to use the factory default setup when power is turned on.

Also see

Start-up (power-on) configuration (on page 2-41)

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 specifying 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 unit recalls the factory default setup; when *n* = 1 to 5, the instrument recalls a user-saved setup from nonvolatile memory.

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(2)`	Recall the user-saved setup at location 2.

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

User setup (on page 2-39)

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 instrument's nonvolatile memory.

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 at location 5.

Also see

User setup (on page 2-39)

smuX.abort()

This function terminates all overlapped operations on the specified SMU.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.abort()
```

X	Source-measure unit (SMU) channel (for example, `smua.abort()` specifies SMU channel A).

Details

`smuX.abort()` will not turn the output off or change any settings.

If used to abort a sweep, when executed, the SMU will exit its trigger model immediately and return to the trigger model's idle state.

Example

`smua.abort()`	Terminates all overlapped operations on SMU channel A.

Also see

smuX.measure.overlappedY() (on page 7-201)
smuX.trigger.initiate() (on page 7-234)

smuX.buffer.getstats()

This function returns a specified reading buffer's statistics.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

statistics = smu*X*.buffer.getstats(*bufferVar*)

statistics	The reading buffer's statistics
X	Source-measure unit (SMU) channel (for example, `smua.buffer.getstats()` specifies SMU channel A)
bufferVar	The reading buffer to process

Details

This function returns a table with statistical data about the data placed in the buffer.

The SMU will automatically update 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 will 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 will not be updated. To get fresh statistics, call this function again.

The *statistics* parameter has the following attributes:

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 will be `nil` because there is no data to base any statistics on. If `n` equals 1, the `stddev` attribute will be `nil` because the standard deviation of a sample size of 1 is undefined.

The `min` and `max` entries each have the following attributes:

Attribute	Description
`measurefunction`	String indicating the function measured for the reading (current, voltage, ohms or watts)
`measurerange`	The full-scale range value for the measure 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 *`bufferVar`*`.collectsourcevalues` 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 *`bufferVar`*`.collecttimestamps` 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 will give the actual time the measurement was acquired

Also see

smuX.buffer.recalculatestats() (on page 7-173)

smuX.buffer.recalculatestats()

This function recalculates the specified reading buffer's statistics.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu`*`X`*`.buffer.recalculatestats(`*`bufferVar`*`)`

`X`	Source-measure unit (SMU) channel (for example, `smua.buffer.recalculatestats()` specifies SMU channel A).
`bufferVar`	The reading buffer to process.

Details

This function will cause the SMU to regenerate the reading buffer statistics on the specified reading buffer. As the SMU automatically updates reading buffer statistics as 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 will include the data that was overwritten. Use this function to recalculate the statistics including only the data that is currently stored in the buffer.

Also see

bufferVar.fillmode (on page 7-23)
smuX.buffer.getstats() (on page 7-172)

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
```

adjustDate	Date of the last calibration adjustment.
X	Source-measure unit (SMU) channel (for example, `smua.cal.adjustdate` 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 adjust date without first making a change to the calibration constants.

Once you change any calibration constants, you must set the adjustment date before being allowed to save the calibration data to the SMU's nonvolatile memory.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect 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

`smua.cal.adjustdate = os.time()`	Sets the adjustment date for SMU channel A to the current time set on the instrument.

Also see

smuX.cal.date (on page 7-175)
smuX.cal.due (on page 7-175)
smuX.cal.lock() (on page 7-177)
smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.state (on page 7-180)
smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

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 cal. restore	SMU nonvolatile memory	Initially set to factory calibration date

Usage

```
calDate = smuX.cal.date
smuX.cal.date = calDate
```

calDate	The active calibration set's calibration date.
X	Source-measure unit (SMU) channel (for example, `smua.cal.date` applies to SMU channel A).

Details

This attribute stores the calibration date 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.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect 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

```
smua.cal.date = os.time()
```

Sets calibration date for SMU channel A to the current time set on the instrument.

Also see

smuX.cal.adjustdate (on page 7-174)
smuX.cal.due (on page 7-175)
smuX.cal.lock() (on page 7-177)
smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.state (on page 7-180)
smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

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
```

calDue	Due date of next calibration.
X	Source-measure unit (SMU) channel (for example, `smua.cal.due` 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.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect 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 current time set on the instrument.

Also see

smuX.cal.adjustdate (on page 7-174)
smuX.cal.date (on page 7-175)
smuX.cal.lock() (on page 7-177)
smuX.cal.restore() (on page 7-179)
smuX.cal.state (on page 7-180)
smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

smuX.cal.fastadc()

This function performs calibration of the fast analog-to-digital converter (fast ADC).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.fastadc()
```

X	Source-measure unit (SMU) channel (for example, `smua.cal.fastadc()` specifies SMU channel A).

Details

This function automatically performs all the steps required to calibrate the fast ADC. This function uses the readings obtained when calibrating the measure ranges (make sure both voltage and current calibration has been completed before calling this function).

Example

`smua.cal.fastadc()`	Performs fast ADC calibration for SMU Channel A.

Also see

smuX.measure.calibrateY() (on page 7-193)
Adjustment (on page B-16)

smuX.cal.lock()

This function disables commands that change calibration settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu`*`X`*`.cal.lock()`

X	Source-measure unit (SMU) channel (for example, `smua.cal.lock()` specifies SMU channel A).

Details

This function disables calibration functions that can change calibration settings. The calibration constants must be written to nonvolatile memory, or a previous calibration set must be restored prior to locking calibration. Error number 5012 will result if this function is called when the calibration state is `smu`*`X`*`.CALSTATE_CALIBRATING`.

Example

`smua.cal.lock()`	Disables calibration functions for SMU Channel A.

Also see

smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.state (on page 7-180)
Adjustment (on page B-16)

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

`smu`*`X`*`.cal.password = `*`newPassword`*

X	SMU channel (for example, `smua.cal.password` applies to SMU channel A)
newPassword	The new password (string)

Details

A new password can only be assigned when calibration has been unlocked.

The calibration password is write-only and cannot be read.

Example

`smua.cal.password = "LetMeIn"`	Assigns a new calibration password for SMU Channel A.

Also see

smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

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	Not saved	smuX.CAL_AUTO

Usage

```
calPolarity = smuX.cal.polarity
smuX.cal.polarity = calPolarity
```

calPolarity	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
X	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()` or `smuX.source.calibrateY()` commands. The polarity for those commands are dictated by the range parameter given to the command. The measurement calibration commands require the measurements provided to have been made using the polarity being calibrated.

When making those measurements with calibration points far away from zero, the desired polarity constants are inherently used. When measuring near zero, it is possible for the measurement to be made using the calibration constants from either polarity without knowing which was used. Setting this attribute 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 will automatically be 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

reset() (on page 7-151)
smuX.cal.lock() (on page 7-177)
smuX.cal.unlock() (on page 7-181)
smuX.measure.calibrateY() (on page 7-193)
smuX.reset() (on page 7-209)
smuX.source.calibrateY() (on page 7-211)
Adjustment (on page B-16)

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)
```

X	Source-measure unit (SMU) channel (for example, `smua.cal.restore()` applies to SMU channel A).
calset	The calibration set to be loaded. Set *calset* to one of the following values: `0` or `smuX.CALSET_NOMINAL`: A set of calibration constants that are uncalibrated, but set to nominal values to allow rudimentary functioning of the instrument. `1` or `smuX.CALSET_FACTORY`: The calibration constants when the instrument left the factory. `2` or `smuX.CALSET_DEFAULT`: The normal calibration set. `3` or `smuX.CALSET_PREVIOUS`: The calibration set that was used before the last default set was overwritten.

Details

This function will overwrite the current set of calibration constants with constants read from nonvolatile memory.

This function will be disabled until a successful call to `smuX.cal.unlock()` is made.

If *calset* is not specified, `smuX.CALSET_DEFAULT` will be used.

Example

`smua.cal.restore()`	Restores factory calibration constants for SMU channel A.

Also see

smuX.cal.lock() (on page 7-177)
smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

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, `smua.cal.save()` applies to SMU channel A).

Details

This function will store the active set of calibration constants to nonvolatile memory. The previous calibration constants (from the default calibration set) will be copied to the previous calibration set (`smuX.CALSET_PREVIOUS`) prior to overwriting the default calibration set.

This function will be disabled until a successful call to `smuX.cal.unlock()` is made. If any of the calibration constants have been changed, this function will be disabled unless the calibration date, the calibration due date, and the calibration adjust date have been assigned new values.

Example

`smua.cal.save()`	Stores calibration constants for SMU channel A in nonvolatile memory.

Also see

smuX.cal.adjustdate (on page 7-174)
smuX.cal.date (on page 7-175)
smuX.cal.due (on page 7-175)
smuX.cal.lock() (on page 7-177)
smuX.cal.restore() (on page 7-179)
smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

smuX.cal.state

This attribute stores 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
```

calState	The present calibration state. When reading this read-only attribute, *calState* will have one of the following values: 0 or smu*X*.CALSTATE_LOCKED: Calibration is locked. 1 or smu*X*.CALSTATE_CALIBRATING: The calibration constants or dates have been changed but not yet saved to nonvolatile memory. 2 or smu*X*.CALSTATE_UNLOCKED: Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore.
X	SMU channel (for example, smua.cal.state applies to SMU Channel A).

Details

This read-only attribute indicates the calibration state of the instrument: locked, unlocked, or calibrating.

Example

```
calstate = smua.cal.state
print(calstate)
```

Reads calibration state for SMU Channel A.
Output: 0.000000e+00
The above output indicates that calibration is locked.

Also see

smuX.cal.lock() (on page 7-177)
smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.unlock() (on page 7-181)
Adjustment (on page B-16)

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)
```

X	Source-measure unit (SMU) channel (for example, smua.cal.unlock() applies to SMU channel A)
password	Calibration password

Details

This function enables the calibration functions to change the calibration settings.

The password when the unit is shipped from the factory is "KI0026XX".

Example

`smua.cal.unlock("KI0026XX")`	Unlocks calibration for SMU channel A.

Also see

smuX.cal.lock() (on page 7-177)
smuX.cal.password (on page 7-177)
smuX.cal.state (on page 7-180)
Adjustment (on page B-16)

smuX.contact.calibratehi()

This function calibrates 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)`

X	Source-measure unit (SMU) channel (for example, `smua.contact.calibratehi()` applies to SMU channel A)
cp1Measured	The value measured by this SMU for calibration point 1
cp1Reference	The reference measurement for calibration point 1 as measured externally
cp2Measured	The value measured by this SMU for calibration point 2
cp2Reference	The reference measurement for calibration point 2 as measured externally

Details

Contact check measurement calibration does not require range information.

Typically, calibration points one and two will be 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 will be activated immediately but are not written to nonvolatile storage. Use `smuX.cal.save()` to save the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile storage with the `smuX.cal.restore()` function.

This function will be disabled until a successful call to `smuX.cal.unlock()` is made.

Example

```
-- Short sense LO and output LO terminals
-- Short sense HI and output HI terminals
-- Allow readings to settle, then get measurements
r0_hi, r0_lo = smua.contact.r()

-- Connect 50 OHM resistor between sense LO and output LO
-- Connect 50 OHM resistor between sense HI and output 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)
```

Performs contact check on SMU channel A.

Install and measure two resisters.

Sends contact check LO calibration command.

Send contact check HI calibration command.

Also see

smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.unlock() (on page 7-181)
smuX.contact.calibratelo() (on page 7-183)
Adjustment (on page B-16)

smuX.contact.calibratelo()

This function calibrates 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)`

X	Source-measure unit (SMU) channel (for example, `smua.contact.calibratelo()` applies to SMU channel A)
cp1Measured	The value measured by this SMU for calibration point 1
cp1Reference	The reference measurement for calibration point 1 as measured externally
cp2Measured	The value measured by this SMU for calibration point 2
cp2Reference	The reference measurement for calibration point 2 as measured externally

Details

Contact check measurement calibration does not require range information.

Typically, calibration points one and two will be 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 will be activated immediately but are not written to nonvolatile storage. Use `smuX.cal.save()` to save the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile storage with the `smuX.cal.restore()` function.

This function will be disabled until a successful call to `smuX.cal.unlock()` is made.

Example

`-- Short sense LO and output LO terminals`
`-- Short sense HI and output HI terminals`
`-- Allow readings to settle, then get measurements`
`r0_hi, r0_lo = smua.contact.r()`
	Performs contact check on SMU channel A.

`-- Connect 50 OHM resistor between sense LO and output LO`
`-- Connect 50 OHM resistor between sense HI and output HI`
`-- Allow readings to settle, then get measurements`
`r50_hi, r50_lo = smua.contact.r()`
	Install and measure two resisters.

`smua.contact.calibratelo(r0_lo, Z_actual_lo, r50_lo,` `  50_ohm_actual_lo)`	Sends contact check LO calibration command.
`smua.contact.calibratehi(r0_hi, Z_actual_hi, r50_hi,` `  50_ohm_actual_hi)`	Send contact check HI calibration command.

Also see

smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.unlock() (on page 7-181)
smuX.contact.calibratehi() (on page 7-182)
Adjustment (on page B-16)

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, `smua.contact.check()` 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.

Attempting to perform a contact check measurement when any of the following conditions exist will generate errors:

When output is on and any of the following:

- SMU is a current source with current range set to less than 1 mA (error number 5065)
- SMU is a voltage source with current limit set to less than 1 mA (error number 5050)

When output is off and any of the following:

- The output off mode is High-Z (error number 5048)
- 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 number 5066)
- 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 number 5065)

Example

`if not smua.contact.check() then`
`  -- take action`
`end`
	Takes action if contact check on SMU channel A fails.

Also see

smuX.contact.speed (on page 7-188)
smuX.contact.threshold (on page 7-189)
smuX.source.offfunc (on page 7-218)
Contact check connections (on page 2-45)
Contact check measurements (on page 2-37)

smuX.contact.r()

This function measures contact resistance.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
rhi, rlo = smuX.contact.r()
```

rhi	The measured contact resistance on the high/sense high side
rlo	The measured contact resistance on the low/sense low side
X	Source-measure unit (SMU) channel (for example, `smua.contact.r()` applies to SMU channel A)

Details

Attempting to perform a contact resistance measurement when any of the following conditions exist will generate an error:

When output is on and any of the following:

- SMU is a current source with current range set to less than 1 mA (error number 5065)
- SMU is a voltage source with current limit set to less than 1 mA (error number 5050)

When output is off and any of the following:

- The output off mode is High-Z (error number 5048)
- 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 number 5066)
- 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 number 5065)

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

smuX.contact.check() (on page 7-186)
smuX.contact.speed (on page 7-188)
Contact check connections (on page 2-45)
Contact check measurements (on page 2-37)

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	Not saved	smuX.CONTACT_FAST

Usage

```
speedSetting = smuX.contact.speed
smuX.contact.speed = speedSetting
```

speedSetting	The speed setting. Set to one of the following: 0 or smuX.CONTACT_FAST 1 or smuX.CONTACT_MEDIUM 2 or smuX.CONTACT_SLOW
X	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

reset() (on page 7-151)
smuX.contact.check() (on page 7-186)
smuX.contact.r() (on page 7-187)
smuX.reset() (on page 7-209)
Contact check connections (on page 2-45)
Contact check measurements (on page 2-37)

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	Not saved	50

Usage

```
rValue = smuX.contact.threshold
smuX.contact.threshold = rValue
```

rValue	The resistance above which contact check should fail (measured in ohms).
X	SMU channel (for example, `smua.contact.threshold` applies to SMU Channel A).

Details

The threshold should be set to less than 1 kΩ.

Example

`smua.contact.threshold = 5`	Set the contact check threshold for SMU Channel A to 5 Ω.

Also see

reset() (on page 7-151)
smuX.contact.check() (on page 7-186)
smuX.reset() (on page 7-209)
Contact check connections (on page 2-45)
Contact check measurements (on page 2-37)

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)
```

bufferVar	The created reading buffer.
X	Source-measure unit (SMU) channel (for example, `smua.makebuffer()` applies to SMU channel A).
bufferSize	Maximum number of readings that can be stored.

Details

Reading buffers can be created and allocated dynamically using this function. Use *bufferSize* to designate the number of readings the buffer can store.

Dynamically allocated reading buffers can be used interchangeably with the smu*X*.nvbuffer*Y* buffers.

A reading buffer can be deleted by setting all references to the reading buffer equal to nil, then running the garbage collector (see the collectgarbage() function in Standard libraries (on page 6-30)).

Example

`mybuffer2 = smua.makebuffer(200)`	Creates a 200 element reading buffer (mybuffer2) for SMU channel A .

Also see

collectgarbage()

savebuffer() (on page 7-152)

smuX.nvbufferY (on page 7-208)

Working with reading buffers in the remote state (see "Remote reading buffer programming" on page 3-10)

smuX.measure.adc

This attribute contains the analog to digital converter selection.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	smuX.ADC_INTEGRATE

Usage

```
adc = smuX.measure.adc
smuX.measure.adc = adc
```

adc	The analog to digital (A/D) converter used for measurements. Set to one of the following: 0 or smu*X*.ADC_INTEGRATE: Integrating A/D converter 1 or smu*X*.ADC_FAST: Fast A/D converter
X	Source-measure unit (SMU) channel (for example, smua.measure.adc applies to SMU channel A).

Details

When making measurements, the SMU uses one of two types of analog to digital converters. This attribute controls which analog to converter is used.

Example

`smua.measure.adc = smua.ADC_FAST`	Selects the fast A/D converter for SMU channel A.

Also see

smuX.measure.nplc (on page 7-201)

Analog to digital converter (on page 4-1)

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	Saved setups	smuX.AUTORANGE_ON

Usage

```
autoRange = smuX.measure.autorangeY
smuX.measure.autorangeY = autoRange
```

autoRange	The state of the measurement autorange setting; set to one of the following values: `0` or `smuX.AUTORANGE_OFF`: Disabled `1` or `smuX.AUTORANGE_ON`: Enabled `2` or `smuX.AUTORANGE_FOLLOW_LIMIT`: Measure range automatically set to the limit range.
X	Source-measure unit (SMU) channel (for example, `smua.measure.autorangev` applies to SMU channel A).
Y	SMU measure function (`v` = voltage, `i` = current)

Details

This attribute indicates the measurement autorange state. Its value will be `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 will be the present SMU measure range.

Setting this attribute to `smuX.AUTORANGE_ON` puts the SMU measure circuit into autorange mode. It will remain on its present measure range until the next measurement is requested.

If source highC mode is enabled, current autorange will be set to `smuX.AUTORANGE_FOLLOW_LIMIT` and cannot be changed.

Example

`smua.measure.autorangev = 1`	Enables voltage measurement autoranging for SMU channel A. Alternatively, the value 1 may be replaced with `smua.AUTORANGE_ON`.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.measure.rangeY (on page 7-202)
smuX.reset() (on page 7-209)
Range (on page 2-65)

smuX.measure.autozero

This attribute sets the behavior of the source-measure unit's (SMU's) A/D internal reference measurements (autozero).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	smuX.AUTOZERO_AUTO

Usage

```
azMode = smuX.measure.autozero
smuX.measure.autozero = azMode
```

azMode	Indicates status of autozero; set to one of the following values: 0 or smuX.AUTOZERO_OFF: Autozero disabled 1 or smuX.AUTOZERO_ONCE: Performs autozero once, then disables autozero 2 or smuX.AUTOZERO_AUTO: Automatic checking of reference and zero measurements; an autozero is performed when needed
X	SMU channel (for example, smua.measure.autozero applies to SMU channel A)

Details

The integrating ADC uses a ratio metric A/D conversion technique. To ensure the accuracy of readings, the instrument must periodically obtain fresh 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. Separate reference and zero measurements are used 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 will automatically take 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 additional 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, the smuX.measure.autozero attribute can be used to disable the automatic reference measurements. Keep in mind that disabling automatic reference measurements may allow the instrument to gradually drift out of specification.

To minimize the drift, a reference and zero measurement should be made just prior to the critical test sequence. The smuX.AUTOZERO_ONCE setting can be used to force a refresh of the reference and zero measurements used for the current aperture setting.

Autozero reference measurements for the last 10 used NPLC settings are stored 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 will be discarded to make room for the new entry.

The fast ADC (analog to digital converter) does not need or use the reference measurements associated with this attribute (only the integrating ADC). When this attribute is set to smuX.AUTOZERO_AUTO, the fast ADC will not require these reference measurements. However, if the fast ADC is selected, setting this attribute to smuX.AUTOZERO_ONCE will cause the integrating ADC to acquire a new set of reference measurements.

Example

`smua.measure.autozero = 1`	Performs autozero once for SMU channel A. Alternatively, the value 1 may be replaced with smua.AUTOZERO_ONCE.

Also see

reset() (on page 7-151)
smuX.measure.nplc (on page 7-201)
setup.recall() (on page 7-170)
smuX.reset() (on page 7-209)
Autozero (on page 2-22)

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)
```

X	Source-measure unit (SMU) channel (for example, `smua.measure.calibratev()` applies to SMU channel A).
Y	SMU measurement function (`v` = voltage, `i` = current).
range	The measurement range to calibrate.
cp1Measured	The value measured by this SMU for calibration point 1.
cp1Reference	The reference measurement for calibration point 1 as measured externally.
cp2Measured	The value measured by this SMU for calibration point 2.
cp2Reference	The reference measurement for calibration 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 calibrated separately. Use a positive value for *range* to calibrate the positive polarity and a negative value for *range* to calibrate the negative polarity.

Typically the two calibration points used will be near zero for calibration point 1 and 90% of full scale for calibration point 2.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the active calibration set. Corruption of the calibration constants may result if this is not heeded.

The new calibration constants will be activated immediately but they will not be written to nonvolatile storage. Use `smuX.cal.save()` to commit the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile storage with the `smuX.cal.restore()` function.

This function will be disabled until a successful call to `smuX.cal.unlock()` is made.

Example

`smua.measure.calibratev(1, 1e-4, 1e-5, 0.92, 0.903)`	SMU channel A calibrates voltage measurement using 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, and 0.903 for the +FS DMM measurement reading.

Also see

smuX.cal.lock() (on page 7-177)
smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.unlock() (on page 7-181)
smuX.source.calibrateY() (on page 7-211)
Adjustment (on page B-16)

smuX.measure.count

This attribute sets the number of measurements performed when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	1

Usage

```
count = smuX.measure.count
smuX.measure.count = count
```

count	Number of measurements.
X	SMU channel (for example, `smua.measure.count` applies to SMU Channel A).

Details

This attribute controls the number of measurements taken 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 `smuX.measure.count` is set to a value greater than 1, any measurement delay set by `smuX.measure.delay` will only occur before the first measurement, while the `smuX.measure.interval` controls the interval between successive measurements.

Example

`smua.measure.count = 10`	Sets the SMU Channel A measure count to 10.

Also see

reset() (on page 7-151)
smuX.measure.delay (on page 7-195)
smuX.measure.interval (on page 7-199)
smuX.measure.overlappedY() (on page 7-201)
smuX.measure.Y() (on page 7-205)
smuX.reset() (on page 7-209)
setup.recall() (on page 7-170)

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	Not saved	smuX.DELAY_OFF

Usage

```
mDelay = smuX.measure.delay
smuX.measure.delay = mDelay
```

mDelay	Set to your desired measurement delay value (for example, to specify an additional 10 ms measurement 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
X	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 taking a measurement. You can set *mDelay* to `smuX.DELAY_OFF`, `smuX.DELAY_AUTO`, or to a user-defined value (in seconds). A user-defined value will set the delay used, regardless of range.

The `smuX.DELAY_AUTO` setting also 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 auto delay (on page 2-66)
reset() (on page 7-151)
smuX.measure.count (on page 7-194)
smuX.measure.delayfactor (on page 7-196)
smuX.source.delay (on page 7-213)
smuX.reset() (on page 7-209)

smuX.measure.delayfactor

This attribute stores a multiplier to the delays 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	Not saved	1

Usage

```
delayFactor = smuX.measure.delayfactor
smuX.measure.delayfactor = delayFactor
```

delayFactor	The delay factor multiplier.
X	Source-measure unit (SMU) channel (for example, `smua.measure.delayfactor` 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 auto delay (on page 2-66)
reset() (on page 7-151)
smuX.measure.delay (on page 7-195)
smuX.reset() (on page 7-209)

smuX.measure.filter.count

This attribute sets the number of measured readings required to yield one filtered measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	1

Usage

```
filterCount = smuX.measure.filter.count
smuX.measure.filter.count = filterCount
```

filterCount	The number of readings required for each filtered measurement. Set to a value between 1 and 100.
X	SMU channel (for example, `smua.measure.filter.count` applies to SMU Channel A).

Details

This attribute sets the size of the stack used for filtered measurements.

Example

`smua.measure.filter.count = 10`	Sets the filter count for SMU Channel A to 10.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.measure.filter.enable (on page 7-197)
smuX.measure.filter.type (on page 7-198)
smuX.reset() (on page 7-209)
Filters (on page 3-3)

smuX.measure.filter.enable

This attribute enables or disables filtered measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	smuX.FILTER_OFF

Usage

```
filterState = smuX.measure.filter.enable
smuX.measure.filter.enable = filterState
```

filterState	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
X	SMU channel (for example, `smua.measure.filter.enable` applies to SMU Channel A).

Details

This attribute enables or disables the filter.

Example

`smua.measure.filter.enable = 1`	Enables the filter for SMU Channel A. Alternatively, the value 1 may be replaced with `smua.FILTER_ON`.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.measure.filter.count (on page 7-196)
smuX.measure.filter.type (on page 7-198)
smuX.reset() (on page 7-209)
Filters (on page 3-3)

smuX.measure.filter.type

This attribute sets the type of filter used for measurements when `smuX.measure.filter.enable` is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	smuX.FILTER_REPEAT_AVG

Usage

```
filterType = smuX.measure.filter.type
smuX.measure.filter.type = filterType
```

filterType	The filter type to use when filtering is enabled. Set to one of the following values: `0` or `smuX.FILTER_MOVING_AVG`: Selects the moving average filter `1` or `smuX.FILTER_REPEAT_AVG`: Selects the repeat filter `2` or `smuX.FILTER_MEDIAN`: Selects the median filter
X	SMU channel (for example, `smua.measure.filter.type` applies to SMU Channel A).

Details

There are two averaging filter types and one median filter type available. Both repeating and moving types of averaging filters are available.

For the repeating filter (which is the power-on default), 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 “middle-most” reading 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.type = 2`	Selects the median filter for SMU Channel A. Alternatively, the value 2 may be replaced with `smua.FILTER_MEDIAN`.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.measure.filter.count (on page 7-196)
smuX.measure.filter.enable (on page 7-197)
smuX.reset() (on page 7-209)
Filters (on page 3-3)

smuX.measure.highcrangedelayfactor

This attribute contains a delay multiplier that is used during range changes when High-C mode is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	10

Usage

```
delayFactor = smuX.measure.highcrangedelayfactor
smuX.measure.highcrangedelayfactor = delayFactor
```

delayFactor	The delay factor. Set to a value between 1 and 99.
X	SMU channel (for example, `smua.measure.highcrangedelayfactor` applies to SMU Channel A).

Details

This delay multiplier is only active when High-C mode is active.

Example

`smua.measure.highcrangedelayfactor = 5`	Increases the delay for SMU channel A by a factor if 5.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.reset() (on page 7-209)
smuX.source.highc (on page 7-215)
High-capacitance mode (on page 3-65)

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	Saved setups	0

Usage

```
interval = smuX.measure.interval
smuX.measure.interval = interval
```

interval	The interval value (in seconds). Set to a value between 0 and 1.
X	SMU channel (for example, `smua.measure.interval` 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 will do its best to start each measurement when scheduled. If the SMU cannot keep up with the interval setting, measurements will be made as fast as possible.

If filtered measurements are being made, this 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.

Also see

reset() (on page 7-151)
smuX.measure.count (on page 7-194)
setup.recall() (on page 7-170)
smuX.reset() (on page 7-209)

smuX.measure.lowrangeY

This attribute sets the lowest measure range that will be used during autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	Current: 100e-9 (100 nA) Voltage: 100e-3 (100 mV)

Usage

```
lowRange = smuX.measure.lowrangeY
smuX.measure.lowrangeY = lowRange
```

lowRange	The lowest voltage or current measure range used during autoranging.
X	SMU channel (for example, `smua.measure.lowrangev` applies to SMU Channel A).
Y	SMU measure function (`v` = voltage, `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 will be changed to the *lowRange* range value.

Example

`smua.measure.lowrangev = 1`	Sets voltage low range for SMU channel A to 1 V.

Also see

reset() (on page 7-151)
smuX.measure.autorangeY (on page 7-191)
smuX.reset() (on page 7-209)
setup.recall() (on page 7-170)
Range (on page 2-65)

smuX.measure.nplc

This attribute sets the integration aperture for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	1.0

Usage

```
nplc = smuX.measure.nplc
smuX.measure.nplc = nplc
```

nplc	The integration aperture. Set from 0.001 to 25 (in seconds).
X	SMU channel (for example, `smua.measure.nplc` applies to SMU Channel A).

Details

When making measurements, the SMU uses one of two types of analog to digital converters. This attribute controls the integration aperture for the integrating ADC. This attribute is not used when the fast ADC is selected.

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 0.5/60 seconds.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.measure.adc (on page 7-190)
smuX.reset() (on page 7-209)
Speed (on page 2-71)

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.overlappediv(ibuffer, vbuffer)
```

X	Source-measure unit (SMU) channel (for example, `smua.measure.overlappedv()` applies to SMU channel A).
Y	SMU measurement type (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
rbuffer	A reading buffer object where the reading(s) will be stored.
ibuffer	A reading buffer object where current reading(s) will be stored.
vbuffer	A reading buffer object where voltage reading(s) will be stored.

Details

This function will start a measurement and return immediately. The measurements, as they are performed, are stored in a reading buffer (along with any ancillary information also being acquired). If the instrument is configured to return multiple readings where one is requested, the readings will be available as they are made. Measurements are in the following units of measure: `v` = volts, `i` = amperes, `r` = ohms, `p` = watts.

The `smuX.measure.overlappediv()` function stores current readings in *ibuffer* and voltage readings in *vbuffer*.

This function is an overlapped command. Script execution will continue while the measurement(s) is made in the background. Attempts to access result values that have not yet been generated will cause the script to block and wait for the data to become available. The `waitcomplete()` function can also be used to wait for the measurement(s) to complete before continuing.

If a given reading buffer contains any data, it will be cleared prior to taking 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

smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
waitcomplete() (on page 7-374)
Reading buffers (on page 3-6)

smuX.measure.rangeY

This attribute contains the positive full-scale value of the measure range for voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall	Saved setups	Current: 100e-9 (100 nA) Voltage: 100e-3 (100 mV)

Usage

```
rangeValue = smuX.measure.rangeY
smuX.measure.rangeY = rangeValue
```

rangeValue	Set to the maximum expected voltage or current to be measured.
X	Source-measure unit (SMU) channel (for example, `smua.measure.rangev` applies to SMU channel A).
Y	SMU measurement function (`v` = voltage, `i` = current).

Details

Reading this attribute returns the positive full-scale value of the measure range 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 required by the automatic range selection performed by a measuring instrument. Because selecting a fixed range will prevent autoranging, an over-range condition can occur. For example, measuring 3.0 V on the Model 2651A 1 V range will cause 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 will be used.

Model 2651A example: Assume the source function is voltage. The source range is 1 V and you set the measure range for 10 V. Since the source range is 1 V, the SMU will perform voltage measurements on the 1 V range. If you now change the source function to current, voltage measurements will be performed on the 10 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 will not update the hardware settings, but querying will return the range setting that will be used once the output is turned on. Setting a range while the output is on will take effect immediately.

With measure autoranging enabled, the range will be changed only when a measurement is taken. Querying the range after a measurement will return the range selected for that measurement.

Example

`smua.measure.rangev = 0.5`	Selects the 1 V measurement range for SMU Channel A.

Also see

reset() (on page 7-151)
smuX.measure.autorangeY (on page 7-191)
smuX.source.rangeY (on page 7-223)
smuX.reset() (on page 7-209)
setup.recall() (on page 7-170)
Range (on page 2-65)

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	Not saved	smuX.REL_OFF

Usage

```
relEnable = smuX.measure.rel.enableY
smuX.measure.rel.enableY = relEnable
```

relEnable	Relative measurement control. Set *relEnable* to one of the following values: 0 or smuX.REL_OFF: Disables relative measurements. 1 or smuX.REL_ON: Enables relative measurements.
X	SMU channel (for example, smua.measure.rel.enablev applies to SMU Channel A).
Y	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 will be offset by the relative offset value specified by smuX.measure.rel.levelY. Specifically, each returned measured relative reading will be the result of the following calculation:

Relative reading = Actual measured reading - Relative offset value

Example

`smua.measure.rel.enablev = smua.REL_ON`	Enables relative voltage measurements for SMU Channel A.

Also see

reset() (on page 7-151)
smuX.measure.rel.levelY (on page 7-205)
smuX.reset() (on page 7-209)
setup.recall() (on page 7-170)
Rel (on page 3-1)

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	Not saved	0

Usage

```
relValue = smuX.measure.rel.levelY
smuX.measure.rel.levelY = relValue
```

relValue	Relative measurement offset value.
X	SMU channel (for example, `smua.measure.rel.levelv` applies to SMU Channel A).
Y	SMU measurement function (`v` = voltage, `i` = current, `r` = resistance, `p` = 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 will be offset by the value of this attribute. Specifically, each returned measured relative reading will be 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

reset() (on page 7-151)
smuX.measure.rel.enableY (on page 7-204)
smuX.reset() (on page 7-209)
Rel (on page 3-1)

smuX.measure.Y()

This function performs 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)
```

reading	Returned value of the last (or only) reading of the measurement process
X	Source-measure unit (SMU) channel (for example, `smua.measure.v()` applies to SMU channel A)
Y	SMU measurement function (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
readingBuffer	A reading buffer object where all readings will be stored
iReading	The last reading of the current measurement process
vReading	The last reading of the voltage measurement process.
iReadingBuffer	A reading buffer object where current readings will be stored.
vReadingBuffer	A reading buffer object where voltage readings will be stored.

Details

This function (without specifying a reading buffer) will only make one measurement and return that measurement as *reading*. To use the additional information 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 will be available in *readingBuffer*, but only the last measurement will be 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 will ignore the `smuX.measure.count` attribute and only make one measurement.

The *readingBuffer* will be cleared before taking any measurements unless the buffer is configured to append data.

Example

`smua.measure.count = 10` `smua.measure.v(smua.nvbuffer1)`	Performs ten voltage measurements using SMU channel A and stores them in a buffer.

Also see

smuX.measure.count (on page 7-194)

smuX.measure.overlappedY() (on page 7-201)

smuX.nvbufferY (on page 7-208)

Reading buffer overview (see "Reading buffers" on page 3-6)

smuX.measureYandstep()

This function performs 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)
```

reading	The measured reading before stepping the source
X	Source-measure unit (SMU) channel (for example, `smua.measurevandstep()` applies to SMU channel A)
Y	SMU measurement function (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
sourceValue	Source value to be set after the measurement is made
iReading	The current reading before stepping the source
vReading	The voltage reading before stepping the source

Details

The `smuX.measureYandstep()` function performs a measurement and then sets the source to *sourceValue*. Usage of the `smuX.measureivandstep()` function is similar, but performs 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.

The specified source value should be 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 highC mode is enabled (highC mode requires autoranging to be enabled).

This function is provided for very fast execution of source-measure loops. The measurement will be made prior to stepping the source. Prior to 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.Y() (on page 7-205)
smuX.trigger.source.limitY (on page 7-242)
smuX.trigger.source.linearY() (on page 7-243)
smuX.trigger.source.listY() (on page 7-244)
smuX.trigger.source.logY() (on page 7-245)
Sweep Operation (on page 3-19)

smuX.nvbufferY

This attribute contains the 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
```

bufferVar	The dedicated reading buffer
X	Source-measure unit (SMU) channel (for example, `smua.nvbuffer1` applies to SMU channel A)
Y	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

savebuffer() (on page 7-152)
smuX.makebuffer() (on page 7-189)
smuX.measure.overlappedY() (on page 7-201)
smuX.savebuffer() (on page 7-209)
smuX.trigger.measure.action (on page 7-235)
smuX.trigger.measure.set() (on page 7-236)
smuX.trigger.measure.stimulus (on page 7-237)
smuX.trigger.measure.Y() (on page 7-238)
Configuring and running sweeps (on page 3-29)
Reading buffers (on page 3-6)

smuX.reset()

This function turns off the output and resets the SMU to the 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, `smua.reset()` 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 7-151)

smuX.savebuffer()

This function saves one source-measure unit (SMU) dedicated reading buffer to internal memory (there are two dedicated reading buffers per SMU).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.savebuffer(smuX.nvbufferY)
```

X	SMU channel (for example, `smua.savebuffer(smua.nvbuffer1)` applies to SMU channel A)
Y	SMU dedicated reading buffer (1 or 2)

Details

When the unit is turned off and back on, the dedicated reading buffers will be restored (from internal memory) to their last saved values.

Example

```
smua.savebuffer(smua.nvbuffer1)
```

Saves buffer 1 (SMU channel A) to internal memory.

Also see

savebuffer() (on page 7-152)
smuX.nvbufferY (on page 7-208)
Reading buffers (on page 3-6)

smuX.sense

This attribute contains the state of the remote/local sense mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall	Saved setups	smuX.SENSE_LOCAL

Usage

```
senseMode = smuX.sense
smuX.sense = senseMode
```

senseMode	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.
X	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 (on page 2-43)
4-wire remote sensing (on page 2-44)
Sense mode selection (on page 2-44)

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	Saved setups	smuX.AUTORANGE_ON

Usage

```
sourceAutorange = smuX.source.autorangeY
smuX.source.autorangeY = sourceAutorange
```

sourceAutorange	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
X	Source-measure unit (SMU) channel (for example, `smua.source.autorangev` applies to SMU channel A)
Y	SMU source function (`v` = voltage, `i` = current)

Details

This attribute indicates the source autorange state. Its value will be `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 will be 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 will immediately change range to the range most appropriate for the value being sourced if that range is different from the present SMU range.

Autorange will disable if the source level is edited from the front panel. Setting the source range turns off autorange when set by using the `smuX.source.rangeY` attribute as well.

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 7-191)
smuX.source.rangeY (on page 7-223)

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)
```

X	Source-measure unit (SMU) channel (for example, `smua.source.calibratev()` applies to SMU channel A).
Y	SMU source function (`v` = voltage, `i` = current).
range	The measurement range to calibrate.
cp1expected	The source value programmed for calibration point 1.
cp1reference	The reference measurement for calibration point 1 as measured externally.
cp2expected	The source value programmed for calibration point 2.
cp2reference	The reference measurement for calibration 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 calibrated separately. Use a positive value for range to calibrate the positive polarity and a negative value for range to calibrate the negative polarity. Do not use 0.0 for a negative calibration point as 0.0 is considered a positive number.

Typically the two calibration points used will be near zero for calibration point 1 and 90% of full scale for calibration point 2. Full scale for calibration point 2 should be avoided if the SMU's source 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 will be activated immediately but they will not be written to nonvolatile storage. Use `smuX.cal.save()` to commit the new constants to nonvolatile storage.

The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile storage with the `smuX.cal.restore()` function.

This function will be disabled until a successful call to `smuX.cal.unlock()` is made.

Example

`smua.source.calibratev(1, 1e-10, 1e-5, 0.9, 0.903)`	SMU channel A calibrates voltage source using the following values: calibrate the 1 V range, 1e-10 for +zero source output value, 1e-5 for +zero DMM measurement reading, 0.9 for +FS source output value, and 0.903 for the +FS DMM measurement reading.

Also see

smuX.cal.restore() (on page 7-179)
smuX.cal.save() (on page 7-180)
smuX.cal.unlock() (on page 7-181)
smuX.measure.calibrateY() (on page 7-193)
Calibration (on page B-1)

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
```

compliance	The state of source compliance
X	Source-measure unit (SMU) channel (for example, `smua.source.compliance` applies to SMU channel A)

Details

This read-only attribute contains the source compliance state (`true` or `false`).

- `true` indicates that the limit function is in control of the source (source in compliance).
- `false` indicates that the source function is in control of the output (source not in compliance).

Writing to this attribute will generate an error.

Reading this attribute also updates the status model and the front panel with generated compliance information.

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 7-217)

See Current Limit (ILMT) of Measurement event registers (on page E-5) Voltage Limit (VLMT) not shown, but similar

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	Not saved	smuX.DELAY_OFF

Usage

```
sDelay = smuX.source.delay
smuX.source.delay = sDelay
```

sDelay	Set to your desired source delay value. For example, to specify an additional 10 ms source delay, set the value to `0.010`. Alternatively, you can set it one of the following values: `0` or `smuX.DELAY_OFF`: No delay `-1` or `smuX.DELAY_AUTO`: Automatic delay value
X	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. Set *sDelay* to a user-defined value (in seconds). Alternatively, set *sDelay* to `smuX.DELAY_OFF` or `smuX.DELAY_AUTO`.

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 in the instrument's specifications.

Example

`smua.source.delay = smua.DELAY_AUTO`	Sets the delay for SMU Channel A to automatic (a range dependent delay is inserted when ever the source is changed).

Also see

reset() (on page 7-151)
smuX.measure.count (on page 7-194)
smuX.measure.delay (on page 7-195)
smuX.reset() (on page 7-209)

smuX.source.func

This attribute sets the source function (V source or I source).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall	Saved setups	smuX.OUTPUT_DCVOLTS

Usage

```
sFunction = smuX.source.func
smuX.source.func = sFunction
```

sFunction	The source function. Set to one of the following values: `0` or `smuX.OUTPUT_DCAMPS`: Selects current source function. `1` or `smuX.OUTPUT_DCVOLTS`: Selects voltage source function.
X	SMU channel (for example, `smua.source.func` applies to SMU Channel A).

Details

Reading this attribute gives the output function of the source for the specified SMU Channel. Setting this attribute configures the specified SMU Channel as either a voltage source or a current source.

Example

`smua.source.func = smua.OUTPUT_DCAMPS`	Selects the source current function for SMU Channel A.

Also see

smuX.source.levelY (on page 7-216)
smuX.source.output (on page 7-221)

smuX.source.highc

This attribute sets high capacitance mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall	Saved setups	smuX.DISABLE

Usage

```
highC = smuX.source.highc
smuX.source.highc = highC
```

highC	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.
X	SMU channel (for example, `smua.source.highc` applies to SMU Channel A).

Details

Turning on High-C 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.

Example

`smua.source.highc = smua.ENABLE`	Activates high capacitance mode for SMU Channel A.

Also see

High-capacitance mode (on page 3-65)

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	Saved setups	0

Usage

```
sourceLevel = smuX.source.levelY
smuX.source.levelY = sourceLevel
```

sourceLevel	The source value. Set to one of the following values: Voltage from 0 V to ±40 V. Current from 0 A to ±20 A.
X	SMU channel (for example, `smua.source.levelv` applies to SMU Channel A).
Y	SMU source function (`v` = voltage, `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 will be sourced immediately. If the output is off or the source is configured as a current source, the voltage level will be 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.leveli` setting will be sourced immediately. If the output is off or the source is configured as a voltage source, the current level will be 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

`smua.source.levelv = 1`	Sets voltage source of SMU Channel A to 1 V.

Also see

smuX.source.compliance (on page 7-213)
smuX.source.func (on page 7-214)
smuX.source.output (on page 7-221)
Source-measure concepts (on page 4-2)

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	Saved setups	Limit voltage: 20 (20V) Limit current: 100e-3 (100 mA) Limit power: 0 (disabled)

Usage

```
limit = smuX.source.limitY
smuX.source.limitY = limit
```

limit	The compliance limit value. Set to one of the following values: Set voltage compliance from 0 V to 40 V Set current compliance from 0 A to 20 A Set power compliance in watts.
X	Source-measure unit (SMU) channel (for example, `smua.source.limitv` applies to SMU channel A)
Y	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 will always choose (autorange) the source range for the limit setting. Use the `smuX.source.limitp` attribute to limit the output power of the source. This attribute should be set in the test sequence before the turning the source on.

Using a limit value of 0 will result in a "`Parameter Too Small`" error message (error 1102) 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 will adjust the source limit where appropriate to limit the output to the specified power. The SMU will use the lower of the programmed compliance value (the compliance level that would be used if power compliance were 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 = 30`	Sets voltage source compliance of SMU Channel A to 30 V.

Also see

smuX.source.compliance (on page 7-213)
smuX.source.func (on page 7-214)
smuX.source.output (on page 7-221)
DUT test connections (on page 2-42)

smuX.source.lowrangeY

This attribute sets the lowest source range that will be used during autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall	Saved setups	Voltage: 100e-3 (100 mV) Current: 100e-9 (100 nA)

Usage

```
sourceRangeLow = smuX.source.lowrangeY
smuX.source.lowrangeY = sourceRangeLow
```

sourceRangeLow	Set to the lowest voltage (in volts) or current (in amperes) range to be used
X	Source-measure unit (SMU) channel (for example, `smua.source.lowrangev` applies to SMU channel A)
Y	SMU source function (v = voltage, i = current)

Details

This attribute is used with source autoranging to put a lower bound on the range used. Lower ranges generally require greater settling times. By setting a low-range value, sourcing small values might be able to be made 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 will be changed to the range specified by *sourceRangeLow*.

Example

`smua.source.lowrangev = 1`	Sets volts low range for Model 2651A SMU channel A to 1 V. This prevents the source from using the 100 mV range when sourcing voltage.

Also see

smuX.source.autorangeY (on page 7-211)

smuX.source.offfunc

This attribute sets the source function 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	Saved setups	smuX.OUTPUT_DCVOLTS

Usage

```
offfunc = smuX.source.offfunc
smuX.source.offfunc = offfunc
```

offfunc	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: smu*X*.OUTPUT_DCVOLTS: Source 0 V smu*X*.OUTPUT_DCAMPS: Source 0 A
X	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 the smu*X*.source.offmode is set to smu*X*.OUTPUT_NORMAL. Set this attribute to smu*X*.OUTPUT_DCVOLTS for the source to be a 0 V source when the output is off (smu*X*.source.offlimiti is used). Set it to smu*X*.OUTPUT_DCAMPS for the source to be a 0 A source when the output is off (smu*X*.source.offlimitv is used).

This attribute is only used when the smu*X*.source.offmode attribute is set to smu*X*.OUTPUT_NORMAL.

Example

```
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 modes (see "Output-off states" on page 2-59)

smuX.source.offlimitY (on page 7-219)

smuX.source.offmode (on page 7-220)

smuX.source.output (on page 7-221)

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	Saved setups	Current: 1e-3 (1 mA) Voltage: 40 (40 V)

Usage

```
sourceLimit = smuX.source.offlimitY
smuX.source.offlimitY = sourceLimit
```

sourceLimit	Set to the limit to be used when the SMU is in normal output-off mode
X	SMU channel (for example, smua.source.offlimiti applies to SMU channel A)
Y	SMU source function (v = voltage, 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() (on page 7-186) and smuX.contact.r() (on page 7-187) 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.source.offfunc (on page 7-218)
smuX.source.offmode (on page 7-220)

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	Saved setups	smuX.OUTPUT_NORMAL

Usage

```
sourceOffMode = smuX.source.offmode
smuX.source.offmode = sourceOffMode
```

sourceOffMode	The output-off setting; set to one of the following values: `0` or `smuX.OUTPUT_NORMAL`: Configures the source function according to `smuX.source.offfunc` attribute. `1` or `smuX.OUTPUT_ZERO`: Configures source to output 0 V as `smuX.OUTPUT_NORMAL` but has different compliance handling (see the **Details** below) `2` or `smuX.OUTPUT_HIGH_Z`: Opens the output relay when the output is turned off
X	Source-measure unit (SMU) channel (for example, `smua.source.offmode` applies to SMU channel A)

Details

Reading this attribute gives the output-off mode of the source. Setting this attribute configures the SMU output-off mode.

The default *sourceOffMode* is smu*X*.OUTPUT_NORMAL. In this mode, the source function will be configured according to the smu*X*.source.offfunc attribute. The smu*X*.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 smu*X*.source.offlimitv attribute sets the voltage limit, just as the smu*X*.source.offlimiti attribute sets the current limit when the SMU is operating as a 0 V voltage source.

When the offmode is set to smu*X*.OUTPUT_ZERO, the source will be configured to output 0 V just as smu*X*.OUTPUT_NORMAL mode with smu*X*.source.offfunc = smu*X*.OUTPUT_DCVOLTS. If the source function is voltage, the current limit will not be changed. If the source function is current, the current limit will be set to the current source level or 10 percent of the current source range, whichever is greater.

When offmode is set to smu*X*.OUTPUT_HIGH_Z, the SMU will open the output relay when the output is turned off.

Example

`smua.source.offmode = smua.OUTPUT_HIGH_Z`	Sets the output-off operation for SMU channel A to open the output relay when the output is turned off.

Also see

Output-off modes (see "Output-off states" on page 2-59)

smuX.source.offfunc (on page 7-218)

smuX.source.offlimitY (on page 7-219)

smuX.source.output (on page 7-221)

smuX.source.output

This attribute sets source output state (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall	Not saved	smuX.OUTPUT_OFF

Usage

```
sourceOutput = smuX.source.output
smuX.source.output = sourceOutput
```

sourceOutput	The source's output state setting; set to one of the following values: 0 or smu*X*.OUTPUT_OFF: Turns off the source output 1 or smu*X*.OUTPUT_ON: Turns on the source output 2 or smu*X*.OUTPUT_HIGH_Z: Turns off the output in high Z mode (allows you to go to high Z mode without first setting the smu*X*.source.offmode attribute to smu*X*.OUTPUT_HIGH_Z).
X	Source-measure unit (SMU) channel (for example, smua.source.output applies to SMU channel A)

Details

Reading this attribute gives the output state of the source. Setting this attribute will turn the output of the source on or off. The default for the source is off. When the output is turned on, the SMU will source either voltage or current as dictated by the `smuX.source.func` setting.

Setting this attribute equal to `smuX.OUTPUT_HIGH_Z` will cause the output to turn off and go to the High Z mode. If the `smuX.source.output` is then read it will return `0`.

Example

`smua.source.output = smua.OUTPUT_ON`	Turns on SMU Channel A source output on.

Also see

smuX.source.func (on page 7-214)
smuX.source.offmode (on page 7-220)
DUT test connections (on page 2-42)

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	Saved setups	smuX.OE_NONE

Usage

```
outputAction = smuX.source.outputenableaction
smuX.source.outputenableaction = outputAction
```

outputAction	The source's output enable action. Set to one of the following values: `0` or `smuX.OE_NONE`: No action. `1` or `smuX.OE_OUTPUT_OFF`: Turns the source output off.
X	SMU channel (for example, `smua.source.outputenableaction` applies to SMU Channel A).

Details

This attribute controls the SMU action taken when the output enable line is deasserted.

When set to `smuX.OE_NONE`, the SMU will take no action when the output enable line goes low (deasserted).

When set to `smuX.OE_OUTPUT_OFF` and the output enable line is de-asserted, the SMU will turn its output off as if the `smuX.source.output = smuX.OUTPUT_OFF` command had been received.

The SMU will 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 will turn off immediately.

Detection of the output enable line going low will not abort any running scripts. This may cause execution errors.

Example

`smua.source.outputenableaction = smua.OE_OUTPUT_OFF`	Sets SMU Channel A to turn the output off if the output enable line goes low (deasserted).

Also see

smuX.source.offmode (on page 7-220)
smuX.source.output (on page 7-221)

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	Saved setups	Voltage: 100e-3 (100 mV) Current: 100e-9 (100 nA)

Usage

```
rangeValue = smuX.source.rangeY
smuX.source.rangeY = rangeValue
```

rangeValue	Set to the maximum expected voltage or current to be sourced
X	Source-measure unit (SMU) channel (for example, `smua.measure.rangev` applies to SMU channel A)
Y	SMU source function (`v` = voltage, `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 will select 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 will return the range the unit 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 will prevent autoranging, an overrange condition can occur. For example, sourcing 3.0 V on the Model 2651A 1 V range will cause an overrange condition.

Example

`smua.measure.rangev = 1`	Selects the 1 V source range for SMU channel A.

Also see

reset() (on page 7-151)
setup.recall() (on page 7-170)
smuX.measure.rangeY (on page 7-202)
smuX.reset() (on page 7-209)
smuX.source.autorangeY (on page 7-211)
Range (on page 2-65)

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	Not saved	smuX.SETTLE_SMOOTH

Usage

```
settleOption = smuX.source.settling
smuX.source.settling = settleOption
```

settleOption	Set to the source settling mode. Set to one of the following values: `0` or `smuX.SETTLE_SMOOTH`: Turns off additional settling operations (default). `1` or `smuX.SETTLE_FAST_RANGE`: Instructs the SMU to use a faster procedure when changing ranges. `2` or `smuX.SETTLE_FAST_POLARITY`: Instructs the SMU to change polarity without going to zero. `3` or `smuX.SETTLE_DIRECT_IRANGE`: Instructs the SMU to change the current range directly. `4` or `smuX.SETTLE_SMOOTH_100NA`: Enables the use of range rampers for the 100 nA range. `128` or `smuX.SETTLE_FAST_ALL`: Enables all `smuX.SETTLE_FAST_*` operations.
X	SMU channel (for example, `smua.source.settling` 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 2-65)

smuX.source.sink

This attribute turns sink mode on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall	Saved setups	smuX.DISABLE

Usage

```
sinkMode = smuX.source.sink
smuX.source.sink = sinkMode
```

sinkMode	Set to the source settling mode. Set to one of the following values: `0` or `smuX.ENABLE`: Turns on sink mode. `1` or `smuX.DISABLE`: Turns off sink mode.
X	SMU channel (for example, `smua.source.sink` applies to SMU Channel A).

Details

This attribute enables or disables sink mode. When enabled, sink mode reduces the source limit inaccuracy seen when operating in quadrants II and IV (quadrants I and III will show this source limit inaccuracy).

Example

`smua.source.sink = smua.ENABLE`	Enables sink mode for SMU Channel A.

Also see

Source or sink (on page 4-5)

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	Not saved	1

Usage

```
triggerArmCount = smuX.trigger.arm.count
smuX.trigger.arm.count = triggerArmCount
```

triggerArmCount	The arm count in the trigger model.
X	SMU channel (for example, `smua.trigger.arm.count` applies to SMU Channel A).

Details

During a sweep, the SMU will iterate through the arm layer of the trigger model this many times. After performing this many iterations, the SMU will return to idle.

If this count is set to zero, the SMU will stay 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 7-230)

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()
```

X	Source-measure unit (SMU) channel (for example, `smua.trigger.arm.set()` applies to SMU channel A).

Details

The SMU will automatically clear all the event detectors when the `smuX.trigger.initiate()` function is executed. This function should be called after the sweep is initiated.

A common example of when this function can be used 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 can also be used to start actions on the SMU in case of a missed trigger event.

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 7-234)
smuX.trigger.measure.set() (on page 7-236)
smuX.trigger.source.set() (on page 7-246)

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	Not saved	0

Usage

```
eventID = smuX.trigger.arm.stimulus
smuX.trigger.arm.stimulus = eventID
```

eventID	Event that triggers the arm detector
X	Source-measure unit (SMU) channel (for example, `smua.trigger.arm.stimulus` 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
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the 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 measure 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 *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.

Trigger event IDs*

Event ID	Event description
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.
* 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
```

`trigger.timer[1].EVENT_ID` causes the arm event detector to enter the detected state.

Also see

Triggering (on page 3-32)

smuX.trigger.ARMED_EVENT_ID

This constant contains the armed event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.ARMED_EVENT_ID
```

eventID	The armed event number.
X	SMU channel (for example, `smua.trigger.ARMED_EVENT_ID` applies to SMU Channel A).

Details

Set the stimulus of any trigger event detector to the value of this constant to have it 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 3-32)

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	Not saved	smuX.DISABLE

Usage

```
autoClear = smuX.trigger.autoclear
smuX.trigger.autoclear = autoClear
```

autoClear	Auto clear setting. Set to one of the following values: `0` or `smuX.DISABLE`: Turns off automatic clearing of the event detectors `1` or `smuX.ENABLE`: Turns on automatic clearing of the event detectors.
X	SMU channel (for example, `smua.trigger.autoclear` applies to SMU Channel A).

Details

This attribute is used to enable or disable 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 state of the event detectors will be cleared.

The overrun status 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.

Also see

Triggering (on page 3-32)

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	Not saved	1

Usage

```
triggerCount = smuX.trigger.count
smuX.trigger.count = triggerCount
```

triggerCount	The trigger count is the number of times the SMU will iterate in the trigger layer for any given sweep.
X	SMU channel (for example, `smua.trigger.count` applies to SMU Channel A).

Details

During a sweep, the SMU will iterate through the trigger layer of the trigger model this many times. After performing this many iterations, SMU will return to the arm layer.

If this count is set to zero (`0`), the SMU will stay in the trigger model indefinitely until aborted.

Also see

Triggering (on page 3-32)

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	Not saved	smuX.SOURCE_HOLD

Usage

```
pulseAction = smuX.trigger.endpulse.action
smuX.trigger.endpulse.action = pulseAction
```

pulseAction	The pulse mode setting; set to one of the following values (see details for definition):
	`1` or `smuX.SOURCE_HOLD`
	`0` or `smuX.SOURCE_IDLE`

X	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 sweep, holding the source level for remainder of step. When set to `smuX.SOURCE_IDLE`, this attribute enables pulse mode sweep, 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 end pulse action to achieve a pulse and configures trigger timer 1 to control the end of pulse.

Also see

Triggering (on page 3-32)

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

`smu`*`X`*`.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 will automatically clear all the event detectors when the `smu`*`X`*`.trigger.initiate()` function is executed. This function should be called after the sweep is initiated. Make sure that if the event detectors are configured to automatically clear (because the `smu`*`X`*`.trigger.autoclear` attribute is set to `smu`*`X`*`.ENABLE`) that this command is issued after the SMU has entered the trigger layer.

Also see

smuX.trigger.autoclear (on page 7-229)
smuX.trigger.initiate() (on page 7-234)
Triggering (on page 3-32)

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	Not saved	0

Usage

```
eventID = smuX.trigger.endpulse.stimulus
smuX.trigger.endpulse.stimulus = eventID
```

eventID	Set to the event that triggers the end pulse source off action
X	Source-measure unit (SMU) channel (for example, `smua.trigger.endpulse.stimulus` 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 this attribute's value to `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 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 measure 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 *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.

* 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 end pulse action to achieve a pulse and selects the event `trigger.timer[1].EVENT_ID` that will cause the arm event detector to enter the detected state.

Also see

Triggering (on page 3-32)

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	Not saved	smuX.SOURCE_IDLE

Usage

```
action = smuX.trigger.endsweep.action
smuX.trigger.endsweep.action = action
```

action	The source action at the end of a sweep; set to one of the following values: `0` or `smuX.SOURCE_IDLE`: Sets the source level to the programmed (idle) level at the end of the sweep `1` or `smuX.SOURCE_HOLD`: Set to have the source stay at the level of the last step
X	SMU channel (for example, `smua.trigger.endsweep.action` 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 3-32)

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
```

eventID	The idle event number
X	SMU channel (for example, `smua.trigger.IDLE_EVENT_ID` applies to SMU channel A)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to idle events from this SMU.

Example

`trigger.timer[1].stimulus = smua.trigger.IDLE_EVENT_ID`	Trigger timer when the SMU returns to the idle layer.

Also see

Triggering (on page 3-32)

smuX.trigger.initiate()

This function initiates a sweep operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu`*`X`*`.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 state machine (moves the SMU from the idle state into the arm layer).

To perform source actions during the sweep, it is necessary to configure and enable one of the sweep source actions, `smu`*`X`*`.trigger.source.linear`*`Y`*`()`, `smu`*`X`*`.trigger.source.list`*`Y`*`()`, or `smu`*`X`*`.trigger.source.log`*`Y`*`()`, and measure actions (`smu`*`X`*`.trigger.measure.`*`Y`*`()`, prior to calling this function.

If you run this function more than once without reconfiguring the sweep measurements, the caches on the configured measurement reading buffers will hold stale data. If you do run this function repeatedly without reconfiguring the sweep measurement, 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 7-20)
Configuring and running sweeps (on page 3-29)
smuX.trigger.measure.action (on page 7-235)
smuX.trigger.measure.Y() (on page 7-238)
smuX.trigger.source.action (on page 7-241)
smuX.trigger.source.linearY() (on page 7-243)
smuX.trigger.source.listY() (on page 7-244)
smuX.trigger.source.logY() (on page 7-245)
Triggering (on page 3-32)

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	Not saved	smuX.DISABLE

Usage

```
action = smuX.trigger.measure.action
smuX.trigger.measure.action = action
```

action	The sweep measure action; set to one of the following values: 0 or smuX.DISABLE: Do not make measurements during the sweep 1 or smuX.ENABLE: Make measurements during the sweep 2 or smuX.ASYNC: Make measurements during the sweep, but asynchronously with the source area of the trigger model
X	Source-measure unit (SMU) channel (for example, smua.trigger.measure.action 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`.
- If the integrating ADC is selected (`smuX.measure.adc` attribute is set to `smuX.ADC_INTEGRATE`), autozero must also be turned off. To turn off autozero, set the `smuX.measure.autozero` attribute to `smuX.AUTOZERO_OFF` or `smuX.AUTOZERO_ONCE`.

If either of the above items are incorrectly configured, the `smuX.trigger.initiate()` function will generate an error.

Also see

smuX.measure.adc (on page 7-190)
smuX.trigger.autoclear (on page 7-229)
smuX.trigger.measure.Y() (on page 7-238)
Triggering (on page 3-32)

smuX.trigger.measure.set()

This function sets the measure 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, `smua.trigger.measure.set()` 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.

A common example of when this function can be used 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. Make sure that if the event detectors are configured to automatically clear (because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`) that this command 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 will automatically clear all event detectors when the `smuX.trigger.initiate()` function is executed. This function should be called after the sweep is initiated.

Example

`smua.trigger.measure.set()`	Sets the measure event detector of SMU channel A.

Also see

smuX.trigger.arm.set() (on page 7-226)
smuX.trigger.autoclear (on page 7-229)
smuX.trigger.endpulse.set() (on page 7-231)
smuX.trigger.source.set() (on page 7-246)

smuX.trigger.measure.stimulus

This attribute selects which event will cause 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	Not saved	0

Usage

```
eventID = smuX.trigger.measure.stimulus
smuX.trigger.measure.stimulus = eventID
```

eventID	Event that triggers the measure detector.
X	SMU channel (for example, `smua.trigger.measure.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 will wait for the event at the measure 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
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the 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 measure 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 *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.

Trigger event IDs*

Event ID	Event description
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.
* 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.measure.stimulus = trigger.timer[1].EVENT_ID`	Sets delay before measurement begins on SMU channel A.

Also see

Triggering (on page 3-32)

smuX.trigger.measure.Y()

This function configures the measurements 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)
```

X	Source-measure unit (SMU) channel (for example, `smua.trigger.measure.v()` applies to SMU channel A)
Y	SMU measurement type (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
rbuffer	A reading buffer object where the readings will be stored
ibuffer	A reading buffer object where current readings will be stored
vbuffer	A reading buffer object where voltage readings will be stored

Details

The measurements, as they are performed, are stored in a reading buffer. If the instrument is configured to return multiple readings where one is requested, the readings will be 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 will be cleared prior to taking 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 measure action will be performed.

After configuring the measurements to make with this function, remember to enable the measure action by setting `smuX.trigger.measure.action` = `smuX.ENABLE`.

Example

`smua.trigger.measure.v(vbuffername)` `smua.trigger.measure.action = smua.ENABLE`	Stores voltage readings during the sweep for SMU channel A in buffer `vbuffername`.

Also see

smuX.measure.Y() (on page 7-205)
smuX.nvbufferY (on page 7-208)
smuX.trigger.measure.action (on page 7-235)
waitcomplete() (on page 7-374)
Reading buffers (on page 3-6)
Sweep Operation (on page 3-19)
Triggering (on page 3-32)

smuX.trigger.MEASURE_COMPLETE_EVENT_ID

This constant contains the measure complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

`eventID` = `smuX.trigger.MEASURE_COMPLETE_EVENT_ID`

`eventID`	The measure complete event number.
`X`	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 event detector to the value of this constant to have it respond to measure complete events from this SMU.

Also see

Triggering (on page 3-32)

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
```

eventID	The pulse complete event number
X	SMU channel (for example, `smua.trigger.PULSE_COMPLETE_EVENT_ID` applies to SMU channel A)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to pulse complete events from this SMU.

Also see

Triggering (on page 3-32)

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	Not saved	smuX.DISABLE

Usage

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

action	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.
X	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 7-243)
smuX.trigger.source.listY() (on page 7-244)
smuX.trigger.source.logY() (on page 7-245)
Triggering (on page 3-32)

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	Not saved	smuX.LIMIT_AUTO

Usage

```
sweepSourceLimit = smuX.trigger.source.limitY
smuX.trigger.source.limitY = sweepSourceLimit
```

sweepSourceLimit	Source limit during triggered operation. Set to the desired source limit during triggered operation, `smuX.LIMIT_AUTO`, or `smuX.LIMIT_OFF` (for `smuX.trigger.source.limiti`)
X	SMU channel (for example, `smua.trigger.source.limitv` applies to SMU channel A).
Y	SMU output function (`v` = voltage, `i` = current).

Details

Use this attribute to perform extended operating area (EOA) pulse mode sweeps.

If this attribute is set to `smuX.LIMIT_AUTO` (or `0`), the SMU will use the normal limit setting during sweeping. If this attribute is set to any other numeric value, the SMU will switch in this limit at the start of the source action and will switch back to the normal limit setting at the end of the end pulse action.

Normally, the limit range is automatically adjusted in accordance with the desired limit value. During sweeping, however, the limit range is fixed in order 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`). Note that 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 will be enforced instead. Likewise, if the limit value were to fall below the 10% threshold as a result of power compliance, the 10% value will be 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`").

When using the EOA, the SMU will automatically start the end pulse action if the SMU is not triggered before its maximum pulse width. It will also delay the source action if necessary to limit the pulse duty cycle to stay within the capabilities of the SMU.

Example

`smua.trigger.source.limitv = 10`	Sets the voltage sweep limit of SMU channel A to 10 V.

Also see

smuX.source.limitY (on page 7-217)
Configuring and running sweeps (on page 3-29)
Triggering (on page 3-32)

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)
```

X	Source-measure unit (SMU) channel (for example, `smua.trigger.source.linearv(0, 10, 11)` applies to SMU channel A)
Y	SMU source function (`v` = voltage, `i` = current)
startValue	Source value of the first point
endValue	Source value of the last point
points	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 will generate 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, 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 will restart at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU will satisfy the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *points*, *endValue* will not be reached during the sweep. This means that if the trigger count is less than the number of source values configured, the SMU will satisfy the trigger count and ignore the remaining source values.

In cases where the first sweep point is non-zero, it may be necessary to pre-charge the circuit so that the sweep will return 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 will only store 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.linearv(0, 10, 11)`	Sweeps SMU channel A from 0 V to 10 V in 1 V steps.

Also see

smuX.trigger.source.action (on page 7-241)
smuX.trigger.source.listY() (on page 7-244)
smuX.trigger.source.logY() (on page 7-245)
Sweep Operation (on page 3-19)

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)
```

X	Source-measure unit (SMU) channel (for example, `smua.trigger.source.listv({5})` applies to SMU channel A)
Y	SMU source function (`v` = voltage, `i` = current)
sweepList	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 will output the sequence of source values given in the *sweepList* array.

If the subsequent sweep has more points than specified in *sweepList*, the source will restart at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU will satisfy 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 will be ignored. This means that if the trigger count is less than the number of source values configured, the SMU will satisfy the trigger count and ignore the remaining source values.

In cases where the first sweep point is non-zero, it may be necessary to pre-charge the circuit so that the sweep will return a stable value for the first measured point without penalizing remaining points in the sweep.

The SMU will only store 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.listv({3, 1, 4, 5, 2})`	Sweeps SMU channel A through 3 V, 1 V, 4 V, 5 V, and 2 V.

Also see

smuX.trigger.source.action (on page 7-241)
smuX.trigger.source.linearY() (on page 7-243)
smuX.trigger.source.logY() (on page 7-245)
Sweep Operation (on page 3-19)

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)
```

X	Source-measure unit (SMU) channel (for example, `smua.trigger.source.logv(1, 10, 11, 0)` applies to SMU channel A)
Y	SMU source function (`v` = voltage, `i` = current)
startValue	Source value of the first point
endValue	Source value of the last point
points	The number of points used to calculate the step size
asymptote	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 will generate 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 will restart at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU will satisfy the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *points*, *endValue* will not be reached during the sweep. This means that if the trigger count is less than the number of source values configured, the SMU will satisfy the trigger count and ignore the remaining source values.

In cases where the first sweep point is non-zero, it may be necessary to precharge the circuit so that the sweep will return 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 can be used to customize the inflection and/or 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 would have 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 will only store 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 7-241)
smuX.trigger.source.linearY() (on page 7-243)
smuX.trigger.source.listY() (on page 7-244)
Sweep operation (on page 3-19)

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 will automatically clear all event detectors when the `smuX.trigger.initiate()` function is executed. This function should be called after the sweep is initiated. Make sure that if the event detectors are configured to automatically clear (because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`) that this command is issued after the SMU has entered the trigger layer.

Also see

smuX.trigger.arm.set() (on page 7-226)
smuX.trigger.autoclear (on page 7-229)
smuX.trigger.endpulse.set() (on page 7-231)
smuX.trigger.initiate() (on page 7-234)
smuX.trigger.measure.set() (on page 7-236)
Triggering (on page 3-32)

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	Not saved	0

Usage

```
eventID = smuX.trigger.source.stimulus
smuX.trigger.source.stimulus = eventID
```

eventID	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 will wait for the event at the source event detector portion of the trigger model. To bypass waiting for an event, set this attribute's value to `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 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 measure 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 *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.

Trigger event IDs*	
Event ID	**Event description**
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.
* 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 3-32)

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
```

eventID	The source action complete event number
X	Source-measure unit (SMU) channel (for example, `smua.trigger.SOURCE_COMPLETE_EVENT_ID` applies to SMU channel A)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to source complete events from this SMU.

Also see

Triggering (on page 3-32)

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
```

eventID	The sweep complete event number
X	Source-measure unit (SMU) channel (for example, `smua.trigger.SWEEP_COMPLETE_EVENT_ID` applies to SMU channel A)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to sweep complete events from this SMU.

Also see

Triggering (on page 3-32)

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
```

eventID	The sweeping event number
X	Source-measure unit (SMU) channel (for example, `smua.trigger.SWEEPING_EVENT_ID` applies to SMU channel A)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to sweeping events from this SMU.

Also see

Triggering (on page 3-32)

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
```

statusByte	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
B6	`status.MASTER_SUMMARY_STATUS` `status.MSS`	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 GPIB or VXI-11 serial poll sequence of the 2651A to obtain the status byte (serial poll byte), B6 is the RQS bit. Set bit indicates that the Request Service (RQS) bit of the status byte (serial poll byte) is set and a serial poll (SRQ) has occurred. - When using the `status.condition` register command or the *STB? 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	`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, when more than one bit of the register is set, *`statusByte`* will equal 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: `1.29000e+02`

Converting this output (129) to its binary equivalent yields: 1000 0001

Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSS) and B7 (OSB).

Also see

Status model overview (on page E-4)
Status byte and service request (SRQ) (on page E-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	14,723 (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
```

measurementRegister	The measurement event register's status. 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 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. This attribute is used to read or write to the measurement event registers. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.measurement.VOLTAGE_LIMIT` `status.measurement.VLMT`	Set bit is a summary of the `status.measurement.voltage_limit` register. Bit B0 decimal value: 1
B1	`status.measurement.CURRENT_LIMIT` `status.measurement.ILMT`	Set bit is a summary of the `status.measurement.current_limit` register. Bit B1 decimal value: 2
B2-B6	Not used	Not applicable
B7	`status.measurement.READING_OVERFLOW` `status.measurement.ROF`	Set bit is a summary of the `status.measurement.reading_overflow` register. Bit B7 decimal value: 128
B8	`status.measurement.BUFFER_AVAILABLE` `status.measurement.BAV`	Set bit is a summary of the `status.measurement.buffer_available` register. Bit B8 decimal value: 256
B9-B10	Not used	Not applicable
B11	`status.measurement.OUTPUT_ENABLE` `status.measurement.OE`	Set bit indicates that output enable has been asserted. Bit B11 decimal value: 2048
B12	Not used	Not applicable
B13	`status.measurement.INSTRUMENT_SUMMARY` `status.measurement.INST`	Set bit indicates that a bit in the measurement instrument summary register is set. Bit B13 decimal value: 8192
B14-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	1024	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 E-23)

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	2 (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
```

measurementRegister	The measurement event register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

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 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.buffer_available.SMUA`	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-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 constant, *`measurementRegister`* can be set to the decimal equivalent of the bit to set.

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 E-23)

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	2 (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
```

`measurementRegister`	The measurement event current limit summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

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 information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.current_limit.SMUA`	Set bit indicates that the current limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-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 constant, *`measurementRegister`* can be set to the decimal equivalent of the bit to set.

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 E-23)

status.measurement.instrument.smuX.* (on page 7-257)

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	2 (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
```

measurementRegister	The measurement event instrument summary register status. 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 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.instrument.SMUA`	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-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 constant, *`measurementRegister`* can be set to the decimal equivalent of the bit to set.

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 E-23)

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
```

measurementRegister	The instrument measurement status SMU X summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
X	Source-measure unit (SMU) channel (for example `status.measurement.instrument.smua.enable` 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0*	`status.measurement.instrument.smuX.VOLTAGE_LIMIT` `status.measurement.instrument.smuX.VLMT`	Set bit indicates that the voltage limit was exceeded. Bit B0 decimal value: 1
B1*	`status.measurement.instrument.smuX.CURRENT_LIMIT` `status.measurement.instrument.smuX.ILMT`	Set bit indicates that the current limit was exceeded. Bit B1 decimal value: 2
B2-B6	Not used	Not applicable.
B7	`status.measurement.instrument.smuX.READING_OVERFLOW` `status.measurement.instrument.smuX.ROF`	Set bit indicates that an overflow reading has been detected. Bit B7 decimal value: 128
B8	`status.measurement.instrument.smuX.BUFFER_AVAILABLE` `status.measurement.instrument.smuX.BAV`	Set bit indicates that there is at least one reading stored in either or both of the dedicated reading buffers. Bit B8 decimal value: 256
B9-B15	Not used	Not applicable.
* This bit will be updated only when a measurement is taken 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	1024	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 E-23)

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	2 (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
```

measurementRegister	The measurement reading overflow summary register status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.measurement.reading_overflow.SMUA	Set bit indicates that an overflow reading has been detected. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-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 constant, *measurementRegister* can be set to the numeric equivalent of the bit to set.

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 E-23)

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	2 (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
```

measurementRegister	The measurement voltage limit summary register status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.voltage_limit.SMUA`	Set bit indicates that the voltage limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-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 constant, *`measurementRegister`* can be set to the numeric equivalent of the bit to set.

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 E-23)

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
```

`nodeEnableRegister`	The system node enable register's status. 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 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, assume the value of `1.29000e+02` (which is 129) is returned for the system node enable 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 sets (on page E-2) and Enable and transition registers (on page E-18). 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	Not used	Not applicable.
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
```

Sets the MSB and OSB bits of the system node enable register using constants.

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 7-250)
status.system.* (on page 7-314)
Status model overview (on page E-4)
Status byte and service request (SRQ) (on page E-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 node event register's status. 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	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	Not applicable
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 bit(s) 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.condition (on page 7-250)
status.system.* (on page 7-314)
Status model overview (on page E-4)
Status byte and service request (SRQ) (on page E-14)

status.operation.*

These attributes manage the status model's operation status 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	31,769 (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
```

operationRegister	The operation status register's status. 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 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 B12 and bit B14 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.operation.CALIBRATING` `status.operation.CAL`	Set bit indicates that the summary bit of the `status.operation.calibrating` register has been set. Bit B0 decimal value: 1
B1-B2	Not used	Not applicable
B3	`status.operation.SWEEPING` `status.operation.SWE`	Set bit indicates that the summary bit from the `status.operation.sweeping` register is set. Bit B3 decimal value: 8
B4	`status.operation.MEASURING` `status.operation.MEAS`	Set bit indicates that the summary bit of the `status.operation.measuring` register is set. Bit B4 decimal value: 16
B5-B9	Not used	Not applicable
B10	`status.operation.TRIGGER_OVERRUN` `status.operation.TRGOVR`	Set bit indicates that the summary bit from the `status.operation.trigger_overrun` register is set. Bit B10 decimal value: 1024
B11	`status.operation.REMOTE_SUMMARY` `status.operation.REM`	Set bit indicates that the summary bit of the `status.operation.remote` register is set. Bit B11 decimal value: 2048
B12	`status.operation.USER`	Set bit indicates that the summary bit from the `status.operation.user` register is set. Bit B12 decimal value: 4096
B13	`status.operation.INSTRUMENT_SUMMARY` `status.operation.INST`	Set bit indicates that the summary bit from the `status.operation.instrument` register is set. Bit B13 decimal value: 8192
B14	`status.PROGRAM_RUNNING` `status.PROG`	Set bit indicates that a command or program is running. Bit B14 decimal value: 16,384
B15	Not used	Not applicable

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 4096 + 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	1024	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
```

Sets the USER and PROG bits of the operation status enable register using constants.

Example 2

`-- decimal 20480 = binary 0101 0000 0000 0000`
`operationRegister = 20480`
`status.operation.enable = operationRegister`
	Sets the USER and PROG bits of the operation status enable register using a decimal value.

Also see

Operation Status Registers (on page E-20)

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	2 (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 operation calibrating event register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

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 to B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status calibration summary enable register, set `status.operation.calibrating.enable = status.operation.calibrating.SMUA`.

In addition to the above constant, *operationRegister* can be set to the decimal equivalent of the bit to set.

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 E-20)
status.operation.* (on page 7-266)

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	31,746 (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
```

operationRegister	The operation event register's status. 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 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

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-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: 1024.
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: 2048
B12	`status.operation.instrument.DIGITAL_IO` `status.operation.instrument.DIGIO`	Set bit indicates one or more enabled bits for the operation status digital I/O summary register is set. Bit B12 decimal value: 4096
B13	`status.operation.instrument.TSPLINK`	Set bit indicates one or more enabled bits for the operation status TSP-Link summary register is set. Bit B13 decimal value: 8192
B14	`status.operation.instrument.LAN`	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 + 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	1024	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

status.operation.* (on page 7-266)
Condition register sets of:
• status.operation.instrument.trigger_blender.* (on page 7-283)
• status.operation.instrument.trigger_timer.* (on page 7-286)
• status.operation.instrument.digio.* (on page 7-271)
• status.operation.instrument.tsplink.* (on page 7-289)
• status.operation.instrument.lan.* (on page 7-275)
Operation Status Registers (on page E-20)

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
```

operationRegister	The operation status digital I/O summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0-B9	Not used	Not applicable
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: 1024 Binary value: 0100 0000 0010
B11-B15	Not used	Not applicable

In addition to the above constant, *operationRegister* can be set to the decimal equivalent of the bit to set.

Example

```
status.operation.instrument.digio.enable =
   status.operation.instrument.digio.TRGOVR
```

Sets the TRGOVR bit of the operation status digital I/O summary enable register using a constant.

Also see

Operation Status Registers (on page E-20)

status.operation.instrument.digio.trigger_overrun.* (on page 7-273)

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
```

operationRegister	The operation status digio I/O overrun register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.operation.instrument.digio.trigger_overrun.LINE1`	Bit B1 decimal value: 2
B2	`status.operation.instrument.digio.trigger_overrun.LINE2`	Bit B2 decimal value: 4
B3	`status.operation.instrument.digio.trigger_overrun.LINE3`	Bit B3 decimal value: 8
B4	`status.operation.instrument.digio.trigger_overrun.LINE4`	Bit B4 decimal value: 16
B5	`status.operation.instrument.digio.trigger_overrun.LINE5`	Bit B5 decimal value: 32
B6	`status.operation.instrument.digio.trigger_overrun.LINE6`	Bit B6 decimal value: 64
B7	`status.operation.instrument.digio.trigger_overrun.LINE7`	Bit B7 decimal value: 128
B8	`status.operation.instrument.digio.trigger_overrun.LINE8`	Bit B8 decimal value: 256
B9	`status.operation.instrument.digio.trigger_overrun.LINE9`	Bit B9 decimal value: 512
B10	`status.operation.instrument.digio.trigger_overrun.LINE10`	Bit B10 decimal value: 1024
B11	`status.operation.instrument.digio.trigger_overrun.LINE11`	Bit B11 decimal value: 2048
B12	`status.operation.instrument.digio.trigger_overrun.LINE12`	Bit B12 decimal value: 4096
B13	`status.operation.instrument.digio.trigger_overrun.LINE13`	Bit B13 decimal value: 8192
B14	`status.operation.instrument.digio.trigger_overrun.LINE14`	Bit B14 decimal value: 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 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	1024	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
```

Sets bit B1 and bit B10 of the operation status digital I/O overrun enable register using constants.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.digio.* (on page 7-271)

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 operation status lan summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	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-B9	Not used	Not available
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: 1024
B11-B15	Not used	Not applicable

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	1024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
operationRegister =
   status.operation.instrument.lan.CONF +
   status.operation.instrument.lan.TRGOVR
status.operation.instrument.lan.enable = operationRegister
```

Sets bit B1 and bit B10 of the operation status lan summary enable register using constants.

Also see

Operation Status Registers (on page E-20)

status.operation.instrument.lan.trigger_overrun.* (on page 7-277)

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
```

operationRegister	The operation status lan trigger overrun register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.operation.instrument.lan.trigger_overrun.LAN1`	Bit B1 decimal value: 2
B2	`status.operation.instrument.lan.trigger_overrun.LAN2`	Bit B2 decimal value: 4
B3	`status.operation.instrument.lan.trigger_overrun.LAN3`	Bit B3 decimal value: 8
B4	`status.operation.instrument.lan.trigger_overrun.LAN4`	Bit B4 decimal value: 16
B5	`status.operation.instrument.lan.trigger_overrun.LAN5`	Bit B5 decimal value: 32
B6	`status.operation.instrument.lan.trigger_overrun.LAN6`	Bit B6 decimal value: 64
B7	`status.operation.instrument.lan.trigger_overrun.LAN7`	Bit B7 decimal value: 128
B8	`status.operation.instrument.lan.trigger_overrun.LAN8`	Bit B8 decimal value: 256
B9-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	1024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
operationRegister =
   status.operation.instrument.lan.trigger_overrun.LAN1 +
   status.operation.instrument.lan.trigger_overrun.LAN8
status.operation.instrument.lan.trigger_overrun.enable =
   operationRegister
```

Sets bit B1 and bit B8 of the operation status lan trigger overrun enable register using constants.

Also see

Operation Status Registers (on page E-20)

status.operation.instrument.lan.* (on page 7-275)

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
```

operationRegister	The operation status SMU X summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
X	Source-measure unit (SMU) channel (for example `status.operation.instrument.smua.enable` 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 1025) 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.operation.instrument.smuX.CALIBRATING` `status.operation.instrument.smuX.CAL`	Set bit indicates that `smuX` is unlocked for calibration. Bit B0 decimal value: 1
B1-B2	Not used	Not applicable.
B3	`status.operation.instrument.smuX.SWEEPING` `status.operation.instrument.smuX.SWE`	Set bit indicates that `smuX` is sweeping. Bit B3 decimal value: 8
B4	`status.operation.instrument.smuX.MEASURING` `status.operation.instrument.smuX.MEAS`	Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement. Bit B4 decimal value: 16
B5-B9	Not used	Not applicable.
B10	`status.operation.instrument.smuX.TRIGGER_OVERRUN` `status.operation.instrument.smuX.TRGOVR`	Set bit indicates an enabled bit has been set in the operation status smu X trigger overrun event register. Bit B10 decimal value: 1024
B11-B15	Not used	Not applicable.

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 1025 (which is the sum of 1 + 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	1024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

`status.operation.instrument.smua.enable =` `  status.operation.instrument.smua.MEAS`	Sets the MEAS bit of the operation status SMU A summary enable register using a constant.

Also see

Operation Status Registers (on page E-20)

status.operation.instrument.smuX.trigger_overrrun.* (on page 7-281)

status.operation.instrument.smuX.trigger_overrun.*

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_overrun.condition
operationRegister = status.operation.instrument.smuX.trigger_overrun.enable
operationRegister = status.operation.instrument.smuX.trigger_overrun.event
operationRegister = status.operation.instrument.smuX.trigger_overrun.ntr
operationRegister = status.operation.instrument.smuX.trigger_overrun.ptr
status.operation.instrument.smuX.trigger_overrun.enable = operationRegister
status.operation.instrument.smuX.trigger_overrun.ntr = operationRegister
status.operation.instrument.smuX.trigger_overrun.ptr = operationRegister
```

operationRegister	The operation status SMU X trigger overrun register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.instrument.smuX.trigger_overrun.ARM`	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	`status.operation.instrument.smuX.trigger_overrun.SRC`	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	`status.operation.instrument.smuX.trigger_overrun.MEAS`	Set bit indicates that the measure event detector of the SMU was already in the detected state when a trigger was received. Bit B3 decimal value: 8
B4	`status.operation.instrument.smuX.trigger_overrun.ENDP`	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-B15	Not used	Not applicable.

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	1024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

`status.operation.instrument.smua.trigger_overrun.enable = status.operation.instrument.smua.trigger_overrun.ARM`	Sets the ARM bit of the operation status SMU A trigger overrun enable register using a constant.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.smuX.* (on page 7-279)

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 operation status trigger blender summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0-B9	Not used	Not applicable.
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: 1024 Binary value: 0100 0000 0000
B11-B15	Not used	Not applicable.

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
```

Sets the TRGOVR bit of the operation status trigger blender summary enable using a decimal value.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.trigger_blender.trigger_overrun.* (on page 7-284)

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	30 (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
```

operationRegister	The operation status trigger blender overrun register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.operation.instrument.trigger_blender.trigger_overrun.BLND1`	Bit B1 decimal value: 2
B2	`status.operation.instrument.trigger_blender.trigger_overrun.BLND2`	Bit B2 decimal value: 4
B3	`status.operation.instrument.trigger_blender.trigger_overrun.BLND3`	Bit B3 decimal value: 8
B4	`status.operation.instrument.trigger_blender.trigger_overrun.BLND4`	Bit B4 decimal value: 16
B5-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	1024	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_overrun.enable
   = status.operation.instrument.trigger_blender.trigger_overrun.BLND1
```

Sets the bit for blender 1 of the operation status trigger blender overrun enable register using a constant.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.trigger_blender.* (on page 7-283)

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
```

operationRegister	The operation status trigger timer summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0-B9	Not used	Not applicable
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: 1024 Binary value: 0100 0000 0000
B11-B15	Not used	Not applicable

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
```

Sets the TRGOVR bit of the operation status trigger timer summary enable register using a decimal value.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.trigger_timer.trigger_overrun.* (on page 7-287)

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 operation status trigger timer trigger overrun register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.operation.instrument.trigger_timer.trigger_overrun.TMR1`	Bit B1 decimal value: 2
B2	`status.operation.instrument.trigger_timer.trigger_overrun.TMR2`	Bit B2 decimal value: 4
B3	`status.operation.instrument.trigger_timer.trigger_overrun.TMR3`	Bit B3 decimal value: 8
B4	`status.operation.instrument.trigger_timer.trigger_overrun.TMR4`	Bit B4 decimal value: 16
B5	`status.operation.instrument.trigger_timer.trigger_overrun.TMR5`	Bit B5 decimal value: 32
B6	`status.operation.instrument.trigger_timer.trigger_overrun.TMR6`	Bit B6 decimal value: 64
B7	`status.operation.instrument.trigger_timer.trigger_overrun.TMR7`	Bit B7 decimal value: 128
B8	`status.operation.instrument.trigger_timer.trigger_overrun.TMR8`	Bit B8 decimal value: 256
B9-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	1024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example

```
status.operation.instrument.trigger_timer.trigger_overrun.enable =
   status.operation.instrument.trigger_timer.trigger_overrun.TMR3
```

Sets the Timer 3 bit of the operation status trigger timer overrun enable register using a constant.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.trigger_timer.* (on page 7-286)

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
```

operationRegister	The operation status TSP-Link summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0-B9	Not used	Not applicable
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: 1024 Binary value: 0100 0000 0000
B11-B15	Not used	Not applicable

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`	Sets the trigger overrun bit of the operation status TSP-Link summary enable register using a decimal value.

Also see

Operation Status Registers (on page E-20)

status.operation.instrument.tsplink.trigger_overrun.* (on page 7-290)

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 operation status TSP-link overrun register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.operation.instrument.tsplink.trigger_overrun.LINE1`	Bit B1 decimal value: 2
B2	`status.operation.instrument.tsplink.trigger_overrun.LINE2`	Bit B2 decimal value: 4
B3	`status.operation.instrument.tsplink.trigger_overrun.LINE3`	Bit B3 decimal value: 8
B4-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	1024	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
```

Sets the Line 1 bit of the operation status TSP-Link overrun enable register using a constant.

Also see

Operation Status Registers (on page E-20)
status.operation.instrument.trigger_timer.* (on page 7-286)

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	2 (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
```

operationRegister	The operation status measuring summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.measuring.SMUA`	Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constant, *`operationRegister`* can be set to the decimal equivalent of the bit to set.

Example

`status.operation.measuring.enable = status.operation.measuring.SMUA`	Sets the SMUA bit of the operation status measuring summary enable register using a constant.

Also see

Operation Status Registers (on page E-20)
status.operation.* (on page 7-266)

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 operation status remote summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.remote.COMMAND_AVAILABLE` `status.operation.remote.CAV`	Set bit indicates there is a command available in the execution queue. Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2-B10	Not used	Not applicable.
B11	`status.operation.remote.PROMPTS_ENABLED` `status.operation.remote.PRMPT`	Set bit indicates command prompts are enabled. Bit B11 decimal value: 2048 Binary value: 0000 0100 0000 0000
B12-B15	Not used	Not applicable.

As an example, to set bit B1of 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 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 B11, set *operationRegister* to 2050 (which is the sum of 2 + 2048).

Example

`status.operation.remote.enable =` `  status.operation.remote.CAV`	Sets the CAV bit of the operation status remote summary enable register using a constant.

Also see

Operation Status Registers (on page E-20)

status.operation.* (on page 7-266)

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	2 (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
```

operationRegister	The operation status sweeping summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

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-B5	Not used	Not applicable.

In addition to the above constant, *operationRegister* can be set to the decimal equivalent of the bit to set.

Example

`status.operation.sweeping.enable =` `  status.operation.sweeping.SMUA`	Sets the SMUA bit of the operation status sweeping summary enable register using a constant.

Also see

Operation Status Registers (on page E-20)

status.operation.* (on page 7-266)

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	31,746 (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
```

operationRegister	The operation status trigger overrun summary register's status. 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 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

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.trigger_overrun.SMUA`	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-B9	Not used	Not applicable.
B10	`status.operation.trigger_overrun.TRIGGER_BLENDER` `status.operation.trigger_overrun.TRGBLND`	Set bit indicates one of the enabled bits in the operation status trigger blender overrun event register is set. Bit B10 decimal value: 1024
B11	`status.operation.trigger_overrun.TRIGGER_TIMER` `status.operation.trigger_overrun.TRGTMR`	Set bit indicates one of the enabled bits in the operation status trigger timer overrun event register is set. Bit B11 decimal value: 2048
B12	`status.operation.trigger_overrun.DIGITAL_IO` `status.operation.trigger_overrun.DIGIO`	Set bit indicates one of the enabled bits in the operation status digital I/O overrun event register is set. Bit B12 decimal value: 4096
B13	`status.operation.trigger_overrun.TSPLINK`	Set bit indicates one of the enabled bits in the operation status TSP-Link overrun event register is set. Bit B13 decimal value: 8192
B14	`status.operation.trigger_overrun.LAN`	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_overrun.enable = status.operation.trigger_overrun.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 1026 (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	1024	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
```

Sets bit B1 and bit B10 of the operation status trigger overrun summary enable register using constants.

Also see

Operation Status Registers (on page E-20)
status.operation.* (on page 7-266)

status.operation.user.*

These attributes manage the status model's operation status user register set.

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
```

operationRegister	The operation status user register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.operation.user.BIT0`	Bit B0 decimal value: 1
B1	`status.operation.user.BIT1`	Bit B1 decimal value: 2
B2	`status.operation.user.BIT2`	Bit B2 decimal value: 4
B3	`status.operation.user.BIT3`	Bit B3 decimal value: 8
B4	`status.operation.user.BIT4`	Bit B4 decimal value: 16
B5	`status.operation.user.BIT5`	Bit B5 decimal value: 32
B6	`status.operation.user.BIT6`	Bit B6 decimal value: 64
B7	`status.operation.user.BIT7`	Bit B7 decimal value: 128
B8	`status.operation.user.BIT8`	Bit B8 decimal value: 256
B9	`status.operation.user.BIT9`	Bit B9 decimal value: 512
B10	`status.operation.user.BIT10`	Bit B10 decimal value: 1024
B11	`status.operation.user.BIT11`	Bit B11 decimal value: 2048
B12	`status.operation.user.BIT12`	Bit B12 decimal value: 4096
B13	`status.operation.user.BIT13`	Bit B13 decimal value: 8192
B14	`status.operation.user.BIT14`	Bit B14 decimal value: 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 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 B11 and B14, set *`operationRegister`* to 18,432 (which is the sum of 2048 + 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	1024	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`	Sets bits B11 and B14 of the operation status user enable register using constants.

Example 2

`-- 18432 = binary 0100 1000 0000 0000`
`operationRegister = 18432`
`status.operation.enable = operationRegister`
	Sets bits B11 and B14 of the operation status user enable register using a decimal value.

Also see

status.operation.* (on page 7-266)
Operation Status Register (on page E-21)

status.questionable.*

These attributes manage the status model's questionable status 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	13,056 (All bits set)

Usage

```
quesRegister = status.questionable.condition
quesRegister = status.questionable.enable
quesRegister = status.questionable.event
quesRegister = status.questionable.ntr
quesRegister = status.questionable.ptr
status.questionable.enable = quesRegister
status.questionable.ntr = quesRegister
status.questionable.ptr = quesRegister
```

quesRegister	The questionable status register's status. 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 `1.02400e+04` (which is 10,240) is read as the value of the condition register, the binary equivalent is 0010 1000 0000 0000. This value indicates that bits B11 and B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	1	0	1	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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0-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-B11	Not used	Not available
B12	`status.questionable.OVER_TEMPERATURE` `status.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-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, *`quesRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`quesRegister`* to the sum of their decimal weights. For example, to set bits B12 and B13, set *`quesRegister`* to 12,288 (which is the sum of 4096 + 8192).

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	1024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Also see

Questionable Status Registers (on page E-21)

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	2 (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
```

questionableRegister	The questionable status calibration summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.questionable.calibration.SMUA`	Set bit indicates that the 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-B15	Not used	Not applicable.

In addition to the above constant, *questionableRegister* can be set to the decimal equivalent of the bit to set.

Example

```
status.questionable.calibration.enable =
   status.questionable.calibration.SMUA
```

Sets the SMUA bit of the questionable status calibration summary enable register using a constant.

Also see

Questionable Status Registers (on page E-21)

status.questionable.* (on page 7-300)

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	2 (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
```

questionableRegister	The questionable status instrument summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.questionable.instrument.SMUA`	Set bit indicates one or more enabled bits in the questionable status SMU A summary event register is set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set.

Example

```
status.questionable.instrument.enable =
   status.questionable.instrument.SMUA
```

Sets the SMUA bit of the questionable status instrument summary enable register using a constant.

Also see

Questionable Status Registers (on page E-21)
status.questionable.* (on page 7-300)

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
```

questionableRegister	The questionable status SMU X summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
X	Source-measure unit (SMU) channel (for example `status.questionable.instrument.smua.enable` 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0-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-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: 4096
B13-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	1024	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
```

Sets bit B8 and bit B9 of the questionable status SMU A summary enable register using constants.

Also see

Questionable Status Registers (on page E-21)

status.operation.* (on page 7-266)

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	2 (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
```

operationRegister	The questionable status over temperature summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.questionable.over_temperature.SMUA`	Set bit indicates that an over temperature indication was detected. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set.

Example

```
status.questionable.over_temperature.enable =
   status.questionable.over_temperature.SMUA
```

Sets the SMU A bit in the questionable status over temperature summary enable register using a constant.

Also see

Questionable Status Registers (on page E-21)
status.questionable.* (on page 7-300)

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	2 (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
```

operationRegister	The questionable status unstable output summary register's status. A zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.questionable.unstable_output.SMUA`	Set bit indicates that an unstable output condition was detected. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set.

Example

```
status.questionable.unstable_output.enable =
   status.questionable.unstable_output.SMUA
```

Sets the SMU A bit in the questionable status unstable output summary enable register bit using a constant.

Also see

Questionable Status Registers (on page E-21)

status.questionable.* (on page 7-300)

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
```

`requestSRQEnableRegister`	The service request (SRQ) enable register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). 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 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
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

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
```

Sets the MSB and OSB bits of the service request (SRQ) enable register using constants.

Example 2

`-- decimal 129 = binary 10000001`
`requestSRQEnableRegister = 129`
`status.request_enable = requestSRQEnableRegister`
	Sets the MSB and OSB bits of the service request (SRQ) enable register using a decimal value.

Also see

status.condition (on page 7-250)
status.system.* (on page 7-314)
Status model overview (on page E-4)
Status byte and service request (SRQ) (on page E-14)

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
```

requestSRQEventRegister	The request event register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). 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 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
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 bit(s) 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 7-250)

status.system.* (on page 7-314)

Status model overview (on page E-4)

Status byte and service request (SRQ) (on page E-14)

status.reset()

This function resets all bits in the system 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 sets (on page E-2) and Enable and transition registers (on page E-18).

Example

`status.reset()`	Resets the instrument status model.

Also see

Status Model (on page E-1)

status.standard.*

These attributes manage the status model's standard event status 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	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
```

standardRegister	The standard event status register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.standard.OPERATION_COMPLETE` `status.standard.OPC`	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 `*OPC` command. The `opc()` function can be used in place of the `*OPC` command. Bit B0 decimal value: 1
B1	Not used	Not applicable
B2	`status.standard.QUERY_ERROR` `status.standard.QYE`	Set bit indicates that you attempted to read data from an empty Output Queue. Bit B2 decimal value: 4
B3	`status.standard.DEVICE_DEPENDENT_ERROR` `status.standard.DDE`	Set bit indicates that an instrument operation did not execute properly due to some internal condition. Bit B3 decimal value: 8
B4	`status.standard.EXECUTION_ERROR` `status.standard.EXE`	Set bit indicates that the instrument detected an error while trying to execute a command. Bit B4 decimal value: 16
B5	`status.standard.COMMAND_ERROR` `status.standard.CME`	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	`status.standard.USER_REQUEST` `status.standard.URQ`	Set bit indicates that the LOCAL key on the instrument front panel was pressed. Bit B6 decimal value: 64

Bit	Value	Description
B7	`status.standard.POWER_ON` `status.standard.PON`	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-B15	Not used	Not applicable

As an example, to set bit B0 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
```

Sets the OPC and EXE bits of the standard event status enable register using constants.

Example 2

```
-- decimal 17 = binary 0001 0001
standardRegister = 17
status.standard.enable = standardRegister
```

Sets the OPC and EXE bits of the standard event status enable register using a decimal value.

Also see

Standard Event Register (on page E-19)

status.system.*

These attributes manage the status model's TSP-Link® system summary register 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
```

enableRegister	The system summary register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.system.EXTENSION_BIT` `status.system.EXT`	Bit B0 decimal value: 1
B1	`status.system.NODE1`	Bit B1 decimal value: 2
B2	`status.system.NODE2`	Bit B2 decimal value: 4
B3	`status.system.NODE3`	Bit B3 decimal value: 8
B4	`status.system.NODE4`	Bit B4 decimal value: 16
B5	`status.system.NODE5`	Bit B5 decimal value: 32
B6	`status.system.NODE6`	Bit B6 decimal value: 64
B7	`status.system.NODE7`	Bit B7 decimal value: 128
B8	`status.system.NODE8`	Bit B8 decimal value: 256
B9	`status.system.NODE9`	Bit B9 decimal value: 512
B10	`status.system.NODE10`	Bit B10 decimal value: 1024
B11	`status.system.NODE11`	Bit B11 decimal value: 2048
B12	`status.system.NODE12`	Bit B12 decimal value: 4096
B13	`status.system.NODE13`	Bit B13 decimal value: 8192
B14	`status.system.NODE14`	Bit B14 decimal value: 16384
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 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 B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2048 + 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	1024	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
```

Sets bits B11 and B14 of the system summary enable register using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system.enable = enableRegister
```

Sets bits B11 and B14 of the system summary enable register using a decimal value.

Also see

status.system2.* (on page 7-316)
System summary and standard event registers (on page E-6)

status.system2.*

These attributes manage the status model's TSP-Link® system summary register 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
```

enableRegister	The system summary 2 register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.system2.EXTENSION_BIT` `status.system2.EXT`	Bit B0 decimal value: 1
B1	`status.system2.NODE15`	Bit B1 decimal value: 2
B2	`status.system2.NODE16`	Bit B2 decimal value: 4
B3	`status.system2.NODE17`	Bit B3 decimal value: 8
B4	`status.system2.NODE18`	Bit B4 decimal value: 16
B5	`status.system2.NODE19`	Bit B5 decimal value: 32
B6	`status.system2.NODE20`	Bit B6 decimal value: 64
B7	`status.system2.NODE21`	Bit B7 decimal value: 128
B8	`status.system2.NODE22`	Bit B8 decimal value: 256
B9	`status.system2.NODE23`	Bit B9 decimal value: 512
B10	`status.system2.NODE24`	Bit B10 decimal value: 1024
B11	`status.system2.NODE25`	Bit B11 decimal value: 2048
B12	`status.system2.NODE26`	Bit B12 decimal value: 4096
B13	`status.system2.NODE27`	Bit B13 decimal value: 8192
B14	`status.system2.NODE28`	Bit B14 decimal value: 16384
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 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 B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2048 + 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	1024	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
```

Sets bits B11 and B14 of the system summary 2 enable register using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system2.enable = enableRegister
```

Sets bits B11 and B14 of the system summary 2 enable register using a decimal value.

Also see

status.system.* (on page 7-314)
status.system3.* (on page 7-318)
System summary and standard event registers (on page E-6)

status.system3.*

These attributes manage the status model's TSP-Link® system summary register 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
```

enableRegister	The system summary 3 register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.system3.EXTENSION_BIT` `status.system3.EXT`	Bit B0 decimal value: 1
B1	`status.system3.NODE29`	Bit B1 decimal value: 2
B2	`status.system3.NODE30`	Bit B2 decimal value: 4
B3	`status.system3.NODE31`	Bit B3 decimal value: 8
B4	`status.system3.NODE32`	Bit B4 decimal value: 16
B5	`status.system3.NODE33`	Bit B5 decimal value: 32
B6	`status.system3.NODE34`	Bit B6 decimal value: 64
B7	`status.system3.NODE35`	Bit B7 decimal value: 128
B8	`status.system3.NODE36`	Bit B8 decimal value: 256
B9	`status.system3.NODE37`	Bit B9 decimal value: 512
B10	`status.system3.NODE38`	Bit B10 decimal value: 1024
B11	`status.system3.NODE39`	Bit B11 decimal value: 2048
B12	`status.system3.NODE40`	Bit B12 decimal value: 4096
B13	`status.system3.NODE41`	Bit B13 decimal value: 8192
B14	`status.system3.NODE42`	Bit B14 decimal value: 16384
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 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 B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2048 + 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	1024	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
```

Sets bits B11 and B14 of the system summary 3 enable register using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system3.enable = enableRegister
```

Sets bits B11 and B14 of the system summary 3 enable register using a decimal value.

Also see

status.system2.* (on page 7-316)
status.system4.* (on page 7-320)
System summary and standard event registers (on page E-6)

status.system4.*

These attributes manage the status model's TSP-Link® system summary register 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
```

enableRegister	The system summary 4 register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.system4.EXTENSION_BIT` `status.system4.EXT`	Bit B0 decimal value: 1
B1	`status.system4.NODE43`	Bit B1 decimal value: 2
B2	`status.system4.NODE44`	Bit B2 decimal value: 4
B3	`status.system4.NODE45`	Bit B3 decimal value: 8
B4	`status.system4.NODE46`	Bit B4 decimal value: 16
B5	`status.system4.NODE47`	Bit B5 decimal value: 32
B6	`status.system4.NODE48`	Bit B6 decimal value: 64
B7	`status.system4.NODE49`	Bit B7 decimal value: 128
B8	`status.system4.NODE50`	Bit B8 decimal value: 256
B9	`status.system4.NODE51`	Bit B9 decimal value: 512
B10	`status.system4.NODE52`	Bit B10 decimal value: 1024
B11	`status.system4.NODE53`	Bit B11 decimal value: 2048
B12	`status.system4.NODE54`	Bit B12 decimal value: 4096
B13	`status.system4.NODE55`	Bit B13 decimal value: 8192
B14	`status.system4.NODE56`	Bit B14 decimal value: 16384
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 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 B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2048 + 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	1024	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
```

Sets bit B11 and bit B14 of the system summary 4 enable register using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system4.enable = enableRegister
```

Sets bit B11 and bit B14 of the system summary 4 enable register using a decimal value.

Also see

status.system3.* (on page 7-318)
status.system5.* (on page 7-322)
System summary and standard event registers (on page E-6)

status.system5.*

These attributes manage the status model's TSP-Link® system summary register 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
```

enableRegister	The system summary 5 register's status. 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 sets (on page E-2) and Enable and transition registers (on page E-18). The individual bits of this register have the following meanings:

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.system5.NODE57`	Bit B1 decimal value: 2
B2	`status.system5.NODE58`	Bit B2 decimal value: 4
B3	`status.system5.NODE59`	Bit B3 decimal value: 8
B4	`status.system5.NODE60`	Bit B4 decimal value: 16
B5	`status.system5.NODE61`	Bit B5 decimal value: 32
B6	`status.system5.NODE62`	Bit B6 decimal value: 64
B7	`status.system5.NODE63`	Bit B7 decimal value: 128
B8	`status.system5.NODE64`	Bit B8 decimal value: 256
B9-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	1024	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
```

Sets bits B1 and B4 of the system summary 5 enable register using constants.

Example 2

```
-- decimal 18 = binary 0000 0000 0001 0010
enableRegister = 18
status.system5.enable = enableRegister
```

Sets bits B1 and B4 of the system summary 5 enable register using a decimal value.

Also see

status.system4.* (on page 7-320)
System summary and standard event registers (on page E-6)

SweepILinMeasureV()

This KISweep factory script (on page 5-22) 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(`*`smu, starti, stopi, stime, points`*`)`

smu	System SourceMeter® instrument channel (set to `smua`)
starti	Sweep start current in amperes
stopi	Sweep stop current in amperes
stime	Settling time in seconds; occurs after stepping the source and before performing a measurement
points	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.

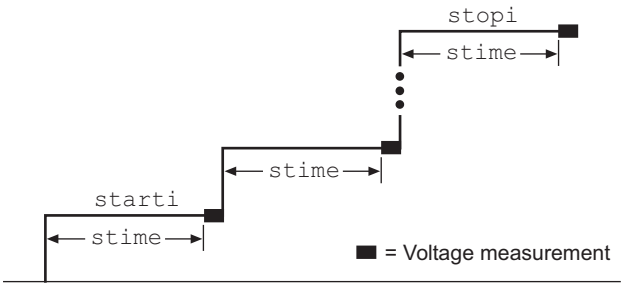
Performs a linear current sweep with voltage measured at every step (point):

1. Sets the *smu* to output *starti* amperes, allows the source to settle for *stime* seconds, and then 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 linear step size is automatically calculated as follows:

step = (stopi – starti) / (points – 1)

Figure 106: 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 mA. Voltage is measured at every step (point) in the sweep. Because *stime* is set for 0 s, voltage will be measured as fast as possible after each current step.

Also see

None

SweepIListMeasureV()

This KISweep factory script (on page 5-22) 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(smu, ilist, stime, points)
```

smu	System SourceMeter® instrument channel (set to `smua`)
ilist	Arbitrary list of current source values; *ilist = {value1, value2, ...valueN}*
stime	Settling time in seconds; occurs after stepping the source and before performing a measurement
points	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

```
myilist = {-100E-9, 100E-9, -1E-6, 1E-6,
   -1E-3, 1E-3}

SweepIListMeasureV(smua, myilist,
   500E-3, 6)
```

This function performs a six-point current list sweep starting at the first point in `myilist`. Voltage is measured at every step (point) in the sweep. The source will be allowed to settle on each step for 500 ms before a measurement is performed.

Also see

None

SweepILogMeasureV()

This KISweep factory script (on page 5-22) 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(smu, starti, stopi, stime, points)
```

smu	System SourceMeter® instrument channel (set to `smua`)
starti	Sweep start current in amperes
stopi	Sweep stop current in amperes
stime	Settling time in seconds; occurs after stepping the source and before performing a measurement
points	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

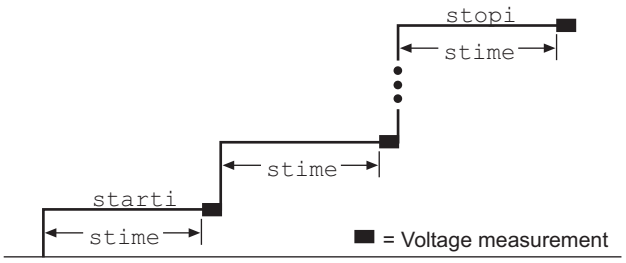
For example, for a five-point sweep (*points* = 5), a measurement is performed at `MeasurePoint` 1, 2, 3, 4, and 5.

LogStepSize = (log10(stopi) – log10(starti)) / (points – 1)

*LogStep = (MeasurePoint – 1) * (LogStepSize)*

*SourceStepLevel = antilog(LogStep) * starti*

Figure 107: 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 will be allowed to settle on each step for 1 ms before a measurement is performed.

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

None

SweepVLinMeasureI()

This KISweep factory script (on page 5-22) 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(`*smu, startv, stopv, stime, points*`)`

smu	System SourceMeter® instrument channel (set to `smua`)
startv	Sweep start voltage in volts
stopv	Sweep stop voltage in volts
stime	Settling time in seconds; occurs after stepping the source and before performing a measurement
points	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.

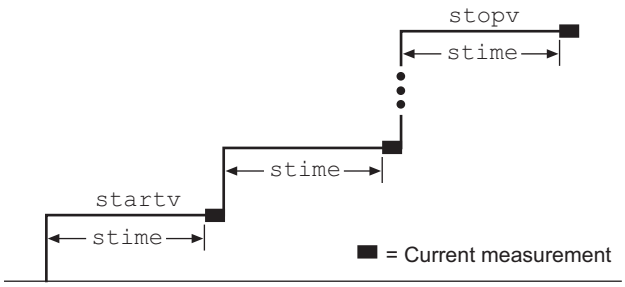
Performs a linear voltage sweep with current measured at every step (point):

1. Sets the *smu* to output *startv* amperes, allows the source to settle for *stime* seconds, and then performs a current 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 *stopv* amperes step.

The linear step size is automatically calculated as follows:

step = (stopv – startv) / (points – 1)

Figure 108: 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

None

SweepVListMeasureI()

This KISweep factory script (on page 5-22) 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(smu, vlist, stime, points)
```

smu	System SourceMeter® instrument channel (set to `smua`)
vlist	Arbitrary list of voltage source values; *vlist = {value1, value2, ... valueN}*
stime	Settling time in seconds; occurs after stepping the source and before performing a measurement
points	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

`myilist = {-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 will be allowed to settle on each step for 500 ms before a measurement is performed.

Also see

None

SweepVLogMeasureI()

This KISweep factory script (on page 5-22) 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(smu, startv, stopv, stime, points)
```

smu	System SourceMeter® instrument channel (set to `smua`)
startv	Sweep start voltage in volts
stopv	Sweep stop voltage in volts
stime	Settling time in seconds; occurs after stepping the source and before performing a measurement
points	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 performs a current measurement.
2. Sets the *smu* to output the next volts step, allows the source to settle for *stime* seconds, and then performs 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

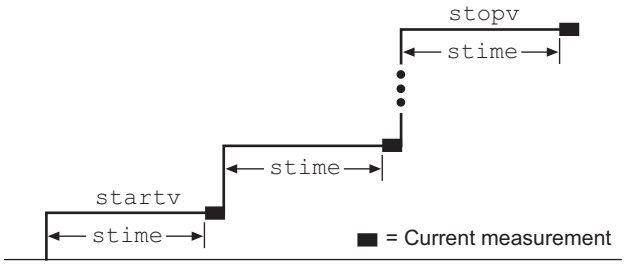
For example, for a five-point sweep (*points* = 5), a measurement is performed at `MeasurePoint` 1, 2, 3, 4, and 5.

LogStepSize = (log10(stopi) – log10(starti)) / (points – 1)

*LogStep = (MeasurePoint – 1) * (LogStepSize)*

*SourceStepLevel = antilog(LogStep) * startv*

Figure 109: 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

None

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)
```

Resets the timer and measures the time since the reset.

Output:

```
1.469077e+01
```

The output will vary. The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Example 2

```
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)
```

Sets the beeper, resets the timer, sets a delay, then verifies the time of the delay before the next beeper.

Output:

```
reset timer
timer after delay: 5.00e-01
```

Also see

timer.reset() (on page 7-333)

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 7-332)

trigger.blender[N].clear()

This function clears the blender event detector and resets blender *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].clear()
```

N	The blender number (1 to 4)

Details

This function sets the blender event detector to the undetected state and resets the event detector's overrun indicator.

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
```

eventID	Trigger event number
N	The blender number (1 to 4)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to trigger events from this trigger blender.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender operates in OR mode or AND mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Trigger blender N reset	Not saved	false (AND mode)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The orenable mode: • `true`: OR mode • `false`: AND mode
N	The trigger blender (1 to 4)

Details

This attribute selects whether the blender waits for any one event (the “OR” mode) or waits for all selected events (the “AND” mode) before signaling an output event.

Also see

trigger.blender[N].reset() (on page 7-335)

trigger.blender[N].overrun

This attribute indicates whether or not an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Trigger blender N clear Trigger blender N reset	Not applicable	Not applicable

Usage

```
overrun = trigger.blender[N].overrun
```

overrun	Trigger blender overrun state
N	The trigger event blender (1 to 4)

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 attribute 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 7-335)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].reset()
```

N	The trigger event blender (1 or 2)

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`.

Also see

trigger.blender[N].orenable (on page 7-334)
trigger.blender[N].overrun (on page 7-335)
trigger.blender[N].stimulus[M] (on page 7-336)

trigger.blender[N].stimulus[M]

This attribute specifies which events trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Trigger blender N reset	Not saved	0

Usage

```
eventID = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = eventID
```

eventID	The event that triggers the blender action; see **Details**
N	An integer representing the trigger event blender (`1` to `4`)
M	An integer representing the stimulus index (`1` to `4`)

Details

There are four acceptors 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 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 measure action

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.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 *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed.
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command device_trigger. 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.
* 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 7-335)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.blender[*N*].wait(*timeout*)

triggered	Trigger detection indication for blender
N	The trigger blender (1 to 4) on which to wait
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```
digio.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 7-333)

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 an event has been detected since the last `trigger.wait()` call. This function clears the trigger's event detector and discards the previous history of command interface trigger events.

Also see

trigger.wait() (on page 7-346)

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
```

eventID	The command interface trigger event number

Details

You can set the stimulus of any trigger event detector to the value of this constant to have it respond to command interface trigger events.

Also see

None

trigger.timer[N].clear()

This function clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].clear()
```

N	Trigger timer number to clear (1 to 4)

Details

This function sets the timer event detector to the undetected state and resets the overrun indicator.

Also see

trigger.timer[N].count (on page 7-339)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer is triggered.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Trigger timer N reset	Not saved	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

count	Number of times to repeat the trigger
N	A trigger timer value from 1 to 4

Details

If *count* is set to a number greater than 1, the timer automatically starts the next delay at expiration of the previous delay.

Set *count* to zero (0) to cause the timer to generate trigger events indefinitely.

Example

```
print(trigger.timer[1].count)
```

Read retrigger count for timer number 1.

Also see

trigger.timer[N].clear() (on page 7-339)
trigger.timer[N].reset() (on page 7-343)

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 Trigger timer N reset	Not saved	10e-6

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

interval	Delay interval in seconds
N	Trigger timer number (1 to 4)

Details

Each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

```
trigger.timer[N].delaylist = {interval}
```

This creates a delay list of one value.

Reading this attribute returns the delay interval that will be used the next time the timer is triggered.

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 7-343)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals that are used when triggered.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Trigger timer N reset	Not saved	{10e-6}

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

intervals	Table of delay intervals in seconds
N	Trigger timer number (1 to 4)

Details

Each time the timer is triggered, it uses the next delay period from the array.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be ≥ 50 µs.

Example

```
trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}

DelayList = trigger.timer[3].delaylist
for x = 1, table.getn(DelayList) do
   print(DelayList[x])
end
```

Set a delay list on trigger timer 3 with three delays (50 µs, 100 µs, and 150 µs).

Read the delay list on trigger timer 3:

Output (assuming the list was set with three delays (50 µs, 100 µs, and 150 µs):

```
5.000000000e-05
1.000000000e-04
1.500000000e-04
```

Also see

trigger.timer[N].reset() (on page 7-343)

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
```

eventID	An integer representing the trigger timer number (1 to 8)
N	An integer representing the trigger event number (1 to 4)

Details

This constant is an identification number that identifies events generated by this timer.

Set the stimulus of any trigger event detector to the value of this constant to have it respond to events from this timer.

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 Trigger timer N clear Trigger timer N reset	Not applicable	false

Usage

```
overrun = trigger.timer[N].overrun
```

overrun	Trigger overrun state
N	Trigger timer value (1 to 4)

Details

This attribute indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

`print(trigger.timer[1].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 7-343)

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 Trigger timer N reset	Not saved	false (disabled)

Usage

```
passthrough = trigger.timer[N].passthrough
trigger.timer[N].passthrough = passthrough
```

passthrough	The state of pass-through mode. Set to to one of the following values: `true`: Enabled `false`: Disabled
N	Trigger timer number (1 to 4)

Details

When 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 7-343)

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()
```

N	Trigger timer number (1 to 4)

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`.

Also see

trigger.timer[N].count (on page 7-339)
trigger.timer[N].delay (on page 7-340)
trigger.timer[N].delaylist (on page 7-341)
trigger.timer[N].overrun (on page 7-342)
trigger.timer[N].passthrough (on page 7-343)
trigger.timer[N].stimulus (on page 7-344)

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 Trigger timer N reset	Not saved	0

Usage

```
eventID = trigger.timer[N].stimulus
trigger.timer[N].stimulus = eventID
```

eventID	The event that triggers the timer delay
N	A number representing a trigger timer number (1 to 4)

Details

The *eventID* parameter may be one of the trigger event IDs shown in the following table.

Trigger event IDs*

Event ID	Event description
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the 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 measure action

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.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 *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed.
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command device_trigger. 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.
* 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 wait for that event.

Use zero 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 7-343)

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.timer[N].wait(timeout)
```

triggered	Trigger detection indication
N	Trigger timer number (1 to 4)
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

`triggered = trigger.timer[3].wait(10)` `print(triggered)`	Waits up to 10 seconds for a trigger on timer 3. If `false` is returned, no trigger was detected during the 10-second timeout. If `true` is returned, a trigger was detected.

Also see

trigger.timer[N].clear() (on page 7-339)

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*)

triggered	`true`: A trigger was detected during the timeout period `false`: No triggers were detected during the timeout period
timeout	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A *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 7-338)

tsplink.group

This attribute is the group number of a TSP-Link node used for DTNS.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not saved	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

groupNumber	The group number of the TSP-Link node (0 to 64)

Details

To remove the node from all groups, set the attribute value to 0.

When the node is turned off, the group number for that node changes to 0.

Also see

None

tsplink.master

This attribute reads the node number assigned to the master node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
masterNodeNumber = tsplink.master
```

masterNodeNumber	The node number of the master node

Also see

None

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
```

nodeNumber	Set node to a number (1 to 64)

Details

This attribute sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until the next time `tsplink.reset()` is executed on any node in the system.

Each node connected to the TSP-Link must be assigned a different node number.

Example

`tsplink.node = 2`	Sets the TSP-Link node to number 2.

Also see

tsplink.reset() (on page 7-349)
tsplink.state (on page 7-350)

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	A custom variable that stores the state of the synchronization line
N	The trigger line (1 to 3)

Details

Returns a value of 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 7-349)
tsplink.writebit() (on page 7-359)

tsplink.readport()

This function reads the TSP-Link synchronization 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 returned indicating which register bits 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 Bit B3 corresponds to Line 3. For example, a returned value of 2 has a binary equivalent of 010. Line 2 is high (1), and the other 2 lines are low (0).

Example

```
data = tsplink.readport()
print(data)
```

Reads state of all three TSP lines.
Assuming Line 2 is set high, the output is:

```
2.000000e+00
```

(binary 010)

Also see

tsplink.readbit() (on page 7-348)
tsplink.writebit() (on page 7-359)
tsplink.writeport() (on page 7-359)

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)
```

nodesFound	The number of nodes actually found on the system
expectedNodes	The number of nodes expected on the system, between 1 and 64

Details

This function erases all knowledge of other nodes connected on the TSP-Link 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 initialized 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 will not generate any errors, even if there are no other nodes on the TSP-Link network.

Also returns the number of nodes found.

Also see

tsplink.node (on page 7-347)
tsplink.state (on page 7-350)

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 is powered 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. If it is online, the output is:

```
online
```

Also see

tsplink.node (on page 7-347)
tsplink.reset() (on page 7-349)

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 7-351)
tsplink.trigger[N].mode (on page 7-352)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].pulsewidth (on page 7-355)
tsplink.trigger[N].release() (on page 7-355)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)

tsplink.trigger[N].clear()

This function clears the event detector for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].clear()
```

N	The trigger line (1 to 3)

Details

The event detector for a trigger recalls if a trigger event has been detected since the last `tsplink.trigger[N].wait()` call. This function clears a trigger event detector, discards the previous 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 7-352)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].release() (on page 7-355)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)

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
```

eventID	The trigger event number.
N	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 event detector to the value of this constant to have it respond to trigger events from this line.

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 TSP-Link trigger N reset	Not saved	tsplink.TRIG_BYPASS

Usage

```
mode = tsplink.trigger[N].mode
tsplink.trigger[N].mode = mode
```

mode	The trigger mode
N	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 *mode* can be one of the following values:

Mode	Number value	Description
tsplink.TRIG_BYPASS	0	Allows direct control of the line as a digital I/O line.
tsplink.TRIG_FALLING	1	Detects falling-edge triggers as input. Asserts a TTL-low pulse for output.
tsplink.TRIG_RISING	2	If the programmed state of the line is high, the tsplink.TRIG_RISING mode behaves similar to tsplink.TRIG_RISINGA. If the programmed state of the line is low, the tsplink.TRIG_RISING mode behaves similar to tsplink.TRIG_RISINGM. Use tsplink.TRIG_RISINGA if the line is in the high output state. Use tsplink.TRIG_RISINGM if the line is in the low output state.
tsplink.TRIG_EITHER	3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
tsplink.TRIG_SYNCHRONOUSA	4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low.
tsplink.TRIG_SYNCHRONOUS	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.
tsplink.TRIG_SYNCHRONOUSM	6	Detects rising-edge triggers as an input. Asserts a TTL-low pulse for output.
tsplink.TRIG_RISINGA	7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
tsplink.TRIG_RISINGM	8	Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.

When programmed to any other mode, 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 will be examined. If the output state selected when the mode is changed is high, the actual mode used will be tsplink.TRIG_RISINGA. If the output state selected when the mode is changed is low, the actual mode used will be tsplink.TRIG_RISINGM.

The custom variable mode 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 the synchronization Line 3 to tsplink.TRIG_RISINGM.

Also see

digio.writebit() (on page 7-51)
digio.writeport() (on page 7-52)
tsplink.trigger[N].assert() (on page 7-350)
tsplink.trigger[N].clear() (on page 7-351)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].release() (on page 7-355)
tsplink.trigger[N].reset() (on page 7-356)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)

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 TSP-Link trigger N clear TSP-Link trigger N reset	Not applicable	Not applicable

Usage

```
overrun = tsplink.trigger[N].overrun
```

overrun	Trigger overrun state
N	The trigger line (1 to 3)

Details

Indicates that an event was ignored because the event detector was in the detected state when the event was detected.

Indicates the overrun state of the event detector built into the line itself.

It 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.

It does not indicate output trigger overrun.

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 7-350)
tsplink.trigger[N].clear() (on page 7-351)
tsplink.trigger[N].mode (on page 7-352)
tsplink.trigger[N].release() (on page 7-355)
tsplink.trigger[N].reset() (on page 7-356)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)

tsplink.trigger[N].pulsewidth

This attribute 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	Not saved	10-e6

Usage

```
width = tsplink.trigger[N].pulsewidth
tsplink.trigger[N].pulsewidth = width
```

width	The pulse width (in seconds)
N	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 7-355)

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()
```

N	The trigger line (1 to 3)

Details

Releases a trigger that was asserted with an indefinite pulse width, as well as 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 7-350)
tsplink.trigger[N].clear() (on page 7-351)
tsplink.trigger[N].mode (on page 7-352)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].pulsewidth (on page 7-355)
tsplink.trigger[N].stimulus (on page 7-357)
tsplink.trigger[N].wait() (on page 7-358)

tsplink.trigger[N].reset()

This function resets some of the TSP-Link® trigger settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].reset()
```

N	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`.

Also see

tsplink.trigger[N].pulsewidth (on page 7-355)
tsplink.trigger[N].mode (on page 7-352)
tsplink.trigger[N].overrun (on page 7-354)
tsplink.trigger[N].stimulus (on page 7-357)

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 TSP-Link trigger N reset	Not saved	0

Usage

```
eventID = tsplink.trigger[N].stimulus
tsplink.trigger[N].stimulus = eventID
```

eventID	The event identifier for the triggering event
N	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to 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
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the 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 measure 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 *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed.

Trigger event IDs*	
Event ID	**Event description**
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface. (GPIB only) Occurs when a GET bus command is received. (VXI-11 only) Occurs with the VXI-11 command `device_trigger`. 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.
* 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).	

Also see

tsplink.trigger[N].assert() (on page 7-350)
tsplink.trigger[N].reset() (on page 7-356)

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)
```

triggered	Trigger detection indication. Set to one of the following values: `true`: A trigger is detected during the timeout period `false`: A trigger is not detected during the timeout period
N	The trigger line (1 to 3)
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events 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 7-351)

tsplink.writebit()

This function sets a TSP-Link synchronization line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writebit(bit, data)
```

bit	The trigger line (1 to 3)
data	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 synchronization 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 digital I/O lines.

Example

`tsplink.writebit(3, 0)`	Sets synchronization Line 3 low (0).

Also see

tsplink.readbit() (on page 7-348)
tsplink.readport() (on page 7-349)
tsplink.writeport() (on page 7-359)
tsplink.writeprotect (on page 7-360)

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)
```

data	Value to write to the port (0 to 7)

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other two lines are set low (0).

Write protected lines are not changed (see `tsplink.writeprotect` (on page 7-360)).

The reset() function does not affect the present states of the digital I/O lines.

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`.

Example

`tsplink.writeport(3)`	Sets the synchronization Lines 1 and 2 high (binary 011).

Also see

tsplink.readbit() (on page 7-348)
tsplink.readport() (on page 7-349)
tsplink.writebit() (on page 7-359)
tsplink.writeprotect (on page 7-360)

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	Saved setups	0

Usage

```
mask = tsplink.writeprotect
tsplink.writeprotect = mask
```

mask	An integer that specifies the value of the bit pattern for write-protect; set bits to `1` to write-protect the corresponding digital I/O line

Details

The binary equivalent of *mask* indicates the mask to be set for the I/O port. For example, a *mask* value of `5` has a binary equivalent 101. This *mask* write-protects digital I/O lines 1 and 3.

Example

`tsplink.writeprotect = 5`	Write-protects digital I/O lines 1 and 3.

Also see

Controlling digital I/O lines (on page 3-85)
tsplink.readbit() (on page 7-348)
tsplink.readport() (on page 7-349)
tsplink.writebit() (on page 7-359)
tsplink.writeport() (on page 7-359)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.clear(connectionID)
```

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

```
tspnet.write(mydevice, "print([[hello]])")
print(tspnet.readavailable(mydevice))
```

Write data to device, print how much is available.
Output:
```
6.00000e+000
```

```
tspnet.clear(mydevice)
print(tspnet.readavailable(mydevice))
```

Clear data and print how much data is available again.
Output:
```
0.00000e+000
```

Also see

tspnet.connect() (on page 7-361)
tspnet.readavailable() (on page 7-366)
tspnet.write() (on page 7-371)

tspnet.connect()

This function connects the device processing the command to another device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
connectionID = tspnet.connect(ipAddress)
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

connectionID	The connection ID to be used as a handle in all other tspnet function calls
ipAddress	IP address to which to connect
portNumber	Port number
initString	Initialization string to send to *ipAddress*

Details

This command connects a device to another device through the LAN interface. The default port number is 5025. 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 Model 2651A does not perform any extra processing, prompt handling, error handling, or sending of commands. Additionally, 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 `tspnet.tsp.abortonconnect`, the Model 2651A sends an `abort` command to the remote device on connection. The Model 2651A also enables TSP prompts on the remote device and error management. The Model 2651A 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, or subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

`myID = tspnet.connect("192.0.2.1")`
`if myID then`
`  -- Use myID as needed here`
`  tspnet.disconnect(myID)`
`end`
	Connect to a TSP-enabled device.

Example 2

`myID = tspnet.connect("192.0.2.1", 1394,`
`  "*rst\r\n")`
`if myID then`
`  -- Use myID as needed here`
`  tspnet.disconnect(myID)`
`end`
	Connect to a device that is not TSP-enabled.

Also see

localnode.prompts (on page 7-136)
localnode.showerrors (on page 7-139)
tspnet.tsp.abortonconnect (on page 7-369)
tspnet.disconnect() (on page 7-363)

tspnet.disconnect()

This function disconnects the TSP-Net session specified.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.disconnect(connectionID)
```

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function disconnects the two devices by closing the connection. The *connectionID* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

```
testID = tspnet.connect("192.0.2.0")
-- Use the connection

tspnet.disconnect(testID)
```

Create a TSP-Net session.

Close the session.

Also see

tspnet.connect() (on page 7-361)

tspnet.execute()

This function executes a command string on the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.execute(connectionID, commandString)
value1 = tspnet.execute(connectionID, commandString, formatString)
value1, value2 = tspnet.execute(connectionID, commandString, formatString)
value1, ..., valuen = tspnet.execute(connectionID, commandString, formatString)
```

connectionID	The connection ID returned from `tspnet.connect()`
commandString	The command to send to the remote device
value1	The first value decoded from the response message
value2	The second value decoded from the response message
valuen	The nth value decoded from the response message; there is one return value per format specifier in the format string
...	One or more values separated with commas
formatString	Format string for the output

Details

This command sends the command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (see tspnet.termination() (on page 7-367)). Optionally, when a format string is specified, the command waits for a response from the remote instrument. The Model 2651A decodes the response message according to the format specified in the format string and returns this as return values from the function (see tspnet.read() (on page 7-365) for format specifiers).

When this command is sent to a TSP-enabled instrument, the Model 2651A suspends operation until a timeout error is generated or until the instrument responds, even if no format string is specified. The TSP prompt from the remote instrument is read and thrown away. The Model 2651A 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(myID, "runScript()")`	Command remote device to run script named `runScript`.

Example 2

`tspnet.termination(myID, tspnet.TERM_CRLF)`
`tspnet.execute(myID, "*idn?")`
`print("tspnet.execute returns:",`
`  tspnet.read(myID))`
	Print the `*idn?` string from the remote device.

Also see

tspnet.connect() (on page 7-361)
tspnet.read() (on page 7-365)
tspnet.write() (on page 7-371)

tspnet.idn()

This function retrieves the response of the remote device to *IDN?.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

idnString = tspnet.idn(*connectionID*)

idnString	The returned *IDN? string
connectionID	The connection ID returned from `tspnet.connect()`

Details

This function retrieves the response of the remote device to `*IDN?`.

Example

```
myID =
   tspnet.connect("192.0.2.1")
print(tspnet.idn(myID))
tspnet.disconnect(myID)
```

Output:

```
KEITHLEY INSTRUMENTS INC.,MODEL
    707B,00000170,01.10h
```

Also see

tspnet.connect() (on page 7-361)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valuen = tspnet.read(connectionID, formatString)
```

value1	The first value decoded from the response message
value2	The second value decoded from the response message
valuen	The nth value decoded from the response message; there is one return value per format specifier in the format string
...	One or more values separated with commas
connectionID	The connection ID returned from `tspnet.connect()`
formatString	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following identifiers:

`%[width]s`	Read data until the specific length
`%[max width]t`	Read data until the specific length or delimited by punctuation
`%[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 containing the data until a new line is reached. If no data is available, the Model 2651A pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When reading from a TSP-enabled remote instrument, the Model 2651A removes Test Script Processor (TSP®) prompts and places any errors received from the remote instrument into its own error queue. The Model 2651A prefaces errors from the remote device with "Remote Error," and follows this with the error number and error description.

Example

`tspnet.write(myID, "*idn?\r\n")`	Send the `"*idn?\r\n"` message to the instrument connected as `myID`.
`print("write/read returns:", tspnet.read(myID))`	Display the response that is read from `myID` (based on the `*idn?` message).

Also see

tspnet.connect() (on page 7-361)
tspnet.readavailable() (on page 7-366)
tspnet.timeout (on page 7-368)
tspnet.write() (on page 7-371)

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	Yes			

Usage

```
bytesAvailable = tspnet.readavailable(connectionID)
```

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from `tspnet.connect()`

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")

repeat
   bytes = tspnet.readavailable(ID)
until bytes > 0

print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that will create data.

Wait for data to be available.

Also see

tspnet.connect() (on page 7-361)
tspnet.read() (on page 7-365)

tspnet.reset()

This function disconnects all TSP-Net sessions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

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	Yes			

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

type	An enumerated value indicating the termination type: • 1 or tspnet.TERM_LF • 4 or tspnet.TERM_CR • 2 or tspnet.TERM_CRLF • 3 or tspnet.TERM_LFCR
connectionID	The connection ID returned from tspnet.connect()
termSequence	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

```
myID = tspnet.connect("192.0.2.1")
if myID then
   tspnet.termination(myID, tspnet.TERM_LF)
end
```

Sets termination type for IP address 192.0.2.1 to TERM_LF.

Also see

tspnet.connect() (on page 7-361)
tspnet.disconnect() (on page 7-363)

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)	Yes	Instrument reset	Not saved	20.0

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

value	The timeout duration in seconds (0.001 s to 30.000 s)

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands will wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution, but is only accurate to the nearest 10 ms.

Example

```
tspnet.timeout = 2.0
```

Sets the timeout duration to two seconds.

Also see

tspnet.connect() (on page 7-361)
tspnet.execute() (on page 7-363)
tspnet.read() (on page 7-365)

tspnet.tsp.abort()

This function stops remote instrument execution.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.tsp.abort(connectionID)
```

connectionID	Integer value used as a handle for other tspnet commands

Details

Sends an abort command to the remote instrument.

Example

`tspnet.tsp.abort(myConnection)`	Stops `myConnection`.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset	Not saved	1 (enabled)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	1 (enable) or 0 (disable)

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 issue an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument will not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been issued. See Communications interfaces (on page 2-73).

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 7-361)

tspnet.tsp.rbtablecopy()

This function copies a reading buffer synchronous table from a remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
table = tspnet.tsp.rbtablecopy(connectionID, name)
table = tspnet.tsp.rbtablecopy(connectionID, name, startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other tspnet commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function 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 will return 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.rbtablecopy(testConnection, "testRemotebuffername.readings", 1, 3)`
`print(t[1], t[2], t[3])`
	Copies the specified readings table for buffer items 1 through 3, then displays the first three readings. Sample 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 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)
```

connectionID	Integer value used as an identifier for other `tspnet` commands
name	The name that is assigned to the script
script	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions at a later time.

If no name is specified, the script will be 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 7-363)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`tspnet.write(`*`connectionID`*`, `*`inputString`*`)`

connectionID	The connection ID returned from `tspnet.connect()`
inputString	The string to be written

Details

The `tspnet.write()` function sends *inputString* to the remote instrument. It does not wait for command completion on the remote instrument.

The Model 2651A 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 run script named "myID".

Also see

tspnet.connect() (on page 7-361)

tspnet.read() (on page 7-365)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
userstring.add(name, value)
```

name	The name of the string; the key of the key-value pair
value	The string to associate with *name*; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

Example

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
```

Stores user-defined strings in nonvolatile memory.

Also see

userstring.catalog() (on page 7-372)
userstring.delete() (on page 7-373)
userstring.get() (on page 7-374)

userstring.catalog()

This function creates an iterator for the user string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do body end
```

name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The catalog provides access for userstring 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 strings in nonvolatile memory.

Example 2

```
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
```

The above output lists the user strings added in the example for the `userstring.add()` function. Notice the key-value pairs are not listed in the order they were added.

Also see

userstring.add() (on page 7-372)
userstring.delete() (on page 7-373)
userstring.get() (on page 7-374)

userstring.delete()

This function deletes a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
userstring.delete(name)
```

name	The name (key) of the key-value pair of the userstring 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 7-372)
userstring.catalog() (on page 7-372)
userstring.get() (on page 7-374)

userstring.get()

This function retrieves a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = userstring.get(name)
```

value	The value of the userstring key-value pair
name	The name (key) of the userstring

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

```
value = userstring.get("assetnumber")
print(value)
```

Output: 236

Also see

userstring.add() (on page 7-372)
userstring.catalog() (on page 7-372)
userstring.delete() (on page 7-373)

waitcomplete()

This function waits for all overlapped commands in a specified group to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
waitcomplete()
waitcomplete(group)
```

group	Specifies which TSP-Link group on which to wait

Details

This function will wait for all previously started overlapped commands to complete.

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 0 is specified for the *group*, this function waits for all nodes in the system.

NOTE

Any nodes that are not assigned to a group (group number is 0) are part of the master node's group.

Example 1

`waitcomplete()`	Waits for all nodes in the local group.

Example 2

`waitcomplete(G)`	Waits for all nodes in group G.

Example 3

`waitcomplete(0)`	Waits for all nodes on the TSP-Link network.

Also see

None

Section 8

Troubleshooting guide

In this section:

Introduction 8-1
Error summary 8-1
Error effects on scripts 8-1
Reading errors 8-2
LAN troubleshooting suggestions..... 8-6

Introduction

Troubleshooting information includes information on the Keithley Instruments Model 2651A High Power System SourceMeter® Instrument error levels, how to read errors, and a complete listing of error messages.

Error summary

Error messages are listed in Error summary list (on page 8-2). The following error levels are supported:

NO_SEVERITY	Informational status message only.
INFORMATIONAL	Informational status message only.
RECOVERABLE	Error not serious, can be recovered.
SERIOUS	Error serious, but unit still operational by correcting error.
FATAL	Unit nonoperational.

Error effects on scripts

Most errors will not abort a running script. The only time a script is aborted is when a Lua run-time error (error number -286) is detected. Run-time errors are caused by actions such as trying to index into a variable that is not a table. Syntax errors (error number -285) in a script/command will not technically abort the script, but it will prevent the script/command from being executed in the first place.

Reading errors

When errors occur, the error messages are placed in the error queue (see Queues (on page E-2)). The following table lists commands associated with the error queue (see Remote commands (on page 5-1) for more information):

Error queue command	Description
`errorqueue.clear()`	Clear error queue of all errors
`errorqueue.count`	Number of messages in the error/event queue
`errorqueue.next()`	Request error message

For example, the following commands request the next complete error information from the error queue and then output the message portion of the error:

```
errorCode, message, severity, errorNode = errorqueue.next()
print(message)
```

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 Over-run
-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
-223	RECOVERABLE	Too Much Data
-222	RECOVERABLE	Parameter Data Out of Range
-221	RECOVERABLE	Settings Conflict
-220	RECOVERABLE	Parameter
-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

Error summary

Error number	Error level	Error Message
-121	RECOVERABLE	Invalid Character In Number
-120	RECOVERABLE	Numeric Data
-109	RECOVERABLE	Missing Parameter
-108	RECOVERABLE	Parameter Not Allowed
-105	RECOVERABLE	Trigger Not Allowed
-104	RECOVERABLE	Data Type
-101	RECOVERABLE	Invalid Character
0	NO_SEVERITY	Queue Is Empty
603	RECOVERABLE	Power On State Lost
702	FATAL	Unresponsive digital FPGA
802	RECOVERABLE	Output Blocked By Interlock
820	RECOVERABLE	Parsing Value
900	FATAL	Internal System
1100	RECOVERABLE	Command Unavailable
1101	RECOVERABLE	Parameter Too Big
1102	RECOVERABLE	Parameter Too Small
1103	RECOVERABLE	Max Greater Than Min
1104	RECOVERABLE	Too many digits for param type
1106	RECOVERABLE	Battery Not Present
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist
1109	RECOVERABLE	Menu name already exists
1110	FATAL	Catastrophic analog supply failure
1200	RECOVERABLE	TSPlink initialization failed
1201	RECOVERABLE	TSPlink initialization failed
1202	RECOVERABLE	TSPlink initialization failed
1203	RECOVERABLE	TSPlink initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSPlink initialization failed
1205	RECOVERABLE	TSPlink initialization failed (no remote nodes found)
1206	RECOVERABLE	TSPlink initialization failed
1207	RECOVERABLE	TSPlink initialization failed
1208	RECOVERABLE	TSPlink initialization failed
1209	RECOVERABLE	TSPlink initialization failed
1210	RECOVERABLE	TSPlink initialization failed (node ID conflict)
1211	RECOVERABLE	Node %u is inaccessible
1212	RECOVERABLE	Invalid node ID
1400	RECOVERABLE	Expected at least %d parameters
1401	RECOVERABLE	Parameter %d 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
1501	RECOVERABLE	Invalid parity setting
1502	RECOVERABLE	Invalid terminator setting
1503	RECOVERABLE	Invalid bits setting

Error summary

Error number	Error level	Error Message
1504	RECOVERABLE	Invalid flow control setting
1600	RECOVERABLE	Maximum GPIB message length exceeded
1700	RECOVERABLE	Display area boundary exceeded
1800	RECOVERABLE	Invalid Digital Trigger Mode
1801	RECOVERABLE	Invalid digital I/O Line
2000	RECOVERABLE	Flash download error
2001	RECOVERABLE	Cannot flash with error in queue
2100	FATAL	Could not open socket
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
2107	RECOVERABLE	Duplicate IP address detected
2108	RECOVERABLE	DHCP lease lost
2109	RECOVERABLE	LAN cable disconnected
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
2208	RECOVERABLE	Cannot create directory
2209	RECOVERABLE	Cannot remove directory
2210	RECOVERABLE	File is not a valid script format
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
2401	RECOVERABLE	Invalid specified connection
2402	RECOVERABLE	TSPnet remote error: %s, where %s 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

Error summary

Error number	Error level	Error Message
2408	RECOVERABLE	TSPnet write failure, aborted
2409	RECOVERABLE	TSPnet write failure, timeout
2410	RECOVERABLE	TSPnet max connections reached
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 musty specify both port and init
2417	RECOVERABLE	TSPnet disconnected by other side
4900	RECOVERABLE	Reading buffer index %s is invalid
4901	RECOVERABLE	The maximum index for this buffer is %d
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
5003	SERIOUS	Saved calibration constants corrupted
5004	SERIOUS	Operation conflicts with CALA sense mode
5005	SERIOUS	Value too big for range
5007	SERIOUS	Operation would exceed safe operating area of the instrument
5008	SERIOUS	Operation not permitted while output is on
5009	SERIOUS	Unknown sourcing function
5010	SERIOUS	No such SMU function
5011	SERIOUS	Operation not permitted while cal is locked
5012	SERIOUS	Cal data not saved - save or restore before lock
5013	SERIOUS	Cannot save cal data - unlock before save
5014	SERIOUS	Cannot restore cal data - unlock before restore
5015	SERIOUS	Save to cal set disallowed
5016	SERIOUS	Cannot change cal date - unlock before operation
5017	SERIOUS	Cannot change cal constants - unlock before operation
5018	SERIOUS	Cal version inconsistency
5019	SERIOUS	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	SERIOUS	Cannot set filter.count > 1 when measure.count > 1
5027	SERIOUS	Unlock cal data with factory password
5028	SERIOUS	Cannot perform requested operation while source autorange is enabled
5029	SERIOUS	Cannot save without changing cal date and cal due values
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	SERIOUS	Invalid while overlapped measure
5043	SERIOUS	Cannot perform requested operation while voltage measure autorange is enabled

Error summary

Error number	Error level	Error Message
5044	SERIOUS	Cannot perform requested operation while current measure autorange is enabled
5045	SERIOUS	Cannot perform requested operation while filter is enabled
5046	SERIOUS	SMU too hot
5047	RECOVERABLE	Minimum timestamp resolution is 1µs
5048	SERIOUS	Contact check not valid with HIGH-Z output off
5049	SERIOUS	Contact check not valid while an active current source
5050	SERIOUS	I limit too low for contact check
5051	FATAL	Model number/SMU hardware mismatch. Disconnect DUT and cycle power
5052	RECOVERABLE	Interlock engaged; system stabilizing
5053	RECOVERABLE	Unstable output detected - Measurements may not be valid
5054	RECOVERABLE	High C voltage limit exceeded
5055	SERIOUS	Cannot change adjustment date - change cal constants before operation
5059	SERIOUS	trigger.source.action enabled without configuration
5060	SERIOUS	trigger.measure.action enabled without configuration
5061	SERIOUS	Operation not permitted while OUTPUT is off
5063	SERIOUS	Cannot perform requested operation while measure autozero is on
5064	SERIOUS	Cannot use reading buffer that collects source values
5065	SERIOUS	I range too low for contact check
5066	SERIOUS	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 instrument's web interface, check the following items:

- Verify that the network cable is in the LAN port, not one of the TSP-Link® ports (see the description in 5. LAN (on page 2-6)).
- Verify that the network cable is in the correct port on the computer. The side LAN port of a laptop may be disabled while the unit is in a docking station.
- Verify that the correct Ethernet card's configuration information was used during the setup procedure.
- Verify that the computer's network card is enabled.
- Verify the instrument's IP address is compatible with the IP address on the computer.
- Verify the instrument's subnet mask address is the same as the computer's subnet mask address.
- Turn the instrument's power off, and then on.
- Restart your computer.
- Contact your system administrator for assistance.

Section 9

Frequently asked questions (FAQs)

In this section:

How do I optimize performance? 9-1
How do I upgrade the firmware? 9-2
How do I use the Digital I/O port? 9-2
How do I trigger other instruments? 9-2
How do I generate a service request? 9-3
How do I store measurements in nonvolatile memory? 9-4
When should I change the output off state? 9-5
How do I make contact check measurements? 9-5

How do I optimize performance?

There are three primary factors that affect measurement accuracy and speed:

- **Warm-up:** For rated measurement accuracy, allow the Model 2651A to warm up for at least two hours before use.
- **Speed setting:** The Speed setting affects both speed and accuracy (see Setting speed (on page 9-1)).
- **Autozero:** Autozero can be disabled to increase speed at the expense of accuracy (see Disabling autozero to increase speed (on page 9-1)).

Setting speed

To set speed from the front panel:

Press the **SPEED** key, and then choose the setting based on your requirements:

- Choose the **FAST** setting for the fastest measurements (note that measurement accuracy will be reduced).
- For best accuracy, use the **HI-ACCURACY** setting (note that speed will be reduced).
- To balance accuracy and speed, use the **MED** or **NORMAL** setting.

For additional information, including the OTHER setting and remote configuration, refer to Setting speed (on page 2-72).

Disabling autozero to increase speed

In certain measuring situations, disabling autozero (setting it to OFF) can increase measurement speed. If autozero is disabled, accuracy will drift with time and temperature.

NOTE

Turning autozero OFF will disable the autozero function and possibly increase measurement speed. To minimize drift, setting autozero to ONCE will perform an autozero one time (at the time when it is selected) and then disable the autozero function. For a more detailed discussion of autozero, see the Autozero (on page 2-22) topic.

To configure autozero from the front panel:

1. Press the **CONFIG** key, and then select **MEAS** from the menu.
2. Select **AUTO-ZERO**, then press the **ENTER** key or the navigation wheel ◷.
3. Select the desired mode (**OFF**, **ONCE**, or **AUTO**), and then press the **ENTER** key or the navigation wheel ◷.
4. Press the **EXIT (LOCAL)** key as necessary to return to the normal display.

Refer to the Remote command autozero (on page 2-23) topic 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 A-4).

How do I use the Digital I/O port?

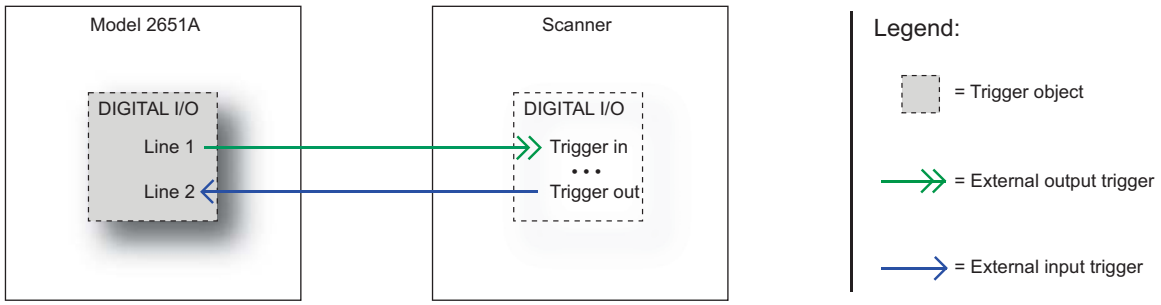
For information on the Model 2651A digital I/O port, see Digital I/O (on page 3-83).

How do I trigger other instruments?

Triggering a scanner

A typical test scenario might call for using the Model 2651A with a scanner to test a number of DUTs in sequence. A basic example of this uses the Model 2651A 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 110: Triggering a scanner



Programming triggering

The programming example below illustrates how to set up triggering. The example sets the output 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(timeout)
```

More information on triggering

For additional information, see Triggering (on page 3-32).

How do I generate a service request?

NOTE

For detailed information on this topic, see the Status Model (on page E-1) section of this manual.

Setting up a service request

The exact programming steps necessary to generate a GPIB service request (SRQ) will vary somewhat depending on the event(s) intended to generate the SRQ. In general, the steps used will be as follows:

1. Clear all status registers to prevent anomalous events from generating an SRQ.
2. Set bits in the appropriate operation: questionable, measurement enable, or transition register(s).
3. Set the proper summary bit in the service request enable register. At least one bit in this register must always be set, but the exact bit(s) to be set will depend on the desired SRQ event(s).

Service request programming example

The example below shows how to program the Model 2651A to generate an SRQ 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
-- Enable status SRQ MSB.
status.request_enable = status.MSB
```

Polling for SRQs

To determine if the Model 2651A is the GPIB device that generated the SRQ, serial poll the unit for the status byte, and test to see if the corresponding summary bits are set.

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. An SMU has two dedicated reading buffers for measured readings, source values, and timestamps: Buffer 1 (`nvbuffer1`) and Buffer 2 (`nvbuffer2`). Reading buffers can be saved to nonvolatile memory so that saved data will load automatically when power is cycled. For additional information, see Saving reading buffers (on page 3-8).

When should I change the output off state?

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and compliance levels before connecting the Model 2651A High Power System SourceMeter® Instrument to a device that can deliver energy such as other voltage sources, batteries, capacitors, solar cells, or other System SourceMeter instruments. Program recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance levels may result in damage to the instrument or to the device under test.

The Model 2651A provides multiple output-off states. The multiple states are required because different types of connected devices (or loads) require different behaviors from the High Power System SourceMeter® Instrument when its output is turned off. For instance, a passive device such as a diode is untroubled 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, thoughtful selection of the proper output-off state is important in order to prevent damage to devices and instruments. This is especially true when the device can deliver energy to the Model 2651A, such as a battery, capacitor, or when another SourceMeter instrument is connected across the output terminals. In these cases, you want to use an output-off state that isolates the instrument from the device, either setting `smuX.source.offfunc = smuX.OUTPUT_DCAMPS`, or `smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`, as applicable. There are other guidelines to follow when connecting the output of multiple Model 2651A instruments together to obtain a larger current or voltage. Please refer to following references for more information:

- Refer to Combining SMU outputs (on page 2-48)
- User's Manual section "Combining SMUs"
- Keithley application notes on the Keithley Instruments website (http://www.keithley.com)

How do I make contact check measurements?

For information on making contact check measurements, see Contact check measurements (on page 2-37) and Contact check (on page 4-16).

Section 10

Next steps

In this section:

Next steps 10-1

Next steps

Additional Model 2651A information

For additional information on the Model 2651A, refer to:

- The CD-ROM (ships with the product): Contains software tools, drivers, and product documentation, including documentation for switch cards that are compatible with the Model 2651A
- The Keithley Instruments website (http://www.keithley.com): Contains the most up-to-date information; from the website, you can access:
  - The Knowledge Center, which contains the following handbooks:
    - *The Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*
    - *Switching Handbook: A Guide to Signal Switching in Automated Test Systems*
  - Application notes
  - Updated drivers
  - Information on related products, including:
    - The Model 4200-SCS Semiconductor Characterization System
    - The Series 2600A System SourceMeter Instruments
- Your local Field Applications Engineer can help you with product selection, configuration, and usage. Check the website for contact information.

Appendix A

Maintenance

In this appendix:

Introduction A-1
Line fuse replacement A-1
Front panel tests A-2
Upgrading the firmware A-4

Introduction

The information in this section deals with routine maintenance of the Keithley Instruments Model 2651A High Power System SourceMeter® Instrument that can be performed by the operator.

Line fuse replacement

A fuse located on the Model 2651A rear panel protects the power line input of the High Power System SourceMeter® Instrument.

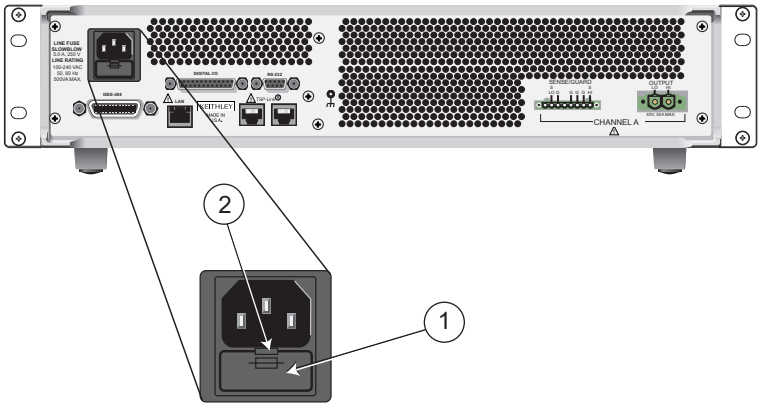
⚠ WARNING

Disconnect the line cord at the rear panel, and remove all test leads connected to the instrument before replacing the line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.

NOTE

The power line fuse is accessible from the rear panel, just below the AC power receptacle (see the following figure).

Figure 111: Fuse replacement



Perform the following steps to replace the line fuse:

1. Power off the unit and remove the line cord.
2. The fuse drawer (1) is located below the AC receptacle. A small tab is located on the top of the fuse drawer (2). Using a thin-bladed knife or a screwdriver, pry this tab away from the AC receptacle.
3. Slide the fuse drawer out to gain access to the fuse (the fuse drawer does not pull completely out of the power module).
4. Snap the fuse out of the drawer and replace it with the same type (the fuse is specified in the table below).

⚠ CAUTION

To prevent instrument damage, use only the correct fuse type (see table).

5. Push the fuse drawer back into the module.

If the power line fuse continues to blow, a circuit malfunction exists and must be corrected. Return the unit to Keithley Instruments for repair.

Line fuse

Line voltage	Rating	Keithley Instruments part number
100 V - 240 V	5 A, 250 V, FAST-ACTING CARTRIDGE FUSE, 5 × 20mm	FU-154-5

Front panel tests

There are two front panel tests: one to test the functionality of the front panel keys and one to test the display.

NOTE

In the following procedures, once highlighted, menu items are selected by pressing the **ENTER** key. Alternatively, menu items can be selected by pressing the navigation wheel ⟳.

Keys test

This test lets you check the functionality of each front panel key.

Perform the following steps to run the KEYS test:

1. If the Model 2651A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Navigate through the menus by turning the navigation wheel ⟳, and then pressing the **ENTER** key to select the items as follows: **DISPLAY > TEST > DISPLAY-TESTS**.
4. Turn the navigation wheel ⟳ until the **KEYS** menu item is highlighted.
5. To start the test, press the **ENTER** key. While testing, when a key is pressed, the label name for that key will be displayed to indicate that it is functioning properly. When the key is released, the message “No keys pressed” is displayed.
6. Pressing the **EXIT (LOCAL)** key tests the **EXIT (LOCAL)** key. However, the second consecutive press of the **EXIT (LOCAL)** key aborts the test and returns the instrument to the FRONT PANEL TESTS menu. Continue pressing the **EXIT (LOCAL)** key to back out of the menu structure.

Display patterns test

This test lets you verify that each pixel and indicator in the vacuum fluorescent display is working properly.

Perform the following steps to run the display test:

1. If the Model 2651A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Navigate through the menus by turning the navigation wheel ⟳, and then pressing the **ENTER** key to select the items as follows: **DISPLAY > TEST > DISPLAY-TESTS**.
4. Turn the navigation wheel ⟳ until the **DISPLAY-PATTERNS** menu item is highlighted.
5. To start the display test, press the **ENTER** key. There are three parts to the display test. Each time the **ENTER** key or the navigation wheel ⟳ is pressed, the next part of the test sequence is selected. The three parts of the test sequence are as follows:
   - Checkerboard pattern and the indicators that are on during normal operation
   - Checkerboard pattern (alternate pixels on) and all the numeric indicators (which are not used) are illuminated
   - Each digit (and adjacent indicators) is sequenced; all of the pixels of the selected digit are on
6. When finished, abort the display test by pressing the **EXIT (LOCAL)** key. The instrument returns to the FRONT PANEL TESTS menu. Continue pressing the **EXIT (LOCAL)** key to back out of the menu structure.

Upgrading the firmware

You can use a USB flash drive to upgrade the firmware.

⚠ CAUTION

Disconnect the input and output terminals before you upgrade.

Complete the following steps to upgrade the firmware.

1. From your PC, go to the Keithley Instruments website (http://www.keithley.com), and then download the latest version of firmware from the website.
2. Insert a USB flash drive into the USB port on your PC.
3. Copy the firmware file to the USB flash drive.
4. If the Model 2651A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.

5. Remove the USB flash drive from your PC and insert it into the USB port on the front panel of the Model 2651A.
6. From the Model 2651A front panel, press the **MENU** key
7. Scroll to the **UPGRADE** menu item (by turning the navigation wheel), and then press the **ENTER** key.
8. Scroll to and select the file (located on the USB flash drive) that contains the appropriate version of firmware.
9. Press the **ENTER** key to upgrade the firmware.

⚠ CAUTION

Do not turn the Model 2651A off while an upgrade is in progress.

Firmware upgrade using the instrument web interface

To upgrade the firmware from the web interface:

1. From the left navigation area, select **Flash Upgrade**.
2. Log in if necessary.
3. Click **Upgrade Firmware**.
4. A file selection dialog box is shown.
5. Select the file that contains the appropriate version of firmware.
6. Click **Open**. A progress dialog box is displayed. When the upgrade begins, the front panel display will also display the progress.
7. After the instrument automatically restarts, it will be ready for use.

Appendix B

Calibration

In this appendix:

Verification.....B-1
Adjustment B-16

Verification

Use the procedures in this section to verify that the Keithley Instruments Model 2651A High Power System SourceMeter® Instrument 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 unit meets factory specifications.
- To determine if calibration is required.
- Following calibration to make sure it was performed properly.

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.

Verification test requirements

Be sure that you perform the verification tests:

- Under the proper environmental conditions.
- After the specified warm-up 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 an example. Always verify values against actual product specifications.

Environmental conditions

Conduct your performance verification procedures in a test environment with:

- An ambient temperature of 18° C to 28° C (65° F to 82° F).
- A relative humidity of less than 70 percent unless otherwise noted.

NOTE

Product specifications that are listed as 18º C to 28º C assume adjustment has been done at 23º C. If the Model 2651A is adjusted at a different temperature, the specifications apply to +/-5º C of that adjustment temperature.

Line power

The Model 2651A requires a line voltage of 100 V to 240 V and a line frequency of 50 Hz or 60 Hz. Verification tests should be performed within this range.

Warm-up period

Allow the Model 2651A to warm up for at least two hours before conducting the verification procedures. If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the instrument's internal temperature to stabilize. Typically, allow one extra hour to stabilize a unit that is 10° C (18° F) outside the specified temperature range.

Also, allow the 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 Model 2651A specifications.

Description	Manufacturer/model	Accuracy	
Digital multimeter		DC voltage	50 mV: ±1500 ppm 90 mV: ±880 ppm 0.5 V: ±200 ppm 0.9 V: ±130 ppm 5 V: ±200 ppm 9 V: ±130 ppm 10 V: ±170 ppm 18 V: ±110 ppm 20 V: ±200 ppm 36 V: ±130 ppm

Description	Manufacturer/model	Accuracy	
Digital multimeter (continued)		DC current	50 nA: ±2700 ppm 90 nA: ±1500 ppm 500 nA: ±1200 ppm 900 nA: ±750 ppm 5 μA: ±600 ppm 9 μA: ±420 ppm 50 μA: ±170 ppm 90 μA: ±110 ppm 0.5 mA: ±150 ppm 0.9 mA: ±100 ppm 5 mA: ±170 ppm 9 mA: ±110 ppm 50 mA: ±150 ppm 90 mA: ±100 ppm 500 mA: ±1600 ppm 900 mA: ±950 ppm 2.5 A: ±420 ppm 4.5 A: ±290 ppm 5 A: ±600 ppm 9 A: ±460 ppm 10 A: ±400 ppm 18 A: ±310 ppm
		DC current Pulse only*	20 A: ±750 ppm 45 A: ±620 ppm
Precision resistor: 0.1 Ω, 250 W, 0.1%	Isotek RUG-Z-R100-0.1-TK1	Resistance**	0.1 Ω: ±75 ppm
* To verify this pulse current, DMM must be able to complete its measurement of the shunt voltage within 500 μs of receiving a trigger signal ** Resistor used to test 1 A, 5 A, 10 A, and 20 A ranges (DC level), and 50 A range (pulse). Before use, characterize the resistor to the uncertainty shown.			

Verification limits

The verification limits stated in this section have been calculated using only the Model 2651A one-year accuracy specifications, and they do not include test equipment uncertainty. If a particular measurement falls outside the allowable range, recalculate new limits based both on the Model 2651A specifications and corresponding test equipment specifications.

Source limits calculations

As an example of how to calculate source verification limits, assume you are testing the Model 2651A 10 V DC output range using a 9 V output value. Using the one-year accuracy specification for the 10 V range DC output of ± (0.02% of output + 5 mV offset), the calculated output limits are:

Output limits = 9.0 V ± [(9.0 V × 0.02%) + 5 mV]
Output limits = 9.0 V ± (0.0018 + 0.005)
Output limits = 9.0 V ± 0.0068 V
Output limits = 8.9932 V to 9.0068 V

Source limits for current are calculated in the same way but using the Model 2651A instrument's current source specifications.

Measurement limits calculations

Measurement limits are calculated in a similar fashion as the source limits, except that the limits are calculated with respect to the measurement of the external reference instrument. For example, suppose that when the Model 2651A is programmed to source 9.0 V the external precision DMM measures 8.9993 V. Using the one-year accuracy specification for the 10 V DC range measurement of ± (0.02% of output + 3 mV offset), the calculated measurement limits are:

Measurement limits = 8.9993 V ± [(8.9993 V × 0.02%) + 3 mV]
Measurement limits = 8.9993 V ± (0.00179986 + 0.003)
Measurement limits = 8.9993 V ± 0.00479986 V
Measurement limits = 8.9945 V to 9.0041 V

Measurement limits for current are calculated in the same way but using the Model 2651A instrument's current measurement specifications.

Restoring factory defaults

Before performing the verification procedures, restore the instrument to its factory front panel (bench) defaults as follows:

1. Press the **MENU** key.
2. Scroll to the **SETUP** menu item (by turning the navigation wheel), 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 verification test procedures

Test summary

- DC voltage output accuracy
- DC voltage measurement accuracy
- DC current output accuracy
- DC current measurement accuracy

If the Model 2651A is not within specifications and not under warranty, see the procedures in Adjustment (on page B-16) for information on adjusting the unit.

Test considerations

When performing the verification procedures:

- Be sure to restore factory front panel defaults as outlined above.
- Make sure that the test equipment is properly warmed up and connected to the Model 2651A output terminals (use 4-wire sensing for voltage).
- Make sure the Model 2651A is set to the correct source range.
- Be sure the Model 2651A output is turned on before making measurements.
- Be sure the test equipment is set up for the proper function and range.
- Allow the Model 2651A output signal to settle before making a measurement.
- Do not connect test equipment to the Model 2651A through a scanner, multiplexer, or other switching equipment.

⚠ WARNING

The maximum common-mode voltage (voltage between LO and chassis ground) is 250V DC. Exceeding this value may cause a break down in insulation, creating a shock hazard.

The Input/Output terminals of the Model 2651A are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the Model 2651A terminals to CAT II, CAT III, or CAT IV circuits. Connection of the Model 2651A terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

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 Model 2651A while the unit is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables connected to the outputs. Putting the equipment into standby mode does not guarantee the outputs are not powered if a hardware or software fault occurs.

Setting the source range and output value

Before testing each verification 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 verified. 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, and then 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.

Current source accuracy

Follow the steps below to verify that the Model 2651A output current accuracy is within specified limits.

1. With the power off, connect the digital multimeter to the Model 2651A output terminals, as shown in Connections for current verification (100 mA range and below) (on page B-7).
2. Select the multimeter DC current measuring function.
3. Press the **SRC** key to source current, and make sure the source output is turned on.
4. Verify output current accuracy for each of the currents for the 100 nA through 100 mA ranges using the values listed in the following table. For each test point:
   - Select the correct source range.
   - Set the Model 2651A 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[1]	Output current limits (1 year, 18° C to 28° C)
100 nA	50.000 nA 90.000 nA	49.450 nA to 50.550 nA 89.410 nA to 90.590 nA
1 µA	500.00 nA 900.00 nA	497.50 nA to 502.50 nA 897.10 nA to 902.90 nA
10 µA	5.0000 µA 9.0000 µA	4.9850 µA to 5.0150 µA 8.9810 µA to 9.0190 µA
100 µA	50.000 µA 90.000 µA	49.925 µA to 50.075 µA 89.913 µA to 90.087 µA
1 mA	0.50000 mA 0.90000 mA	0.49955 mA to 0.50045 mA 0.89943 mA to 0.90057 mA
10 mA	5.0000 mA 9.0000 mA	4.9905 mA to 5.0095 mA 8.9893 mA to 9.0107 mA
100 mA	50.000 mA 90.000 mA	49.955 mA to 50.045 mA 89.943 mA to 90.057 mA
1 A	500.00 mA 900.00 mA	496.10 mA to 503.90 mA 895.78 mA to 904.22 mA
5 A	2.50000 A 4.50000 A	2.4945 A to 2.5055 A 4.4929 A to 4.5071 A
10 A	5.00000 A 9.00000 A	4.9865 A to 5.0135 A 8.9805 A to 9.0195 A

Output current accuracy limits

Source range	Output current setting[1]	Output current limits (1 year, 18° C to 28° C)
20 A	10.0000 A 18.0000 A	9.977 A to 10.023 A 17.965 A to 18.035 A
50 A (pulse)	20.0000 A (DC level) 45.0000 A (pulse)[2]	19.890 A to 20.110 A 44.853 A to 45.148 A
1. Modify current limits if necessary. See Source limits calculations (on page B-3). 2. Verification of a pulsed output (levels greater than 20 A on the 50 A range) is accomplished during current measurement verification test procedure. See Current measurement accuracy (on page B-8).		

5. Repeat the procedure for negative output currents with the same magnitudes as those listed.
6. Turn the output off, and change connections as shown in Connections for current verification (1 A range and above) (on page B-8) (use the 0.1 Ω resistor as shown).
7. Select the DMM DC volts function.
8. Repeat steps 4 through 6 for the 1 A through 20 A ranges, and the 20.0000 A (DC level) output setting for the 50 A (pulse) range. Calculate the current from the DMM voltage reading and the characterized 0.1 Ω resistance value: *I=V/R*

Figure 112: Connections for current verification (100 mA range and below)

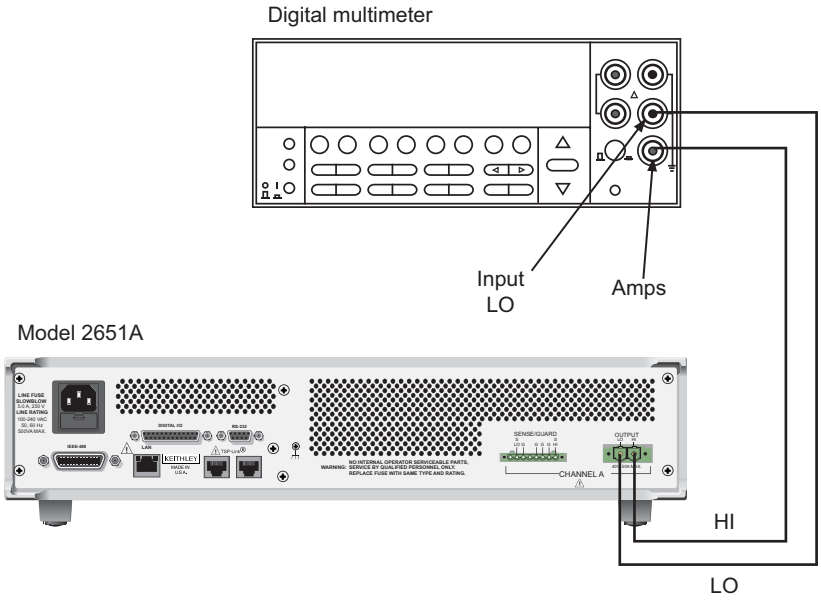
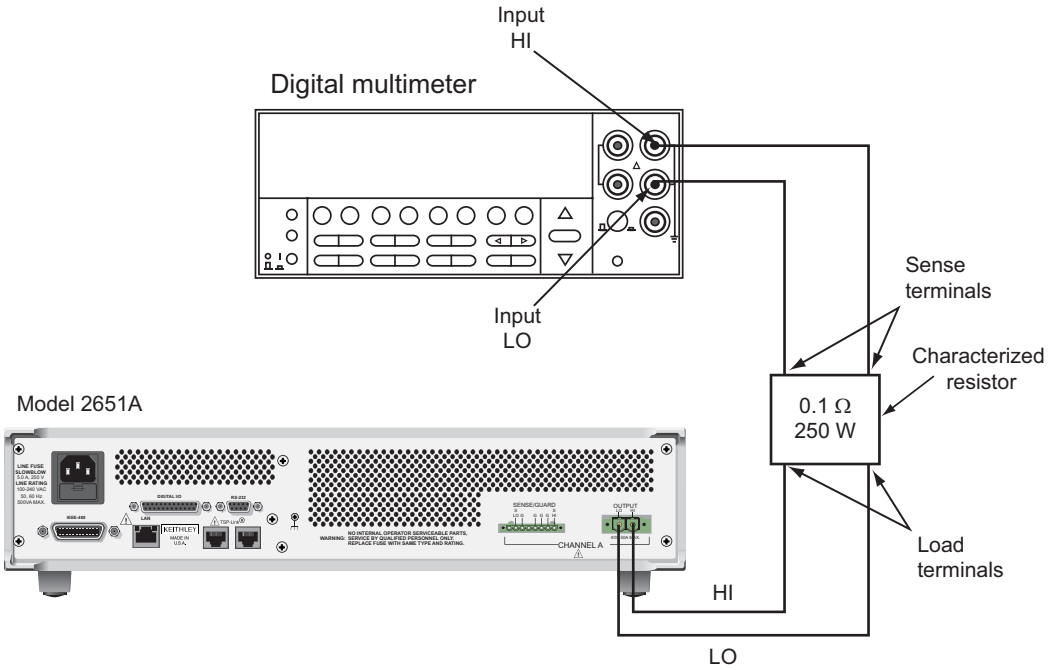


Figure 113: Connections for current verification (1 A range and above)



Current measurement accuracy

Follow the steps below to verify that Model 2651A current measurement accuracy is within specified limits. The procedure involves applying accurate currents from the Model 2651A current source and then verifying that Model 2651A current measurements are within required limits.

1. With the power off, connect the digital multimeter to the Model 2651A terminals as shown in Connections for current verification (100 mA range and below) (on page B-7).
2. Select the multimeter DC current function.
3. Set the Model 2651A to both source and measure current by pressing the **SRC** and then the **MEAS** keys. Make sure the source output is turned on.

4. Verify measure current accuracy for each of the currents listed in the table below for ranges 100 nA through 100 mA.. For each measurement:
   - Select the correct source range.
   - Set the Model 2651A output current such 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 Model 2651A current reading is within the limits given in the table below.
5. 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	50.000 nA 90.000 nA	49.460 nA to 50.540 nA 89.428 nA to 90.572 nA
1 µA	0.5000 µA 0.9000 µA	497.60 nA to 502.40 nA 897.28 nA to 902.72 nA
10 µA	5.0000 µA 9.0000 µA	4.9880 µA to 5.0120 µA 89.848 µA to 9.0152 µA
100 µA	50.000 µA 90.000 µA	49.965 µA to 50.035 µA 89.957 µA to 90.043 µA
1 mA	0.5000 mA 0.9000 mA	499.70 µA to 500.30 µA 899.62 µA to 900.38 µA
10 mA	5.0000 mA 9.0000 mA	4.9965 mA to 5.0035 mA 8.9957 mA to 9.0043 mA
100 mA	50.000 mA 90.000 mA	49.970 mA to 50.030 mA 89.962 mA to 90.038 mA
1 A	0.50000 A 0.90000 A	496.75 mA to 503.25 mA 896.55 mA to 903.45 mA
5 A	2.5000 A 4.5000 A	2.4958 A to 2.5043 A 4.4948 A to 4.5053 A
10 A	5.0000 A 9.0000 A	4.9880 A to 5.0120 A 8.9832 A to 9.0168 A
20 A	10.0000 A 18.0000 A	9.984 A to 10.016 A 17.978 A to 18.022 A
50 A[3]	20.000 A (DC level) 45.000 A (pulse)	19.940 A to 20.060 A 44.888 A to 45.113 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 limits calculations (on page B-4).
3. The 20 A (DC level) source current uses the integrating ADC, while the 45 A (pulse) source current uses the FAST ADC called form the high speed ADC pulse verification script.

6. Turn the output off and change connections as shown in Connections for current verification (1 A range and above) (on page B-8).
7. Select the DMM volts function.
8. Repeat steps 4 through 6 for the 1 A, 5 A, 10 A, and 20 A ranges; and for the 20 A DC level on the 50 A range. Calculate the current from the DMM voltage reading and characterized 0.1 Ω resistance value: *I=V/R*
9. Verification of a pulsed output (levels greater than 20 A on the 50 A range) is accomplished with the same techniques and characterized 0.1 Ω resistor (shunt) as used for other 1 A and above ranges. However, there is an additional requirement that the external DMM must be capable of synchronizing to and measuring the short duration shunt voltage pulse with the required precision.
10. The -45 A high speed ADC pulse verification script (on page B-11) provides an example for setting up a single 1 ms pulse on the 50 A range and measuring the last 500 µs of the pulse using the Model 2651A instrument's fast ADC. The script also sets up a trigger signal that will appear on the rear panel digital I/O connector and be synchronized to the start of the fast ADC measurement (500 µs after the start of the pulse). The external DMM must be able to complete its measurement of the shunt voltage within 500 µs of receiving this trigger signal.

-45 A high speed ADC pulse verification script

As shown, the script generates a single -45 A pulse. To generate a different current level, edit the line:

```
PulseLevel = -45
```

Refer to Instrument programming (on page 6-1) for information on loading and running scripts. Refer to Functions and features (on page 3-1) for detailed information about generating pulses and triggers.

```
-- '-45 A high speed ADC pulse verification script'--
-- Set up SMU idle state ranges and limits
smua.reset()
smua.source.highc = smua.DISABLE
smua.source.delay = 0
smua.source.func = smua.OUTPUT_DCAMPS
smua.source.rangei = 50
smua.measure.rangei = 50
smua.measure.rangev = 10
smua.source.limitv = 10

-- Create a variable for pulsed output value
PulseLevel = -45
-- Note: Make sure that the source idle value has the same polarity
--       as the pulse value to avoid a ~100 uS polarity change
--       time delay during the pulse.
smua.source.leveli = .000001 * PulseLevel

-- Set up single 1 mS pulse using timer[1]
PulseTimer = trigger.timer[1]
PulseTimer.delay = .001
PulseTimer.stimulus = smua.trigger.ARMED_EVENT_ID
PulseTimer.count = 1

-- Set up source pulse parameters
smua.trigger.endpulse.stimulus = PulseTimer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.source.limitv = 10
smua.trigger.source.stimulus = smua.trigger.ARMED_EVENT_ID
smua.trigger.source.action = smua.ENABLE
-- Use a single item list to program the pulse level
smua.trigger.source.listi({PulseLevel})

-- Set up measurement trigger 500 uS into the 1 mS pulse
MeasStartTimer = trigger.timer[2]
MeasStartTimer.delay = .0005
MeasStartTimer.stimulus = smua.trigger.ARMED_EVENT_ID
MeasStartTimer.count = 1

-- Set up Fast ADC to take 500 measurements at 1 uS interval
smua.measure.autozero = smua.AUTOZERO_OFF
smua.measure.filter.enable = smua.FILTER_OFF
smua.measure.adc = smua.ADC_FAST
```

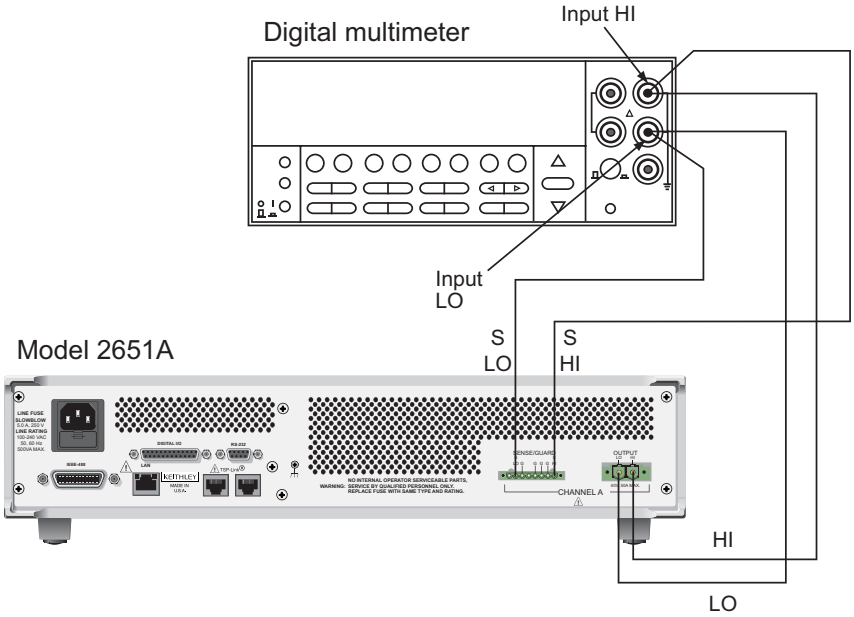
```
smua.trigger.measure.action = smua.ASYNC
smua.trigger.measure.stimulus = MeasStartTimer.EVENT_ID
smua.measure.delay = 0
smua.measure.count = 500
smua.measure.interval = 0.000001

-- Use dedicated buffers to receive measurements
smua.nvbuffer1.clear()
smua.nvbuffer2.clear()
-- Use nvbuffer1 for current and nvbuffer2 for volts
smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2)

-- Generate an external trigger signal on digital I/O connector
--  pin 8 (10 uS falling edge pulse in this example) synchronized
--  to the start of the Fast ADC measurement for use by
--  an external measurement instrument.
digio.trigger[8].mode = digio.TRIG_FALLING
digio.trigger[8].pulsewidth = 10e-6
digio.trigger[8].stimulus = MeasStartTimer.EVENT_ID

-- Set up trigger model state machine for one pulse
--  when smua.trigger.arm.set() command is sent.
smua.trigger.arm.stimulus = trigger.EVENT_ID
smua.trigger.arm.count = 1
smua.trigger.count = 1

-- Enable output (initially at idle level)
smua.source.output = smua.OUTPUT_ON

-- Enter trigger model state machine
smua.trigger.initiate()

-- Start pulse
smua.trigger.arm.set()
waitcomplete()

-- Pulse is done - output can be turned off
smua.source.output = smua.OUTPUT_OFF

-- Retrieve all 500 individual measurements as comma-separated list
--  (parameters can be adjusted to retrieve a subset).
format.data = format.ASCII
printbuffer(1, 500, smua.nvbuffer1) -- current measurements.
printbuffer(1, 500, smua.nvbuffer2) -- voltage measurements.

-- Use getstats functions to retrieve aggregate information
istats = smua.buffer.getstats(smua.nvbuffer1)
vstats = smua.buffer.getstats(smua.nvbuffer2)
print(istats.mean) -- average of current measure
print(vstats.mean) -- average of voltage measure
```

Voltage source accuracy

Follow the steps below to verify that the Model 2651A output voltage accuracy is within specified limits. This test involves setting the output voltage to each full-range value and measuring the voltages with a precision digital multimeter.

1. With the power off, connect the digital multimeter (DMM) to the Model 2651A output terminals using 4-wire connections, as shown below.

Figure 114: Connections for voltage verification



2. Select the multimeter DC volts measuring function.
3. Press the **SRC** key to source voltage and make sure the source output is turned on.
4. Enable the Model 2651A 4-wire (remote sense) mode by pressing the **CONFIG** key and then the **SRC** key.
5. 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 Model 2651A output voltage to the indicated value.
   - Verify that the multimeter reading is within the limits given in the table.

   Repeat the procedure for negative output voltages with the same magnitudes as those listed in the following table, as applicable.

Output voltage accuracy limits

Source range	Output voltage setting*	Output voltage limits (1 year, 18° C to 28° C)
100 mV	50.000 mV 90.000 mV	49.490 mV to 50.510 mV 89.482 mV to 90.518 mV
1 V	0.50000 V 0.90000 V	0.49940 V to 0.50060 V 0.89932 V to 0.90068 V
10 V	5.0000 V 9.0000 V	4.9940 V to 5.0060 V 8.9932 V to 9.0068 V
20 V	10.000 V 18.000 V	9.993 V to 10.007 V 17.991 V to 18.009 V
40 V	20.000 V 36.000 V	19.984 V to 20.016 V 35.981 V to 36.019 V
* Modify voltage limits if necessary. See Source limits calculations (on page B-3).		

Voltage measurement accuracy

Follow the steps below to verify that the Model 2651A voltage measurement accuracy is within specified limits. The test involves setting the source voltage, as measured by a precision digital multimeter, and then verifying that the Model 2651A voltage readings are within required limits.

1. With the power off, connect the digital multimeter to the Model 2651A output terminals using 4-wire connections, (the same Connections for voltage verification (on page B-13) as shown in Voltage source accuracy (on page B-13)).
2. Select the multimeter DC volts function.
3. Enable the Model 2651A 4-wire (remote sense) mode by pressing **CONFIG** then **MEAS,** then select **V-MEAS** > **SENSE-MODE** > **4-WIRE**.
4. Set the Model 2651A to both source and measure voltage by pressing the **SRC** and then the **MEAS** keys.

5. Make sure the source output is turned on (if off, press the **OUTPUT ON/OFF** control).
6. 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 Model 2651A 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.
   - If necessary, press the **TRIG** key to display readings.
   - Verify that the Model 2651A voltage reading is within the limits given in the table.

   Voltage measurement accuracy limits

Source and measure range[1]	Source voltage[2]	Voltage reading limits (1 year, 18° C to 28° C)
100 mV	50.000 mV 90.000 mV	49.690 mV to 50.310 mV 89.682 mV to 90.318 mV
1 V	0.50000 V 0.90000 V	0.49960 V to 0.50040 V 0.89952 V to 0.90048 V
10 V	5.0000 V 9.0000 V	4.9960 V to 5.0040 V 8.9952 V to 9.0048 V
20 V	10.000 V 18.000 V	9.993 V to 10.007 V 17.991 V to 18.009 V
40 V	20.000 V 36.000 V	19.984 V to 20.016 V 35.981 V to 36.019 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 accordingly if necessary. See Measurement limits calculations (on page B-4).		

7. Repeat the procedure for negative source voltages with the same magnitudes as those listed in the table (see above).
8. Perform high speed verification procedure for the High Speed ADC Volts measurements:
   a. Change to using the high-speed ADC by pressing the **SPEED** key and then selecting FAST from the menu.
   b. Repeat the procedure (step 6) for each of the voltages listed in the following table.
   c. Repeat the procedure for negative source voltages with the same magnitudes as those listed in the following table.
   d. Change back to the integrating ADC by pressing the **SPEED** key and then selecting NORMAL from the menu.

   High Speed ADC Volts measurement accuracy limits

Source and measure range[1]	Source voltage[2]	Voltage reading limits (1 year, 18° C to 28° C)
1 V	0.50000 V 0.90000 V	0.49915 V to 0.50085 V 0.89895 V to 0.90105 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 accordingly if necessary. See Measurement limits calculations (on page B-4).		

Adjustment

Introduction

Use the procedures in this section to calibrate the Keithley Instruments Model 2651A High Power System SourceMeter® Instrument. These procedures require accurate test equipment to measure precise DC voltages and currents.

⚠ CAUTION

The information in this section is intended for qualified service personnel only. Do not attempt these procedures unless you are qualified to do so.

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 calibration procedures at an ambient temperature of 18° C to 28° C (65° F to 82° F), with relative humidity of less than 70 percent (unless otherwise noted).

NOTE

Product specifications that are listed as 18º C to 28º C assume adjustment has been done at 23º C. If the Model 2651A is adjusted at a different temperature, the specifications apply to +/-5º C of that adjustment temperature.

Line power

The Model 2651A requires a line voltage of 100 V to 240 V at line frequency of 50 Hz or 60 Hz. The instrument must be calibrated within this range.

Warm-up period

Allow the Model 2651A to warm up for at least two hours before performing calibration.

If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the instrument's internal temperature to stabilize. Typically, allow one extra hour to stabilize a unit that is 10°C (18°F) outside the specified temperature range.

Also, allow the test equipment to warm up for the minimum time specified by the manufacturer.

Calibration considerations

When performing the calibration procedures:

- Make sure that the test equipment is properly warmed up and connected to the correct Model 2651A terminals.
- Always allow the source signal to settle before calibrating each point.
- Do not connect test equipment to the Model 2651A through a scanner or other switching equipment.
- If an error occurs during calibration, the Model 2651A will generate an appropriate error message. See Error summary (on page 8-1) for more information.

⚠ WARNING

The maximum common-mode voltage (voltage between LO and chassis ground) is 250V DC. Exceeding this value may cause a breakdown in insulation, creating a shock hazard that could result in personal injury or death.

The Input/Output terminals of the Model 2651As are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the Model 2651A terminals to CAT II, CAT III, or CAT IV circuits. Connection of the Model 2651A terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

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 Model 2651A while the unit is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables connected to the outputs. Putting the equipment into standby mode does not guarantee the outputs are not powered if a hardware or software fault occurs.

Calibration cycle

Perform calibration at least once a year to ensure the unit meets or exceeds its specifications.

Recommended calibration equipment

The table below contains the recommended equipment for the calibration 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 Model 2651A specifications.

Description	Manufacturer/model	Accuracy	
Digital multimeter		DC voltage	50 mV: ±1500 ppm 90 mV: ±880 ppm 0.5 V: ±200 ppm 0.9 V: ±130 ppm 5 V: ±200 ppm 9 V: ±130 ppm 10 V: ±170 ppm 18 V: ±110 ppm 20 V: ±200 ppm 36 V: ±130 ppm
		DC current	50 nA: ±2700 ppm 90 nA: ±1500 ppm 500 nA: ±1200 ppm 900 nA: ±750 ppm 5 µA: ±600 ppm 9 µA: ±420 ppm 50 µA: ±170 ppm 90 µA: ±110 ppm 0.5 mA: ±150 ppm 0.9 mA: ±100 ppm 5 mA: ±170 ppm 9 mA: ±110 ppm 50 mA: ±150 ppm 90 mA: ±100 ppm 500 mA: ±1600 ppm 900 mA: ±950 ppm 2.5 A: ±420 ppm 4.5 A: ±290 ppm 5 A: ±600 ppm 9 A: ±460 ppm 10 A: ±400 ppm 18 A: ±310 ppm
		DC current Pulse only	20 A: ±750 ppm 45 A: ±620 ppm
Precision resistor: 0.1 Ω, 250 W, 0.1%	Isotek RUG-Z-R100-0.1-TK1	Resistance*	0.1 Ω: ±75 ppm
50 Ω resistors (2)	Any suitable.**		

* Resistor used to test 1 A, 5 A, 10 A, and 20 A ranges. Before use, characterize the resistor to the uncertainty shown.
** Used for contact check calibration. Before use, characterize with the resistance function of the digital multimeter.

Calibration overview

The following topics contain an overview of the entire calibration procedure.

Parameter values

The full-scale parameters are actually 90% of full-scale as indicated in the step sequence table. Note that you cannot send a value of exactly 0 for the two zero parameters, but must instead send a very small value, for example 1e-30 or -1e-30.

Sense modes

The table titled "Model 2651A calibration steps" in Model 2651A step sequence (on page B-19) 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 1 V through 40 V ranges are calibrated using the REMOTE sense mode, and the 1 V, 1 mA, and 5 A source ranges are calibrated using the CALA sense mode.

Model 2651A step sequence

Calibration steps must be performed in a specific sequence. See the following table. Note that all steps are performed using 2-wire (local sensing) except as noted. Calibration of each range is performed as a four-point calibration:

- + ZERO
- + FULL SCALE
- - ZERO
- - FULL SCALE

NOTE

Before performing the calibration steps, refer to Parameter values (on page B-19) and Sense modes (on page B-19).

Model 2651A calibration steps

Function[1]	Calibration steps[2]	Calibration points[3]	Sense mode[4]
Voltage source and measure See Step 2. Voltage calibration adjustment (on page B-23)	100 mV	±1e-30, ±90 mV	smua.SENSE_LOCAL
	100 mV	±1e-30, ±90 mV	smua.SENSE_REMOTE
	1 V	±1e-30, ±0.9 V	smua.SENSE_LOCAL
	1 V	±1e-30, ±0.9 V	smua.SENSE_CALA
	1 V	±1e-30, ±0.9 V	smua.SENSE_REMOTE
	10 V	±1e-30, ±9 V	smua.SENSE_LOCAL
	10 V	±1e-30, ±9 V	smua.SENSE_REMOTE
	20 V	±1e-30, ±18 V	smua.SENSE_LOCAL
	20 V	±1e-30, ±18 V	smua.SENSE_REMOTE
	40 V	±1e-30, ±36 V	smua.SENSE_LOCAL
	40 V	±1e-30, ±36 V	smua.SENSE_REMOTE
Current source and measure See Step 3. Current calibration adjustment (on page B-26)	100 nA	±1e-30, ±90 nA	smua.SENSE_LOCAL
	1 µA	±1e-30, ±0.9 µA	smua.SENSE_LOCAL
	10 µA	±1e-30, ±9 µA	smua.SENSE_LOCAL
	100 µA	±1e-30, ±90 µA	smua.SENSE_LOCAL
	1 mA	±1e-30, ±0.9 mA	smua.SENSE_LOCAL
	1 mA	±1e-30, ±0.9 mA	smua.SENSE_CALA
	10 mA	±1e-30, ±9 mA	smua.SENSE_LOCAL
	100 mA	±1e-30, ±90 mA	smua.SENSE_LOCAL
	1 A	±1e-30, ±0.9 A	smua.SENSE_LOCAL
	5 A	±1e-30, ±4.5 A	smua.SENSE_LOCAL
	5 A	±1e-30, ±4.5 A	smua.SENSE_CALA
	10 A	±1e-30, ±9 A	smua.SENSE_LOCAL
	20 A	±1e-30, ±18 A	smua.SENSE_LOCAL
	50 A	±1e-30, ±20 A[5]	smua.SENSE_LOCAL
	100 A[6]	±1e-30, ±20 A[5]	smua.SENSE_LOCAL
Fast ADC	See Step 4. Fast ADC calibration adjustment (on page B-30)		
Contact check	See Step 5. Contact check calibration adjustment (on page B-31)		

1. Calibrate only the source for the SENSE_CALA sense steps and for the 100 A SENSE_LOCAL steps.
2. Steps must be performed in the order shown.
3. Do not use actual 0 values for zero calibration points. Use very small values such as ±1e-30. Calibration polarities must also be set as shown in the procedures.
4. Output must be off before changing to the CALA sense mode.
5. The maximum DC level that can be used to calibrate 50 A and 100 A ranges is 20 A.
6. The 100 A current source range adjust improves dynamic (pulse) operation of the 50 A range. "`smua.source.rangei = 100`" is valid only when calibration is unlocked.

Calibration commands quick reference

The following table summarizes remote calibration commands. For a more complete description of these commands, refer to Programming commands.

Calibration commands	
Command	**Description**
`smua.cal.adjustdate = caldate`	Set calibration adjustment date for Model 2651A.
`smua.cal.date = caldate`	Set calibration date (caldate of 0 indicates date not set).
`smua.cal.due = caldue`	Set calibration due date (caldue of 0 indicates date not set).
`smua.cal.fastadc()`	Performs calibration of the fast analog-to-digital converter (ADC).
`smua.cal.lock()`	Lock out calibration.
`smua.cal.password = "newpassword"`	Change password to "newpassword".
`smua.cal.polarity = polarity`	Set polarity: smua.CAL_AUTO (automatic polarity). smua.CAL_NEGATIVE (negative polarity). smua.CAL_POSITIVE (positive polarity).
`smua.cal.restore(calset)`	Load calibration set of constants: smua.CALSET_NOMINAL (nominal constants). smua.CALSET_FACTORY (factory constants). smua.CALSET_DEFAULT (normal constants). smua.CALSET_PREVIOUS (previous constants).
`smua.cal.save()`	Store constants in nonvolatile memory as DEFAULT calibration set .
`calstate = smua.cal.state`	Request calibration state: smua.CALSTATE_CALIBRATING smua.CALSTATE_LOCKED smua.CALSTATE_UNLOCKED
`smua.cal.unlock("password")`	Unlock calibration (default password: KI0026XX)
`smua.measure.calibratei(range, cp1measured, cp1reference, cp2measured, cp2reference)`	Calibrate current measure range*: ±range (measurement range to calibrate). cp1measured (Model 2651A measured value for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (Model 2651A measured value for cal. point 2). cp2reference (reference measurement for cal. point 2).

Calibration commands	
Command	**Description**
`smua.measure.calibratev(range, cp1measured, cp1reference, cp2measured, cp2reference)`	Calibrate voltage measure range*: ±range (measurement range to calibrate). cp1measured (Model 2651A measured value for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (Model 2651A measured value for cal. point 2). cp2reference (reference measurement for cal. point 2).
`smua.source.calibratei(range, cp1expected, cp1reference, cp2expected, cp2reference)`	Calibrate current source range*: ±range (range to calibrate). cp1expected (source value programmed for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2expected (source value programmed for cal. point 2). cp2reference (reference measurement for cal. point 2).
`smua.source.calibratev(range, cp1expected, cp1reference, cp2expected, cp2reference)`	Calibrate voltage source range*: ±range (range to calibrate). cp1expected (source value programmed for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2expected (source value programmed for cal. point 2). cp2reference (reference measurement for cal. point 2)
`smua.contact.calibratelo( cp1measured, cp1reference, cp2measured, cp2reference)`	Calibrate the low/sense low contact check measurement. cp1measured (value measured by SMU for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (value measured by SMU for cal. point 2). cp2reference (reference measurement for cal. point 2).
`smua.contact.calibratehi( cp1measured, cp1reference, cp2measured, cp2reference)`	Calibrate the high/sense high contact check measurement. cp1measured (value measured by SMU for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (value measured by SMU for cal. point 2). cp2reference (reference measurement for cal. 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 Model 2651A step sequence (on page B-19) for calibration points.	

Calibration adjustment procedure

Use the following procedure to perform remote calibration by sending commands over the IEEE-488 bus, RS-232 port, or LAN. The remote commands and appropriate parameters are separately summarized for each step.

Step 1. Prepare the Model 2651A for calibration

A. Connect the Model 2651A to the controller IEEE-488 interface, RS-232 port, or LAN using an appropriate interface cable.

B. Turn on the Model 2651A and the test equipment, and allow them to warm up for at least two hours before performing calibration.

C. Make sure the IEEE-488, RS-232, or LAN interface parameters are set up properly (to configure the interface, press the **MENU** key, and then select **RS232**, **LAN**, or **GPIB**, as applicable).

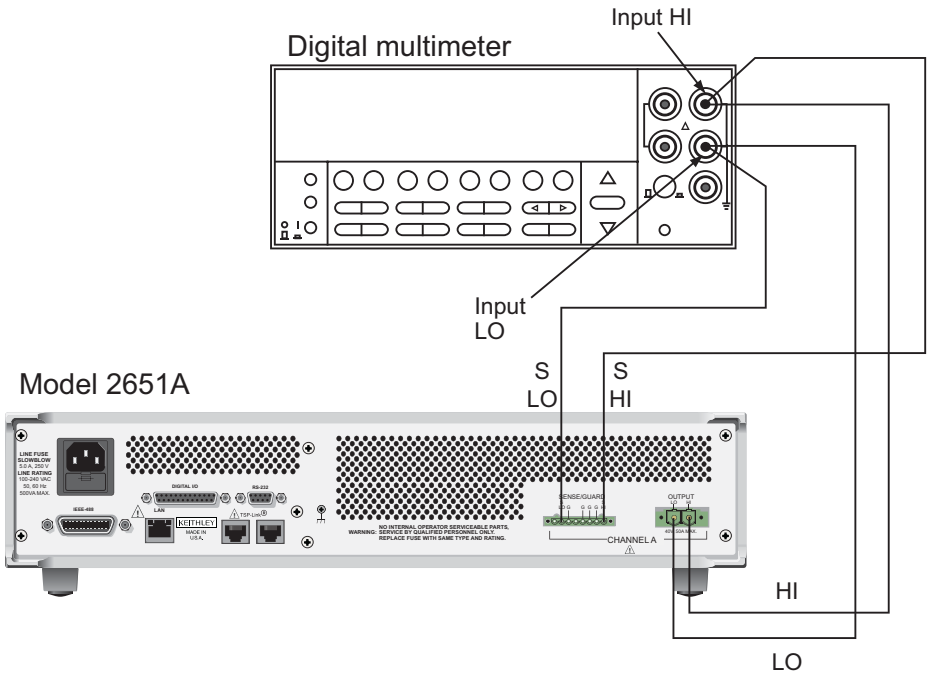
Step 2. Voltage calibration adjustment

A. Connect the Model 2651A to the digital multimeter using the 4-wire connections shown in the figure below, and select the multimeter DC volts function.

B. Send the following commands in order to initialize voltage calibration:

```
smua.cal.unlock("KI0026XX")
smua.reset()
smua.source.func = smua.OUTPUT_DCVOLTS
```

Figure 115: Connections for voltage calibration



C. Perform each calibration step listed for the voltage source and measure function in the Model 2651A step sequence (on page B-19) as follows:

1. Select the range being calibrated with this command:

```
smua.source.rangev = range
```

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.

2. Select the correct sense mode based on the calibration step for the voltage source and measure function from the Model 2651A step sequence (on page B-19), 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 Model 2651A voltage readings at the positive zero value (the Model 2651A 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 Model 2651A:

   ```
   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 Model 2651A voltage readings at the positive full-scale output value (the Model 2651A 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 Model 2651A:

   ```
   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:

range	= The present calibration range
src_Z	= The +zero Model 2651A source output value
DMM_Z_rdg	= The +zero DMM measurement
src_FS	= The +FS Model 2651A source output value
DMM_FS_rdg	= The +FS DMM measurement

    Typical values for the Model 2651A 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 Model 2651A 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 Model 2651A measurement
DMM_Z_rdg	= The +zero DMM measurement
FS_rdg	= The +FS Model 2651A measurement
DMM_FS_rdg	= The +FS DMM measurement

Typical Model 2651A 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 Model 2651A voltage readings at the negative zero value (the Model 2651A 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 Model 2651A:

```
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 Model 2651A voltage readings at the negative full-scale output value (the Model 2651A 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 Model 2651A:

```
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 Model 2651A source output value
DMM_Z_rdg	= The zero DMM measurement
src_FS	= The FS Model 2651A source output value
DMM_FS_rdg	= The FS DMM measurement

Typical values for the Model 2651A 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 Model 2651A 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 Model 2651A measurement
DMM_Z_rdg	= The zero DMM measurement
FS_rdg	= The FS Model 2651A measurement
DMM_FS_rdg	= The FS DMM measurement

    Typical Model 2651A 1 V range values:

    ```
    smua.measure.calibratev(-1, -1e-4, -1e-6, -0.89, -0.896)
    ```

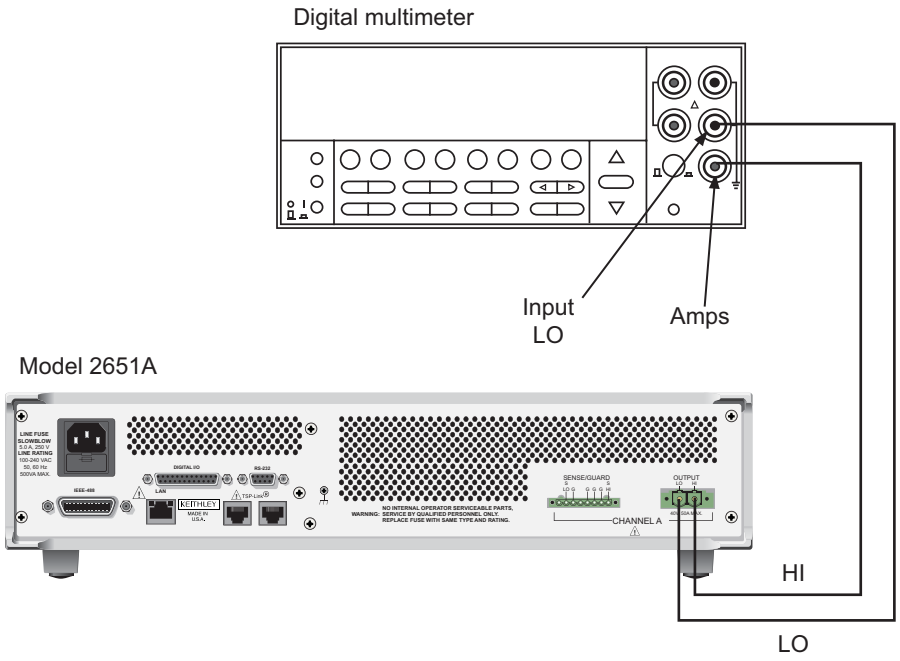
D. Be sure to complete each of the 22 steps of C for all 11 voltage source and measure function steps in the Model 2651A step sequence (on page B-19) before continuing to the current calibration.

E. Select automatic polarity mode:

```
smua.cal.polarity = smua.CAL_AUTO
```

Step 3. Current calibration adjustment

Figure 116: Connections for current calibration (100 mA range and below)



A. Connect the Model 2651A to the digital multimeter (see the figure above), and then select the multimeter DC current function.

B. Send this command to initialize current calibration:

```
smua.source.func = smua.OUTPUT_DCAMPS
```

C. Perform each calibration step for the current source and measure function listed in Model 2651A step sequence (on page B-19) for the 100 mA range and below as follows:

1. Select the range being calibrated:

```
smua.source.rangei = range
```

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.

2. Select the correct sense mode based on the calibration step for the current source and measure function listed in Model 2651A step sequence (on page B-19), for example:

```
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.leveli = 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 Model 2651A current readings at the positive zero value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

```
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.leveli = 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 Model 2651A voltage readings at the positive full-scale output value (the Model 2651A measurement is not necessary if calibration is being done on the CALA sense mode or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

```
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:

range	= The present calibration range
src_Z	= The +zero Model 2651A source output value
DMM_Z_rdg	= The +zero DMM measurement
src_FS	= The +FS Model 2651A 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-5, 90e-3, 88e-3)
```

12. If this step is not on the CALA sense mode or 100 A range, send the measure calibration command using the multimeter and Model 2651A 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 Model 2651A measurement
DMM_Z_rdg	= The +zero DMM measurement
FS_rdg	= +FS Model 2651A 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.leveli = -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 Model 2651A current readings at the negative zero value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

```
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.leveli = -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 Model 2651A current readings at the negative full-scale output value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

```
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 Model 2651A source output value
DMM_Z_rdg	= The zero DMM measurement
src_FS	= The FS Model 2651A 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 or 100 A range, send the measure calibration command using the multimeter and Model 2651A 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 Model 2651A measurement
DMM_Z_rdg	= The zero DMM measurement
FS_rdg	= The FS Model 2651A measurement
DMM_FS_rdg	= The FS DMM measurement

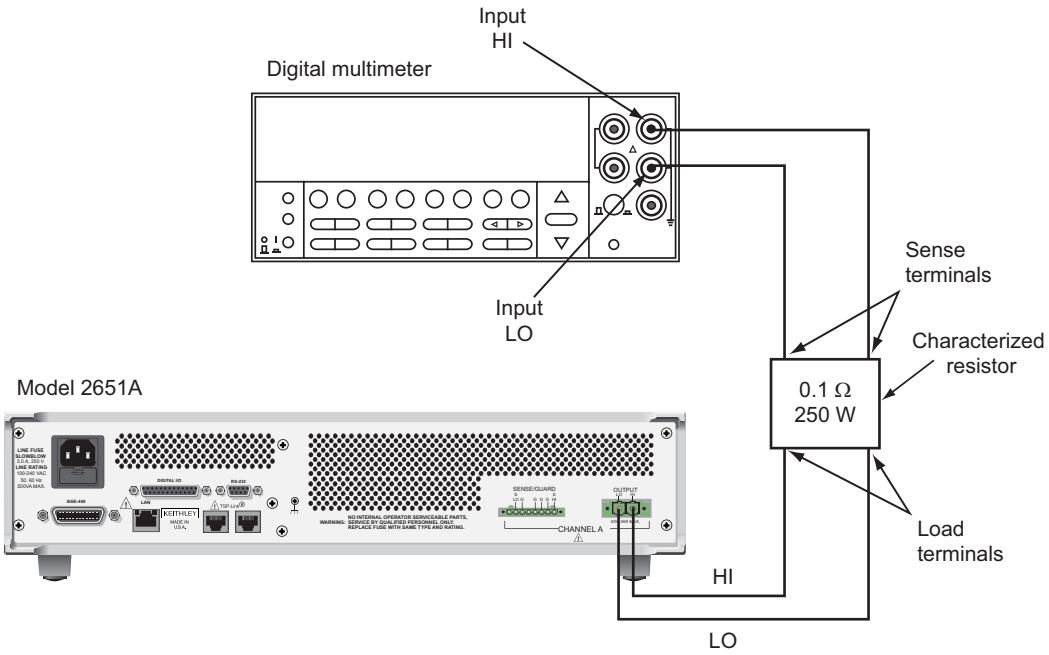
Typical 100 mA range values:

```
smua.measure.calibratei(-100e-3, -1e-5, -1e-6, -91e-3, -89.2e-3)
```

D. Before continuing, be sure to complete each of the 22 steps of C for all the source and measure ranges 100 mA and below for the current source and measure function listed in the Model 2651A step sequence (on page B-19).

E. Change connections as shown in the figure below (use 4-wire connections to the 0.1 Ω resistor as shown).

Figure 117: Connections for current calibration (1 A range and above)



F. Select the DMM DC volts function.

G. Repeat the 22 steps of C for the 1 A, 5 A, 10 A, 20 A, 50 A, and 100 A ranges. Compute the current reading from the DMM voltage reading and characterized 0.1 Ω resistance value: I = V/R.

H. Select automatic polarity mode:

```
smua.cal.polarity = smua.CAL_AUTO
```

Step 4. Fast ADC calibration adjustment

Perform fast analog-to-digital converter (ADC) calibration adjustment:

```
smua.cal.fastadc()
```

NOTE

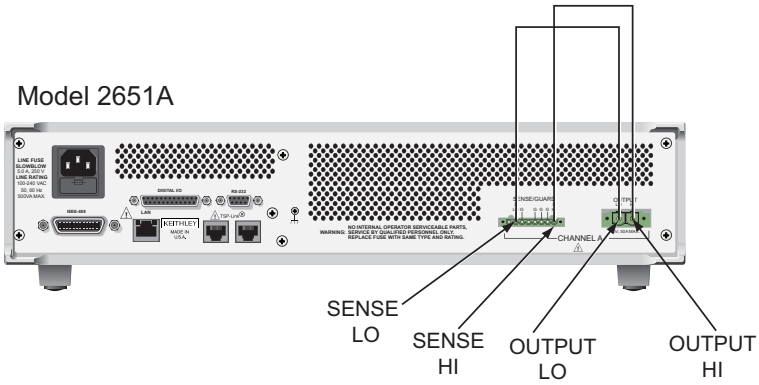
Make sure both the voltage and current calibration steps have been completed before calling the `smua.cal.fastadc()` function.

Step 5. Contact check calibration adjustment

A. As illustrated in the following figure:

- Short the Model 2651A SENSE LO and OUTPUT LO terminals together.
- Short the SENSE HI and OUTPUT HI terminals together.

Figure 118: Connections for contact check 0 OHM calibration



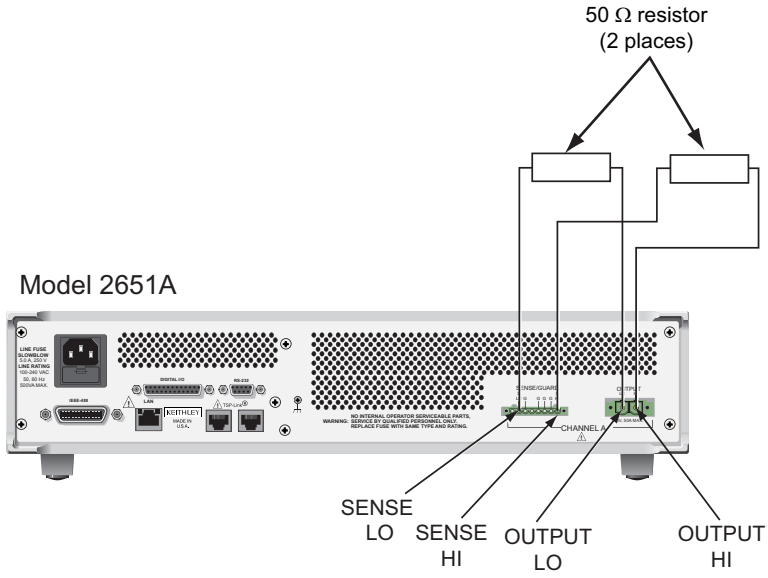
B. Allow the readings to settle, then get the Model 2651A readings:

```
r0_hi, r0_lo = smua.contact.r()
```

C. As illustrated in the following figure:

- Characterize both 50 Ω resistors using the resistance function of the digital multimeter.
- Connect a 50 Ω resistor between the SENSE LO and OUTPUT LO terminals.
- Connect the second 50 Ω resistor between the SENSE HI and OUTPUT HI terminals.

Figure 119: Connections for contact check 50 ohm calibration



D. Allow the readings to settle, then get the Model 2651A readings:

```
r50_hi, r50_lo = smua.contact.r()
```

E. Send the contact check low calibration command:

```
smua.contact.calibratelo(r0_lo, Z_actual, r50_lo, 50_ohm_actual)
```

Where:

r0_lo	=	Model 2651A 0 Ω low measurement
Z_actual	=	Actual zero value; the resistance of the short between the SENSE LO and OUTPUT LO terminals
r50_lo	=	Model 2651A 50 Ω low measurement
50_ohm_actual	=	Actual 50 Ω resistor value; the actual value of the resistor between the SENSE LO and OUTPUT 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 D.

F. Send the contact check high calibration command:

```
smua.contact.calibratehi(r0_hi, Z_actual, r50_hi, 50_ohm_actual)
```

Where:

r0_hi	=	Model 2651A 0 Ω high measurement
Z_actual	=	Actual zero value; the resistance of the short between the SENSE HI anD OUTPUT HI terminals
r50_hi	=	Model 2651A 50 Ω high measurement
50_ohm_actual	=	Actual 50 Ω resistor value; the value of the resistor between the SENSE HI and OUTPUT 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 D.

Step 6. Program calibration dates

Use the following command to set the calibration adjustment date:

```
smua.cal.adjustdate = os.time{year=2010, 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=2010, month=12, day=1}

smua.cal.due = os.time{year=2011, 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, and day, as well as (optional) hour, and minute should be used (seconds can be given but will essentially be ignored due to the precision of the internal date storage format). The allowable range for the year is from 2005 to 2037, the month is from 1 to 12, and the day is from 1 to 31.

Step 7. Save calibration constants

Calibration 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 you just performed will be temporary.

Step 8. Lock out calibration

To lock out further calibration, send the following command after completing the calibration procedure:

```
smua.cal.lock()
```

Appendix C

LAN concepts and settings

In this appendix:

Overview C-1
Establishing a point-to-point connection C-1
Connecting to the LAN C-8
LAN speeds C-11
Duplex mode C-11
Viewing LAN status messages C-12
Viewing the network settings C-13
Selecting a remote command interface C-14
Logging LAN trigger events in the event log C-19

Overview

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument is class C LXI version 1.2 compliant. The Model 2651A is a scalable test system that can connect directly to a host PC or interact with a DHCP or DNS server and other LXI-compliant instruments on a local area network (LAN).

The Model 2651A is compliant with the IEEE Std 802.3 and supports full connectivity on a 10 Mbps or 100 Mpbs network. The LAN interface is an alternative solution to GPIB that can be used to build test systems with a large degree of flexibility, including web accessibility.

NOTE

Please read this entire section before you connect the Model 2651A to the LAN.

Establishing a point-to-point connection

A one-to-one LAN connection to set up a static IP address between the host computer and the instrument enables the use of the instrument's internal web page and TSP® Express.

Use the following instructions to configure the instrument's IP address. The instrument's IP address 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

Contact your corporate information technology (IT) department for permission before you connect the Model 2651A to a corporate network.

If you have problems, see LAN troubleshooting suggestions (on page 8-6).

⚠ CAUTION

Record all network configurations before modifying any existing network configuration information on the network interface card. Once the network configuration settings are updated, the older 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 damage to the equipment and loss of data. Contact your system administrator for more information.

Step 1: Identify and record the existing IP configuration

To identify the existing IP configuration:

1. Open a command prompt window:

   Windows 2000 or Windows XP:

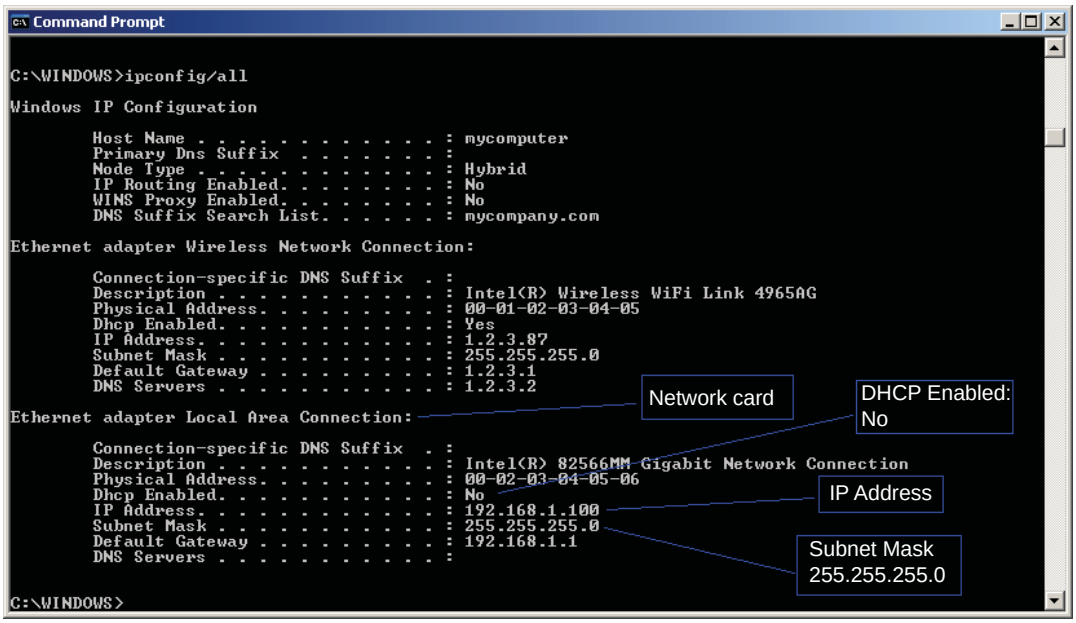
   a. Click **Start** and select **Run**.
   b. In the Open box, type `cmd`.
   c. Click **OK**.

   Windows Vista or Windows 7:

   a. Click **Start**.
   b. Select **All Programs > Accessories > Command Prompt**.

2. At the command prompt, type `ipconfig/all` and press the **Enter** key. A list of existing IP configuration information for your computer is displayed.

Figure 120: Computer IP configuration using the command prompt



NOTE

If the information for the Ethernet adapter displays "Media Disconnected," close the command prompt and go to Step 2: Disable DHCP to use the computer's existing IP address (on page C-4).

3. When the information is displayed, record the following information for the correct network card:
   - DHCP mode: _____
   - IP address: _____
   - Subnet mask: _____
   - Default gateway: _____
   - DNS servers: _____

⚠ CAUTION

The `ipconfig/all` command displays the configuration of every network card. Make sure that you record the information for the proper network card.

4. If:
   - **DHCP Enabled = Yes:** Go to Step 2: Disable DHCP to use the computer's existing IP address (on page C-4)
   - **DHCP Enabled = No:** Go to Step 3: Configure the Instrument's LAN settings (on page C-6).
5. To exit the IP configuration screen, type **exit** at the command prompt and press **Enter**.

Step 2: Disable DHCP to use the computer's existing IP address

NOTE

Do not change the IP address at any time without talking to your system administrator first. Entering an incorrect IP address can prevent your workstation from connecting to your corporate network.

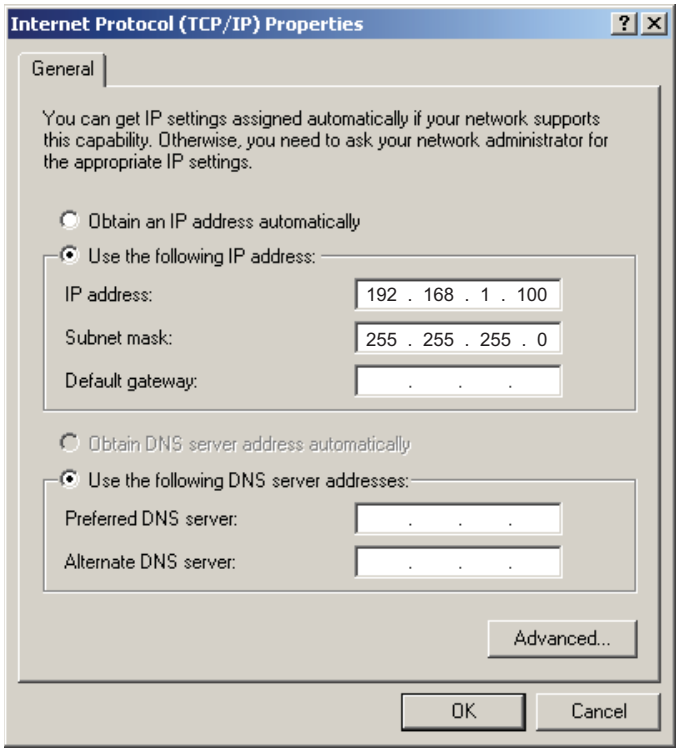
1. Open the Internet Protocol Properties dialog box:

Windows 2000:	**Windows XP:**	**Windows Vista and Windows 7:**
a. Click the **Start > Settings > Control Panel**. b. Open **Network and Dial-up Connections**. c. Right-click **Local AreaConnection** and select **Properties.** The Local Area Connection Properties dialog box is displayed. d. Double-click **Internet Protocol (TCP/IP)** in the items list. The Internet Protocol (TCP/IP) Properties dialog box is displayed (see the figure titled "Internet protocol (TCP/IP) Properties dialog box" below).	a. Click **Start > Settings > Control Panel**. b. Open **Network Connections**. c. Right-click **Local Area Connection** and select **Properties**. The Local Area Connection Properties dialog box is displayed. d. Double-click **Internet Protocol (TCP/IP)** in the items list. The Internet Protocol (TCP/IP) Properties dialog box is displayed (see the figure titled "Internet protocol (TCP/IP) Properties dialog box" below).	a. Click the **Start > Control Panel**. b. Open **Network & Sharing Center**. c. In the list, click **View Status** next to Connection. The Wireless Network Connection Status dialog box is displayed. d. Click **Properties.** Windows displays a permissions message. e. If you are logged in as administrator, click **Continue**. If you are not logged in as administrator, enter the administrator's password to continue. f. The Network Connection Properties dialog box is displayed. g. Double-click **Internet Protocol Version 6 (TCP/IPv6)** in the items list. The Internet Protocol Version 6 (TCP/IPv6) Properties dialog box is displayed (see the figure titled "Internet protocol (TCP/IP) Properties dialog box" below).

2. Select **Use the following IP address**. The option for "Use the following DNS server addresses" is automatically selected.
3. Set the IP address:
   a. Are the IP address and subnet mask fields populated?
      - **Yes:** If populated, record the IP address, subnet mask, default gateway, and DNS servers to use in Step 3: Configure the Instrument's LAN settings (on page C-6).
      - **No:** If blank, enter the IP address 192.168.1.100 in the IP address field and 255.255.255.0 in the subnet mask field. These will be used to configure the instrument's LAN settings.

   b. After recording or entering the IP address, click **OK**.
4. Click **OK** to close the Local Area Connection Properties dialog box.
5. Close the Network Connections window.

Figure 121: Internet protocol (TCP/IP) Properties dialog box



Step 3: Configure the instrument's LAN settings

To configure the Model 2651A using the front panel:

1. Press the **MENU** key to display the MAIN MENU. Use the navigation wheel ◷ to select **LAN** to display the LAN CONFIG menu.
2. Change the IP address assignment method:
   a. Select **CONFIG > METHOD > MANUAL**, then press the **ENTER** key.
   b. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
   c. Select **APPLY_SETTINGS > YES**, then press the **ENTER** key.
3. Enter the IP address using the LAN CONFIG menu:
   a. Select **CONFIG > IP-ADDRESS**.
   b. Refer to the recorded computer's IP address (Step 1: Identify and record the existing IP configuration (on page C-2)). A portion of the computer's IP address will be used as a base for the instrument's unique ID. Only the last three numbers (after the last decimal point) will be different between the PC and instrument. The last three digits may be anything from 1 to 255 for a subnet mask of 255.255.255.0.

      For example, the Internet Protocol (TCP/IP) Properties dialog box shows that the computer's IP address is 192.168.1.100 (see the figure titled "Internet protocol (TCP/IP) Properties dialog box" in Step 2: Disable DHCP to use the computer's existing IP address (on page C-4)). A unique IP address for the instrument is 192.168.001.101.

NOTE

The instrument's IP address can have leading zeros, but the computer's 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 to use in Step 5: Access the instrument's internal web page (on page C-7).
   d. Push the **ENTER** key or navigation wheel ◷ to confirm the changes.
   e. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
   f. From the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, then press the **ENTER** key.
4. Change the subnet mask from within the LAN CONFIG menu:
   a. Select **CONFIG > SUBNETMASK**, 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 PC settings recorded earlier (or 255.255.255.000 if DHCP Enabled = YES).
   c. Push the **ENTER** key or the navigation wheel ◷ when finished changing all the characters.
   d. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
   e. From the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, then press the **ENTER** key.

NOTE

APPLY_SETTINGS must be used before changes to the IP address or subnet mask are applied.

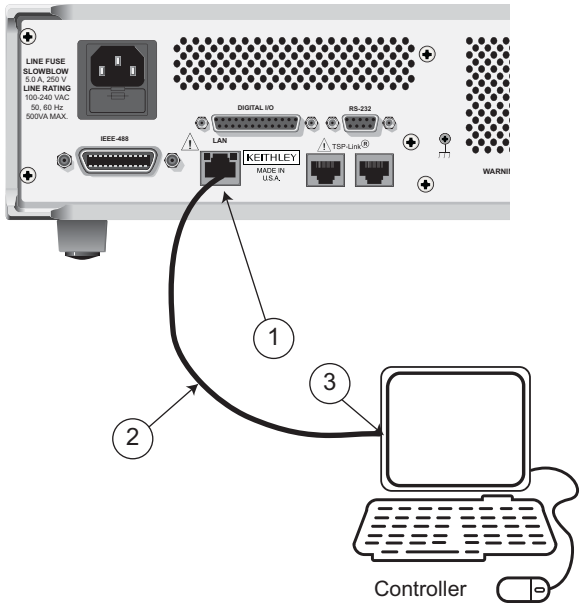
Step 4: Install the crossover cable

Connect the supplied crossover cable between the computer's NIC card and the LAN connector on the instrument's rear panel. There are multiple connectors on the Model 2651A rear panel. Be sure to connect to the LAN connection port (see the following figure).

NOTE

Connect the crossover cable into the same PC LAN port that was used during the configuration of the instrument. This will ensure that the system is using the correct network card.

Figure 122: LAN connection



(1) Model 2651A LAN connection port
(2) Crossover cable
(3) Ethernet port (located on the host PC)

Step 5: Access the instrument's internal web page

1. Open a web browser on the host computer.
2. Enter the instrument's IP address in the browser's address box. For example, if the instrument's IP address is 192.168.1.101, enter 192.168.1.101 in the browser's address box.
3. Press **Enter** on the computer keyboard to open the instrument's web page.

NOTE

If the web page does not open in the browser, see LAN troubleshooting suggestions (on page 8-6).

Connecting to the LAN

Each device on the LAN (corporate or private) requires a unique IP address. Contact your IT department for details on obtaining an IP address before you deploy the Model 2651A on a corporate or private network.

⚠ CAUTION

Contact your corporate information technology (IT) department for permission before you connect the Model 2651A to a corporate network.

Setting the LAN configuration method

There are two methods used to configure the LAN.

AUTO: Use the Auto setting to allow the DHCP server to automatically set the LAN settings.

You do not need to set the LAN options manually. The DHCP server automatically configures the IP address, subnet mask and the default gateway. A DHCP server must be available on the LAN in order to use this option.

MANUAL: Use the Manual setting to manually configure the communication parameters.

The MANUAL setting requires you to configure the following:

- IP address
- Gateway
- Subnet mask

To select a LAN configuration method:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > METHOD**.
2. Select either **AUTO** or **MANUAL**.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, then press the **ENTER** key.

Setting the IP address

NOTE

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

The IP address does not need to be manually set if the LAN configuration method is set to AUTO.

To set the IP address (when LAN configuration method is set to MANUAL):

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > IP-ADDRESS**.
2. Turn the navigation wheel ◌ to select and enter a valid IP address for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, then press the **ENTER** key.

Setting the gateway

NOTE

Contact your IT department to secure a valid gateway for the instrument when placing the instrument on a corporate network.

The gateway does not need to be manually set if the LAN configuration method is set to AUTO.

To set the gateway (when LAN configuration method is set to MANUAL):

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > GATEWAY**.
2. Turn the navigation wheel ◌ to select and enter a valid gateway address for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, then press the **ENTER** key.

Setting the subnet mask

NOTE

Contact your IT department to secure a valid subnet mask for the instrument when placing the instrument on a corporate network.

The subnet mask does not need to be manually set if the LAN configuration method is set to AUTO.

To set the subnet mask (when LAN configuration method is set to MANUAL):

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > SUBNETMASK**.
2. Turn the navigation wheel ◌ to select and enter a valid subnet mask for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, then press the **ENTER** key.

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. The DNS removes the requirement to memorize the IP address; instead you only need to know the domain name.

Example:

```
Series2651A.yourcompany.com
```

NOTE

- If a DNS server is not part of the LAN infrastructure, this setting is not used.
- Contact your IT department to learn more about DNS.

To enable or disable DNS host name verification:

1. From the 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 command `lan.config.dns.hostname` (on page 7-107).
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, then press the **ENTER** key.

To enable or disable DNS registration:

1. From the front panel, press the **MENU** key then 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` (on page 7-107) 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**, then press the **ENTER** key.

To set the DNS server IP addresses:

1. From the front panel, press the **MENU** key then 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.

LAN speeds

Another characteristic of the LAN is speed. The Model 2651A negotiates with the host PC 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 front panel, press the **MENU** key, and then 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 LAN CONFIG 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 front panel, press **MENU** key, and then 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 front panel, press **MENU** key, and then select **LAN > STATUS > CONFIG/FAULT**.
2. Press the **ENTER** key.

Figure 123: LAN CONFIG/FAULT (Series 2650)

There are two types of LAN status messages:

- **LAN fault messages:** Communicates issues related to physical connectivity.
- **LAN configuration messages:** Communicates 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 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 Model 2651A 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 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 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 Model 2651A 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 remote command interface

This section provides details on how to select a remote command interface to connect to the Model 2651A.

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 VISA software is required. Test Script Builder (TSB) uses VISA and can be used with the VXI-11 interface. You can expect a slower connection with this protocol.

Raw socket connection

Raw socket is a basic Ethernet connection that communicates similarly to RS-232 without explicit message boundaries. The instrument will always terminate 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 reboots without first closing the socket. This port cannot be used for command and control functions.

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

Telnet connection

Telnet is similar to raw socket and is used when the user needs to interact directly with the instrument, typically for debugging and troubleshooting. Telnet requires a separate telnet program. The Model 2651A supports the Telnet protocol that you can use over a TCP/IP connection to issue commands to the instrument. You can use a Telnet connection to interact with scripts or issue commands in real-time.

Configuring a Telnet connection

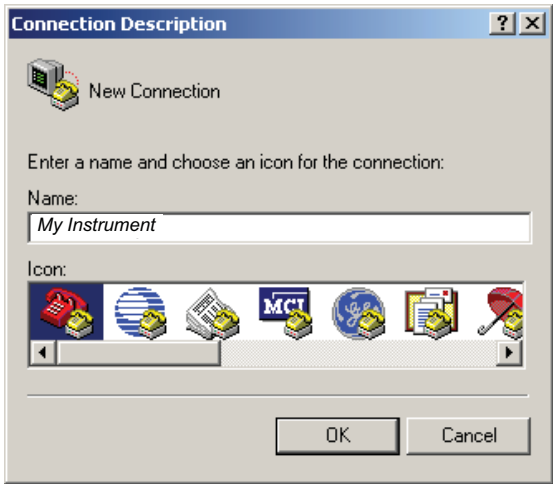
NOTE

This procedure uses HyperTerminal, which is available with the Microsoft® Windows® XP operating system. Consult the help system for your version of Microsoft Windows to identify a compatible tool.

To connect with the Model 2651A using HyperTerminal on a Windows XP system:

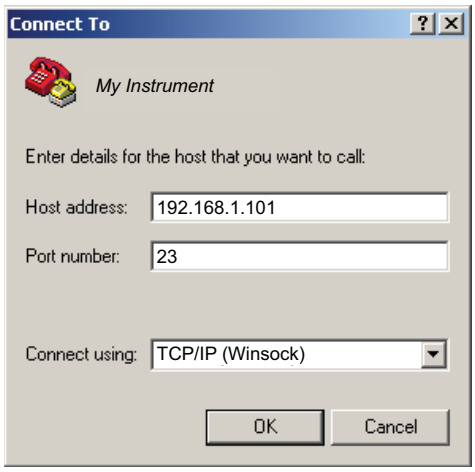
1. On the host computer, click **Start > Accessories > Communications > HyperTerminal**. The Connection Description dialog box will open.

Figure 124: Connection description dialog box



2. Type a name to identify the connection (for example, *My Instrument*) and then click **OK**.
3. In the Connect To dialog box, click the **Connect using** drop-down list and then select **TCP/IP (Winsock)**.

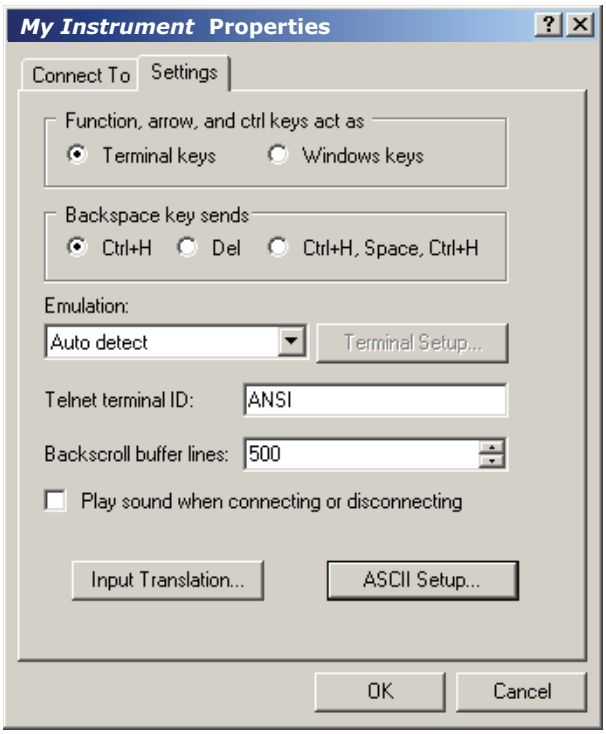
Figure 125: Connect To dialog box



4. In the Host address field, type the instrument's IP address (for example, *192.168.1.101*) .
5. Type **23** in the **Port number** field, and then click **OK**. The HyperTerminal program window displays.
6. From the HyperTerminal program window, click **File > Properties**.

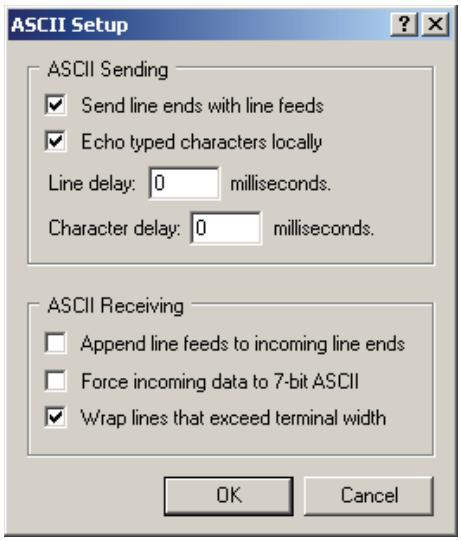
7. In the properties dialog box, click the **Settings** tab.

Figure 126: Properties dialog box



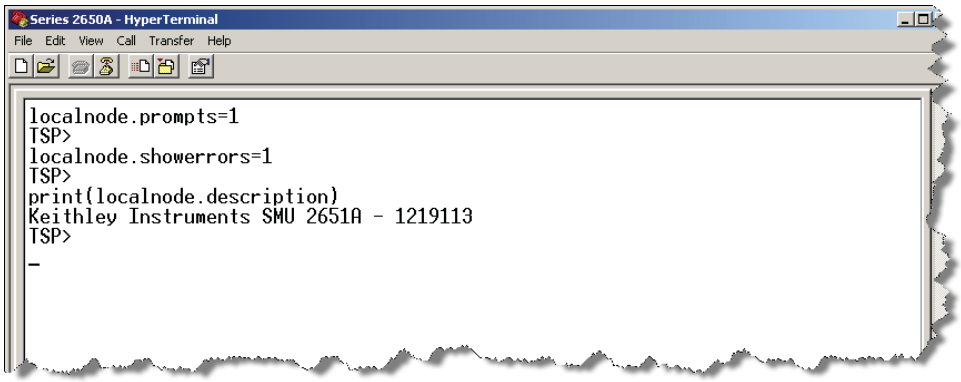
8. Click **ASCII Setup**. The ASCII Setup dialog box is displayed.
9. From the ASCII Setup window, select the following options:
   - **Send line ends with line feeds**
   - **Echo typed characters locally**

Figure 127: ASCII Setup window



10. Click **OK** in the ASCII Setup dialog box. The Properties window displays.
11. Click **OK** in the Properties dialog box.
12. Use the HyperTerminal window to interact directly with the instrument.

Figure 128: Communicating with the instrument through HyperTerminal window



Logging LAN trigger events in the event log

The event log can be used to record all LXI triggers generated and received by the Model 2651A and can be viewed over any command interface. The event log can also be viewed using the embedded web interface. The following figure shows the view of the LXI event log from the embedded web interface.

Figure 129: Event log

The timestamp, event identifier, the 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
Received Time	• Displays the date and time of the LAN trigger occurred • Displays the value in UTC, 24-hour time	06:56:28.000 8 May 2008

EventID	Identifies the `lan.trigger[N]` that generates an event	`LAN0 = lan.trigger[1]`
		`LAN1 = lan.trigger[2]`
		`LAN2 = lan.trigger[3]`
		`LAN3 = lan.trigger[4]`
		`LAN4 = lan.trigger[5]`
		`LAN5 = lan.trigger[6]`
		`LAN6 = lan.trigger[7]`
		`LAN7 = lan.trigger[8]`

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 Model 2651A 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 1523
Flags	Contain data about the LXI trigger packet	Values: • 1 - Error • 2 - Retransmission • 4 - Hardware • 8 - Acknowledgments • 16 - Stateless bit
Data	The Model 2651A does not support the IEEE Std 1588 standard; the values in this are always 0 (zero)	

Accessing the event log from the command interface

The event log can be accessed 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 2008, LAN0, 10.80.64.191, LXI, 0, 1213712000, not
available, 0, 0x10,0x00
```

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

- UTC time
- Event Id
- Sender
- HwDetect / version
- Domain
- sequence number
- ptp time
- epoch (from 1588)
- flags
- Data

See the table in Logging LAN trigger events in the event log (on page C-19) for detailed descriptions.

Appendix D

Common commands

In this appendix:

Command summary D-1
Script command equivalents D-2
Command reference D-2

Command summary

Common commands supported by the Model 2651A are summarized in the following table. Although commands are shown in upper-case, common commands are not case sensitive, and either upper or lower case can be used. Note that although these commands are essentially the same as those defined by the IEEE- 488.2 standard, the Model 2651A does not strictly adhere to that standard.

NOTE

Unlike other commands, such as those in the Command reference (on page 7-1), each common command must be sent as separate lines.

Mnemonic	Name	Description
*CLS	Clear status	Clears all event registers and Error Queue. For status commands details, see Status model (on page E-1).
*ESE <mask>	Event enable command	Program the Standard Event Status Enable Register. For status commands details, see Status model (on page E-1).
*ESE?	Event enable query	Read the Standard event status enable Register. For status commands details, see Status model (on page E-1).
*ESR?	Event status register query	Read/clear the Standard Event Enable Register. For status commands details, see Status model (on page E-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 D-3).
*OPC	Operation complete command	Set the Operation Complete bit in the Standard Event Register after all pending commands have completed. For detailed information, see Operation complete and query: *OPC and *OPC? (on page D-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 D-3).
*RST	Reset command	Returns the Model 2651A to default conditions. For detailed information, see Reset: *RST (on page D-3).
*SRE <mask>	Service request enable command	Programs the Service Request Enable Register. For status commands details, see Status model (on page E-1).
*SRE?	Service request enable query	Reads the Service Request Enable Register. For status commands details, see Status model (on page E-1).
*STB?	Status byte query	Reads the status byte register. For status commands details, see Status model (on page E-1).

Mnemonic	Name	Description
*TRG	Trigger command	Generates the trigger.EVENT.ID for use with the trigger model and scanning. For detailed information, see Trigger: *TRG (on page D-3).
*TST?	Self-test query	Returns a 0. For detailed information, see Self-test query: *TST? (on page D-3).
*WAI	Wait-to-continue command	Waits until all previous commands have completed. For detailed information, see Wait-to-continue: *WAI (on page D-4).

Script command equivalents

Script command equivalents for the common commands are defined in the table below.

Common command	Script command equivalent
*CLS	`status.reset()`
*ESE?	`print(tostring(status.standard.enable))`
*ESE <mask>	`status.standard.enable = <mask>`
*ESR?	`print(tostring(status.standard.event))`

*IDN?	`print([[Keithley Instruments Inc.,`
	`Model]]..localnode.model..[[, ]]..localnode.serialno..`
	`[[, ]]..localnode.revision)`

*OPC?	`waitcomplete() print([[1]])`
*OPC	`opc()`
*RST	`reset()`
*SRE?	`print(tostring(status.request_enable))`
*SRE <mask>	`status.request_enable = <mask>`
*STB?	`print(tostring(status.condition))`
*TRG	N/A
*TST?	`print([[0]])`
*WAI	`waitcomplete()`

Command reference

Details on all common commands except those associated with the status model are covered below.

NOTE

Status command usage is contained in the Status model (on page E-1).

Identification query: *IDN?

`*IDN?` Command that reads ID information. The identification string includes the manufacturer, model number, serial number, and firmware revision levels and is sent in the following format:

```
Keithley Instruments Inc., Model 2651A, xxxxxxx, yyyyy
```

Where:

xxxxxxx is the serial number.

yyyyy is the firmware revision level.

Operation complete and query: *OPC and *OPC?

`*OPC` Operation complete command that sets the OPC bit.

`*OPC?` Operation complete query that places a "1" in output queue.

When `*OPC` is sent, the OPC bit in the Standard Event Register (see Status model (on page E-1)) will set when all overlapped commands complete. An ASCII "`1`" is also placed in the Output Queue to be read by the `*OPC?` query when overlapped commands complete.

Reset: *RST

`*RST` This command returns the instrument to defaults.

When the `*RST` command is sent, the instrument returns to the default conditions (see reset() (on page 7-151)).

Self-test query: *TST?

`*TST?` This command always places a `0` in the Output Queue. It is included for common command compatibility only; the Model 2651A does not actually perform a self-test.

Trigger: *TRG

`*TRG`: This command generates the `trigger.EVENT_ID` for the trigger models. The `trigger.EVENT_ID` is applicable to all trigger stimulus commands (for example, trigger.timer[N].stimulus (on page 7-344)). Also see Trigger model.

Wait-to-continue: *WAI

`*WAI` This pauses until other commands are completed. Two types of device commands exist:

- **Sequential commands**: A command whose operations are allowed to finish before the next command is executed.
- **Overlapped commands**: A command that allows the execution of subsequent commands while device operations of the overlapped command are still in progress.

The `*WAI` command is used to suspend the execution of subsequent commands until the device operations of all previous overlapped commands are finished. The `*WAI` command is not needed for sequential commands.

Appendix E

Status model

In this appendix:

Overview E-1
Clearing registers and queues E-12
Programming and reading registers E-13
Status byte and service request (SRQ) E-14
Status register sets E-18
Queues E-25
TSP-Link system status E-26

Overview

Each Keithley Instruments Model 2651A High Power System SourceMeter® Instrument provides a number of 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. Commands included in your test program 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 eventually flow into the Status Byte Register. As a programmer, you are in full control of all enable registers, while the High Power System SourceMeter® Instrument has full control of all event status registers. In order for an event to be accounted for in the register's summary bit, first enable it by setting the appropriate bit in the applicable enable register. The entire status model is illustrated in the Status model diagrams (on page E-4) topic.

Status Byte Register

The Status Byte Register receives summary bits from the other status register sets and queues, and also from itself (which sets the Master Summary Status, or MSS, bit). For details see Status Byte Register (on page E-14).

Status register sets

Typically, a status register set contains the following registers:

- **Condition** (`.condition`): A read-only register that constantly updates 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 will set to 1.
- **Negative Transition Register** (`.ntr`): A read-write register that when set to 1, specifies a negative transition (change from 1 to 0) sets the event register bit.
- **Positive Transition Register** (`.ptr`): A read-write register that when set to 1, specifies a positive transition (change from 0 to 1) sets the event register bit.

When an event occurs and the appropriate NTR or PTR bit is set, the matching event register bit is set to 1. The event bit remains latched to 1 until the register is read or the status model is reset. When an event register bit is set and its corresponding enable bit is set, the output (summary bit) of the register will set to 1. This, in turn, sets a bit in a higher-level register cascading to the associated summary bit of the Status Byte Register.

Queues

The High Power System SourceMeter® Instrument uses an output queue and an error or event queue. Response messages, such as those generated from print commands, are placed in the output queue. As programming errors and status messages occur, they are placed in the error queue. When a queue contains data, it sets the appropriate summary bit of the Status Byte Register (EAV for the error or event queue; MAV for the output queue).

The Status model overview (on page E-4) shows how the two queues are structured with the other registers.

Service requests and connections

Service requests (SRQs) affect both the GPIB and the VXI-11 connections. On a GPIB connection, the SRQ line is asserted. On a VXI-11 connection, an SRQ event is generated.

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 the Command reference (on page 7-1).

Status function summary

Type	Function or attribute*
Status summary	status.condition status.node_enable status.node_event status.request_enable status.request_event status.reset
Measurement event	status.measurement.* status.measurement.buffer_available.* status.measurement.current_limit.* status.measurement.instrument.* status.measurement.instrument.smua.* status.measurement.reading_overflow.* status.measurement.voltage_limit.*
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.smua.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.*

Status function summary

Type	Function or attribute*
Questionable status	status.questionable.* status.questionable.calibration.* status.questionable.instrument.* status.questionable.instrument.smua.* status.questionable.over_temperature.* status.questionable.unstable_output.* 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 model overview (on page E-4)
- Measurement event registers (on page E-5)
- System summary and standard event registers (on page E-6)
- Operation status registers (on page E-7)
- Operation status trigger overrun registers (on page E-8)
- Operation status trigger timer, trigger blender, and remote registers (on page E-9)
- Operation status digital I/O and TSP-Link registers (on page E-10)
- Questionable status registers (on page E-11)

Figure 130: Status model overview

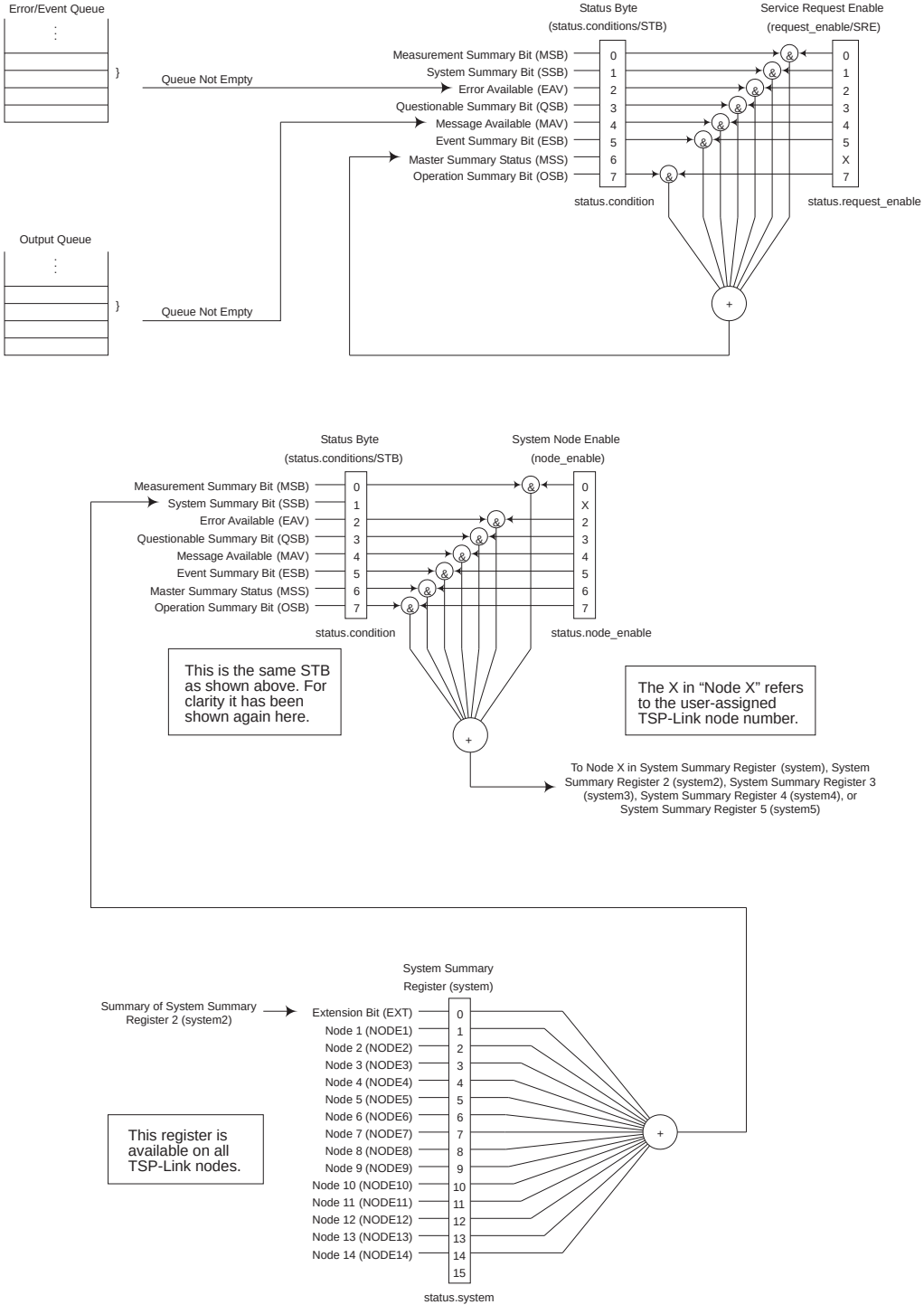


Figure 131: Measurement event registers

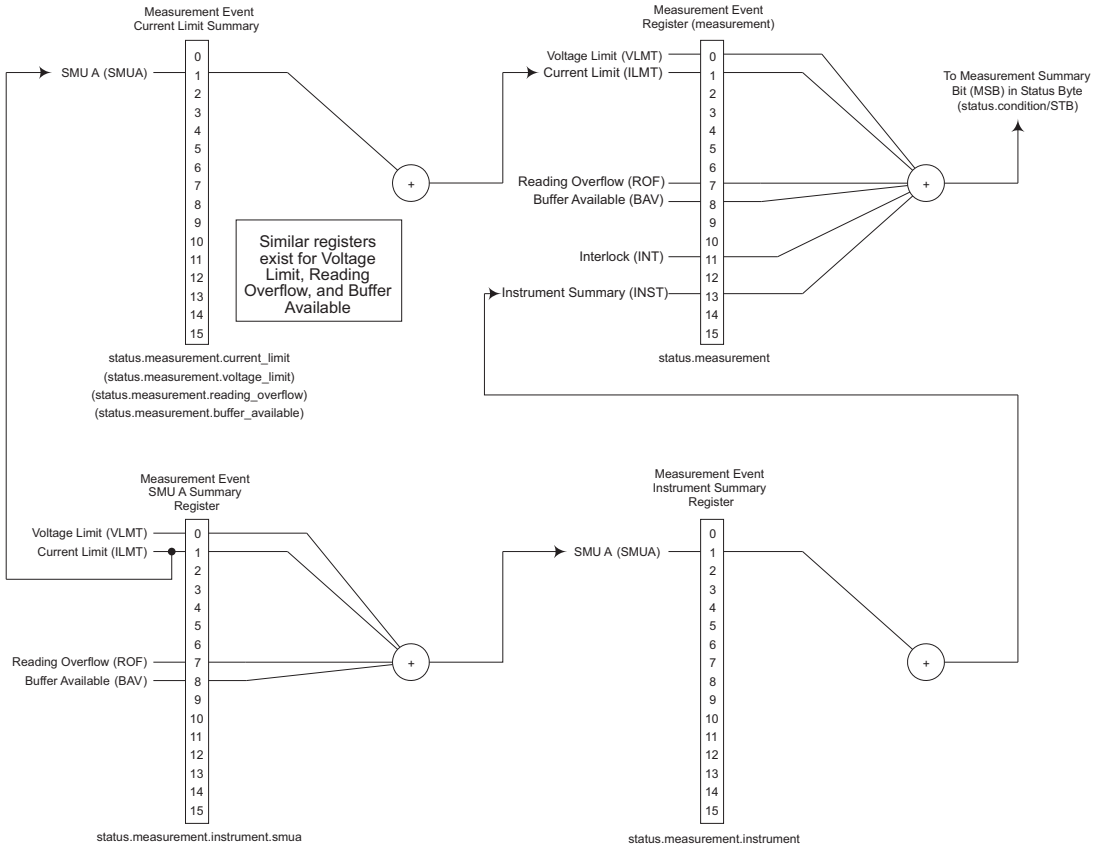


Figure 132: System summary and standard event registers

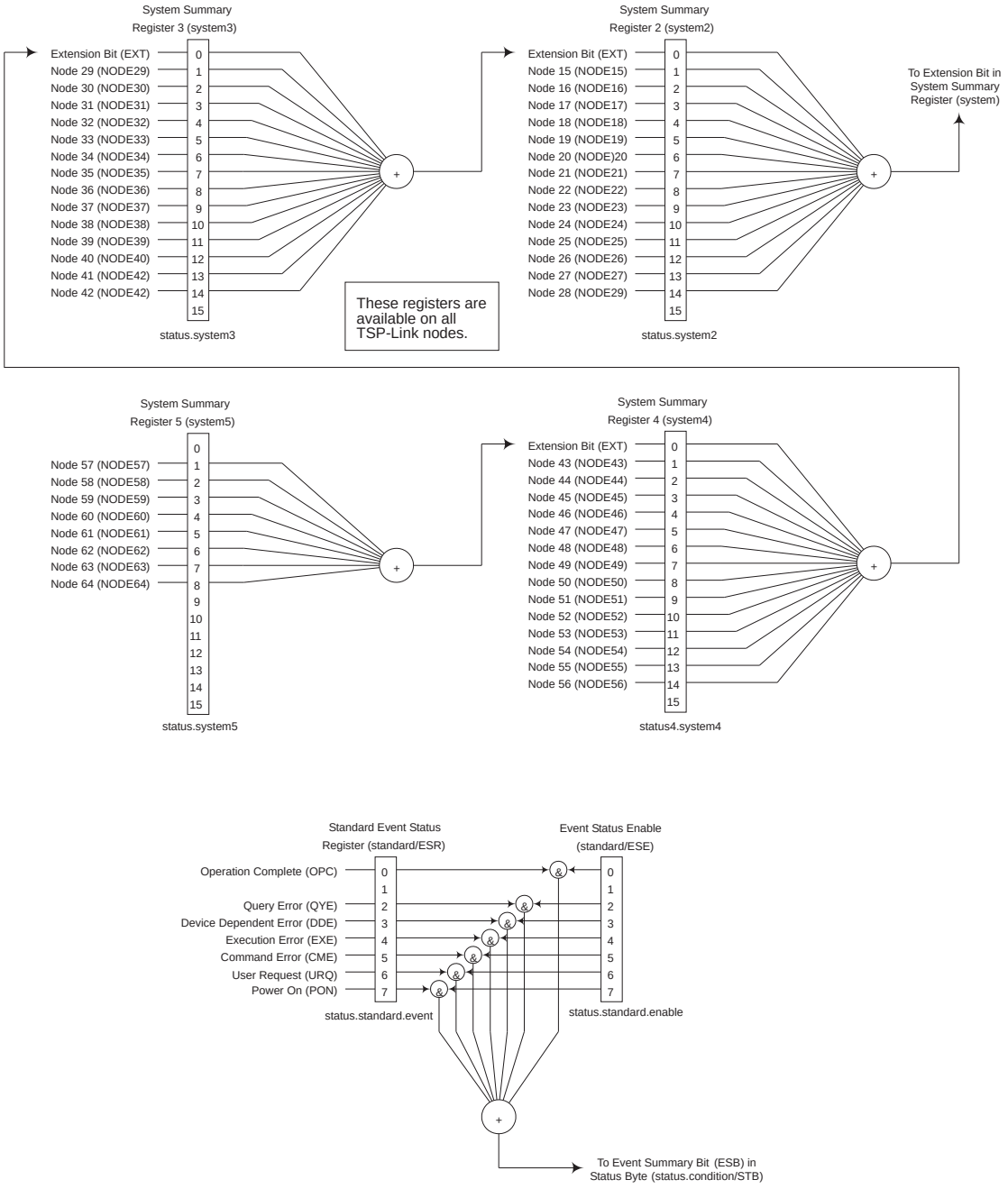


Figure 133: Operation status registers

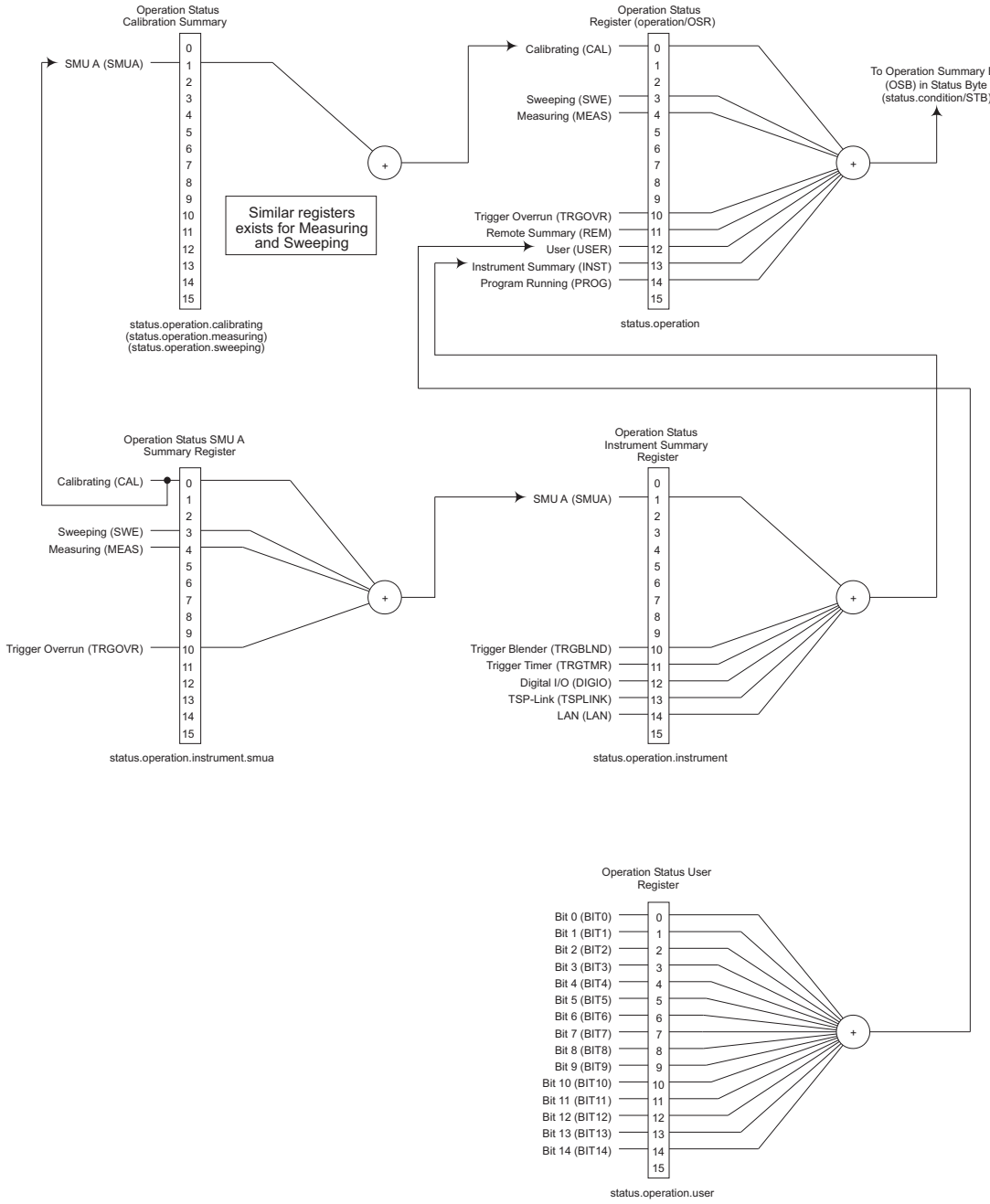


Figure 134: Operation status trigger overrun registers

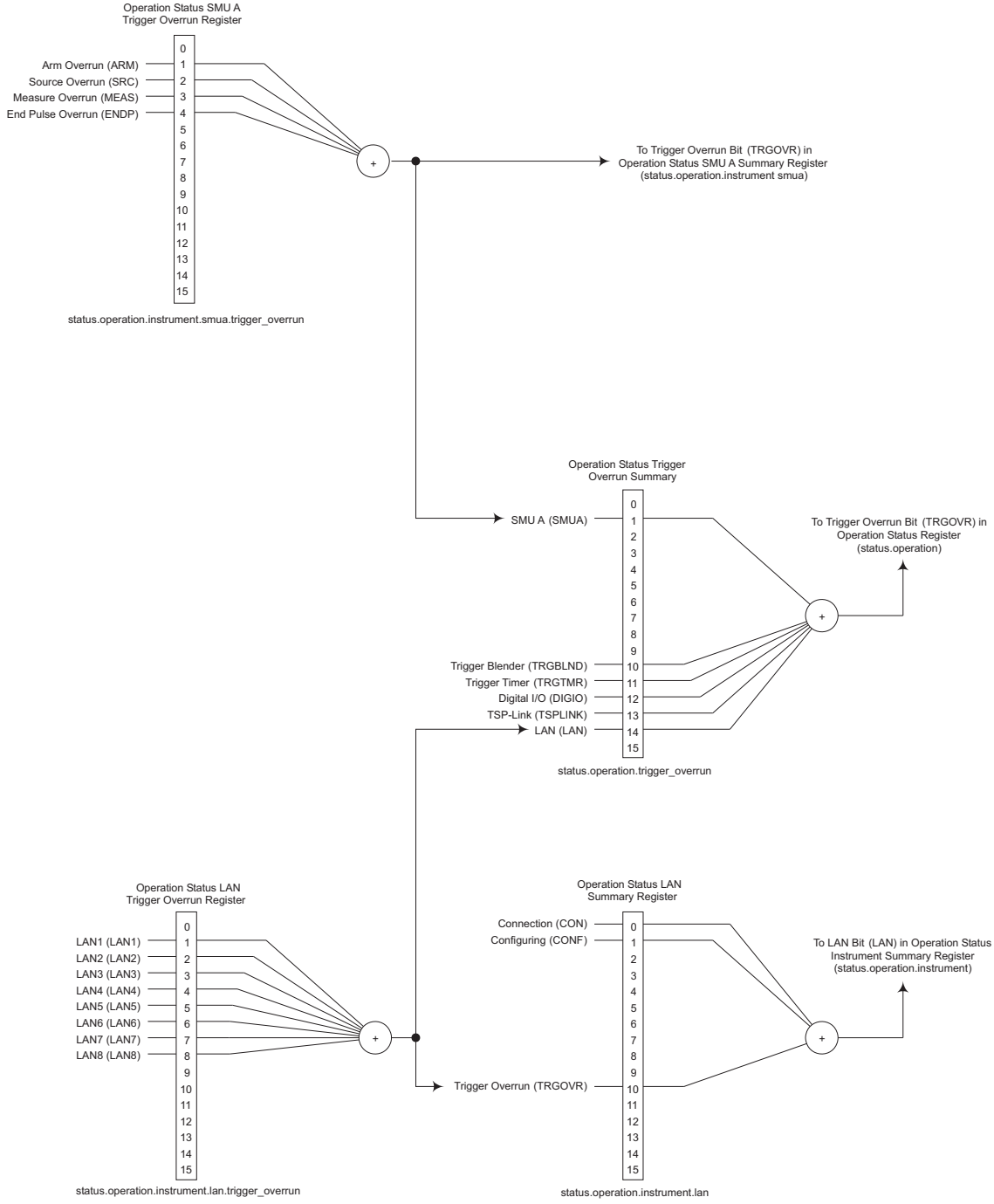


Figure 135: Operation status trigger timer, trigger blender, and remote registers

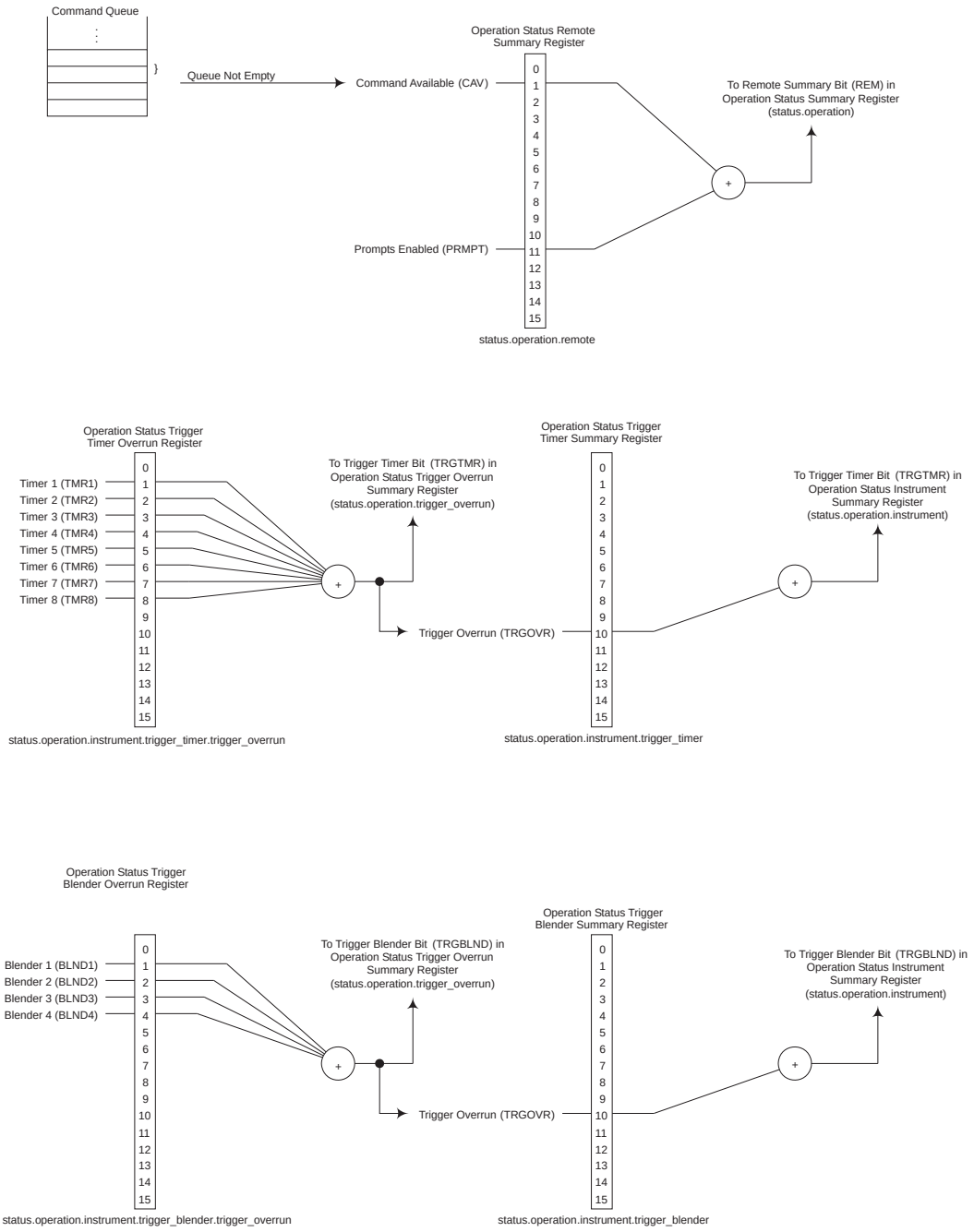


Figure 136: Operation status digital I/O and TSP-Link registers

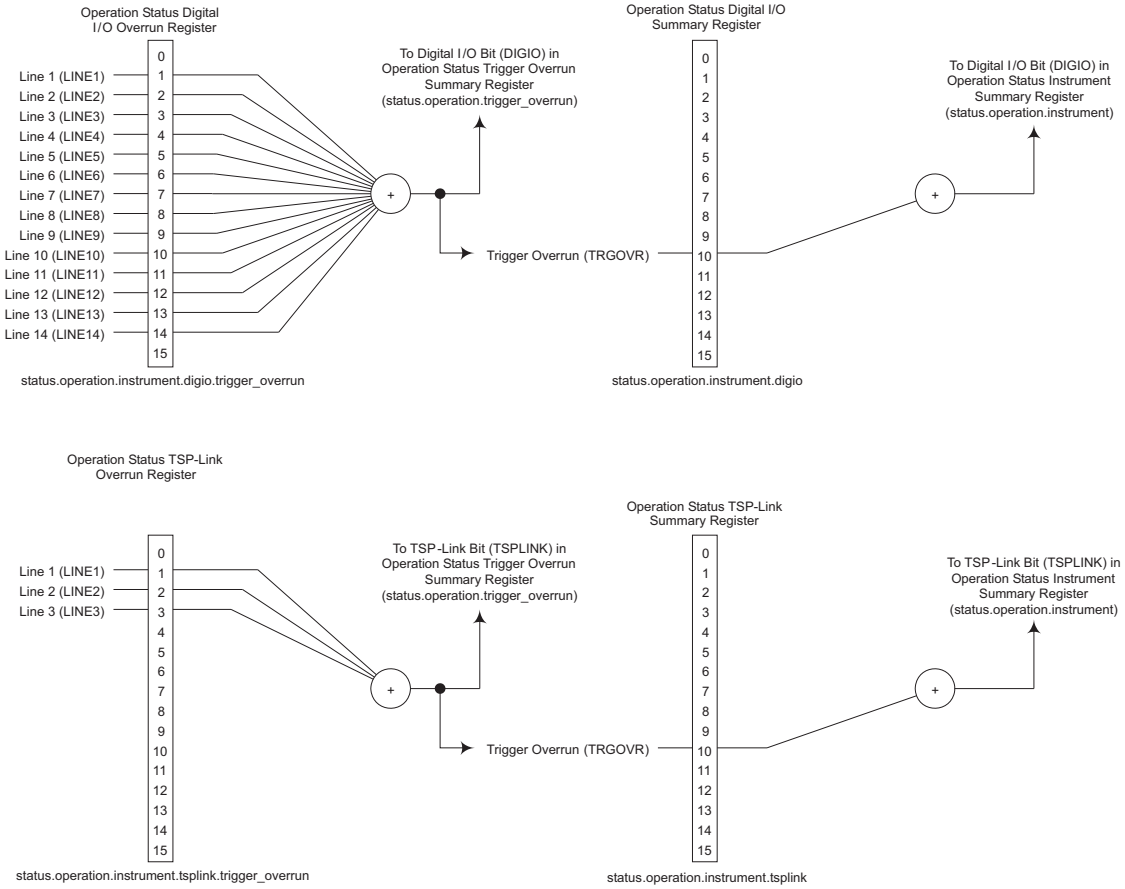
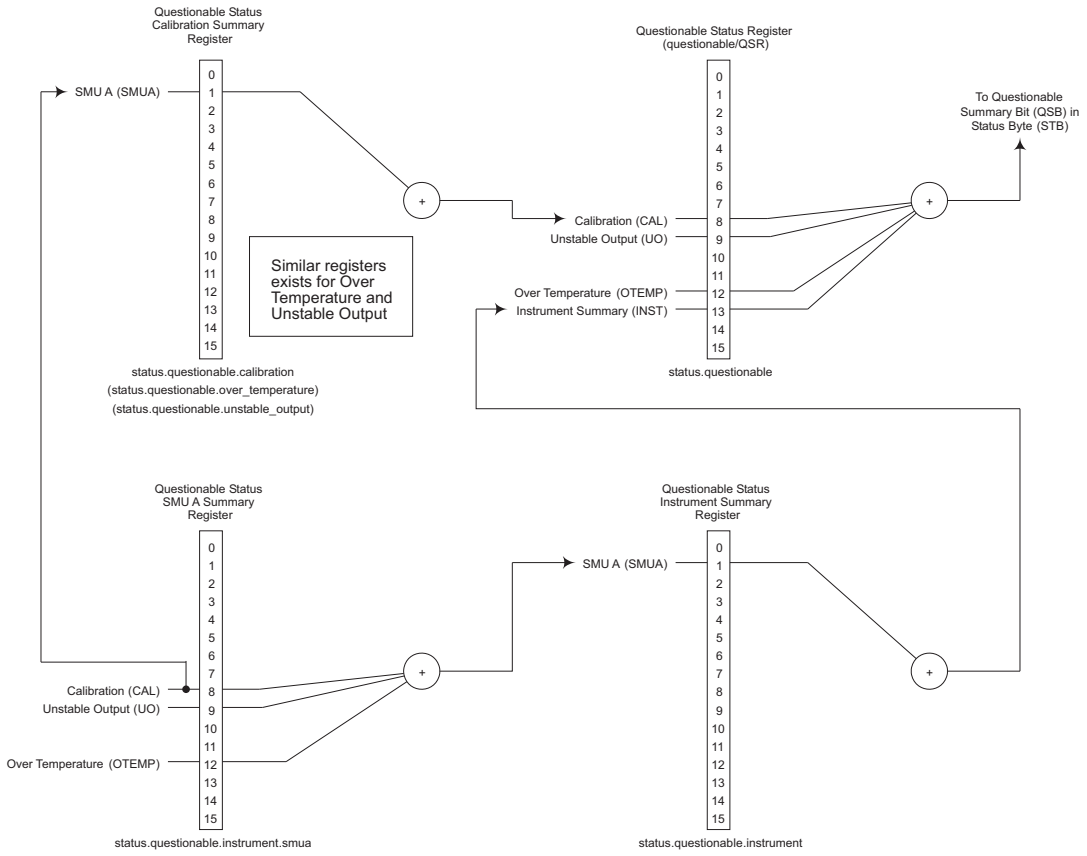


Figure 137: Questionable status registers



Clearing registers and queues

Commands to reset the status registers and the error queue are listed in the table below. In addition to these commands, any programmable register can be reset by sending the individual command to program the register with a 0 as its parameter value.

Commands to reset registers and clear queues

Commands	Description
To reset registers:	
`*CLS`	Clears the output queue. Reset bits of status registers to 0.
`status.reset()`	Reset bits of status registers to 0.
To clear error queue:	
`errorqueue.clear()`	Clear all messages from error queue.

The instrument automatically clears the output queue when the instrument transitions from the local control state to the remote control state.

Programming and reading registers

Programming enable and transition registers

The only registers that can be programmed by the user 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 actual commands are summarized in Common commands (on page D-1) and Status function summary (on page E-3).

A command to program an event enable or transition register is sent with a parameter value that determines the desired 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 E-18).

Figure 138: 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 Weights	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)

A. Bits 0 through 7

Bit position	B15	B14	B13	B12	B11	B10	B9	B8
Binary alue	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal Weights	32768 (2^{15})	16384 (2^{14})	8192 (2^{13})	4096 (2^{12})	2048 (2^{11})	1024 (2^{10})	512 (2^9)	256 (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 will be 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 E-13). For example, for a decimal value of 37 (binary value of 100101), bits B5, B2, and B0 are set.

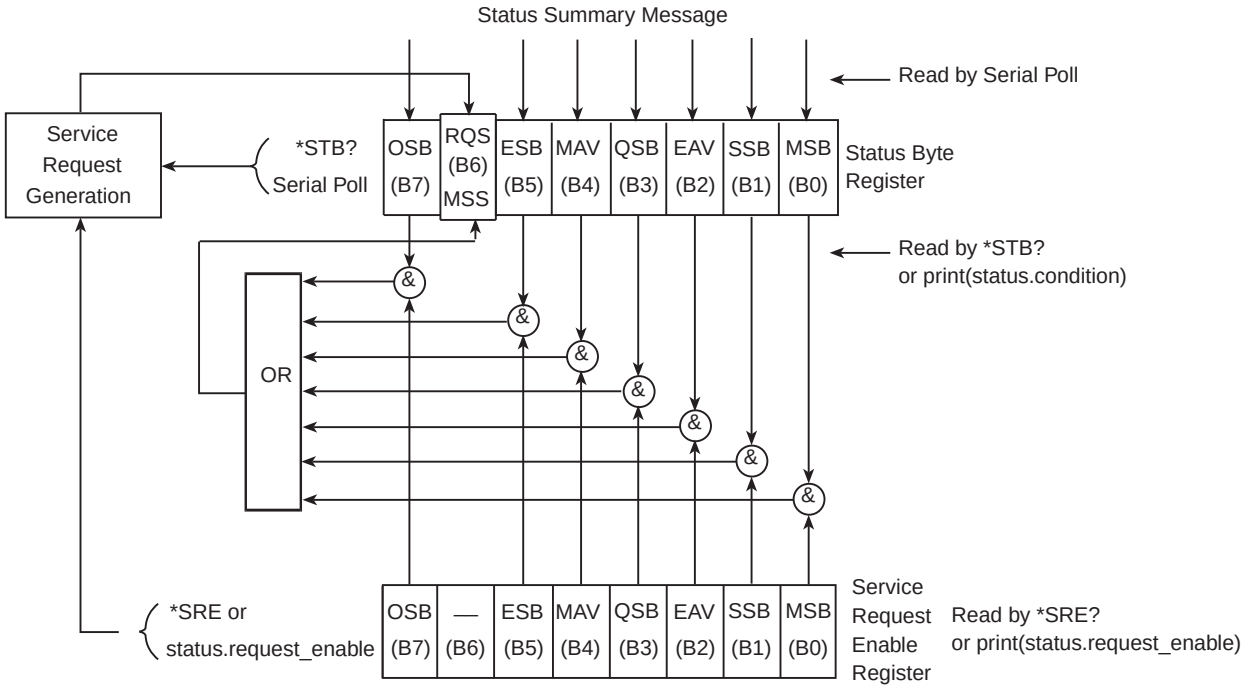
Status byte and service request (SRQ)

Two 8-bit registers control service requests: The Status Byte Register and the Service Request Enable Register. The Status Byte Register (on page E-14) topic describes the structure of these registers.

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.

Figure 139: Status byte and service request (SRQ)



OSB = Operation Summary Bit
MSS = Master Summary Status
RQS = Request for Service
ESB = Event Summary Bit
Mav = Message Available
QSB = Questionable Summary Bit
EAV = Error Available
SSB = System Summary Bit
MSB = Measurement Summary Bit
& = Logical AND
OR = Logical OR

The bits of the Status Byte Register are described as follows:

- **Bit B0, Measurement Summary Bit (MSB):** Set summary bit indicates that an enabled measurement event has occurred.
- **Bit B1, System Summary Bit (SSB):** Set summary bit indicates that an enabled system event has occurred.
- **Bit B2, Error Available (EAV):** Set bit indicates that an error or status message is present in the error queue.
- **Bit B3, Questionable Summary Bit (QSB):** Set summary bit indicates that an enabled questionable event has occurred.
- **Bit B4, Message Available (MAV):** Set bit indicates that a response message is present in the output queue.
- **Bit B5, Event Summary Bit (ESB):** Set summary bit indicates that an enabled standard event has occurred.
- **Bit B6, Request Service (RQS)/Master Summary Status (MSS):** Set bit 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 the GPIB serial poll sequence of the Model 2651A to obtain the status byte (serial poll byte), B6 is the RQS bit. See Serial polling and SRQ (on page E-17) for details on using the serial poll sequence.
  - When using the *STB? common command or status.condition Status byte and service request commands (on page E-17) to read the status byte, B6 is the MSS bit.
- **Bit B7, Operation Summary (OSB):** Set summary bit 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 the Status Byte Register (on page E-14) topic, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic "1" output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

The individual bits of the Service Request Enable Register can be set or cleared 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 E-17).

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 will set bit B6 and generate a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ has occurred and what caused it. If an SRQ occurs, the program can, for example, branch to an appropriate subroutine that will service the request.

SRQs can be managed by the serial poll sequence of the High Power System SourceMeter® Instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register will remain cleared, and the program will simply proceed normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register will 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 generated by other event types.

For common and script commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit stays set until all Status Byte Register summary bits are reset.

SPE, SPD (serial polling)

For the GPIB interface only, the SPE and SPD general bus commands are used to serial poll the High Power System SourceMeter® Instrument (see General bus commands (on page 2-78)). Serial polling obtains the serial poll byte (status byte). Typically, serial polling is used by the controller to determine which of several instruments has requested service with the SRQ line.

Status byte and service request commands

The commands to program and read the Status Byte Register and Service Request Enable Register are listed in Status byte and service request commands (on page E-17). 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 E-13) and Reading registers (on page E-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

`*STB?`
or
`print(status.condition)`
	Read the Status Byte Register.

`*SRE <mask>`
or
`status.request_enable = <mask>`
	Program the Service Request Enable Register where `<mask>` = 0 to 255.

`*SRE?`
or
`print(status.request_enable)`
	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 reference (see "Commands" on page 7-8) command table and defined as follows:

- **Enable register** (identified as `.enable` in each attribute's command listing): Allows various associated events to be included in the summary bit for the register.
- **Negative-transition register** (identified as `.ntr` in each attributes command listing): A particular bit in the event register will be 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 each attributes command listing): A particular bit in the event register will be 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 Register overview (on page E-4). For example, either of the following will 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 High Power System SourceMeter® Instrument:

- System Summary
- Standard Event Status
- Operation Status
- Measurement Event
- Questionable Status

System Summary Registers

As shown in Status model diagrams (on page E-4), there are five register sets associated with system status events. These registers summarize system status for various nodes connected to the TSP-Link® network (see Sweep Operation (on page 3-19)). 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, the status model can be configured such 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 E-26) 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 E-3) table.

For example, either of the following commands will set 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 E-14). For example, the following command will read 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 a another system status register is set.
- **Bits B1-B14* NODEn:** Indicates a bit on TSP-Link node *n* has been set (*n* = 1 to 64).

***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 System SourceMeter® 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 D-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 System SourceMeter 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 System SourceMeter instrument received a message that does not follow the defined syntax of IEEE Std 488.2.
  - Semantic error: System SourceMeter 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 System SourceMeter instrument front panel was pressed.
- **Bit B7, Power ON (PON):** Set bit indicates that the System SourceMeter instrument has been turned off and turned back on since the last time this register has been read.

Commands to program and read the register are summarized below and also in the Status function summary (on page E-3) table.

Standard event commands

Command	Description

`*ESR?`
or
`print(status.standard.event)`
	Read Standard Event Status Register.

`*ESE <mask>`
or
`status.standard.enable = <mask>`
	Program the Event Status Enable Register:
	`<mask>` = 0 to 255
	See Status register sets (on page E-18).

`*ESE?`
or
`print(status.standard.enable)`
	Read Event Status Enable Register.

Operation Status Registers

As shown in the status model's Operation status registers (on page E-7) diagram, there are 22 register sets associated with operation status. Commands are summarized in the Status register sets (on page E-18) topic. Note that bits can also be set by using numeric parameter values. For details, see Programming enable and transition registers (on page E-13).

For example, either of the following commands will set 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 E-14). For example, the following command will read the Operation Status Enable Register:

```
print(status.operation.enable)
```

Commands to program and read the register are summarized in the Status function summary (on page E-3) table.

Operation Status Register

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 will be set when taking an overlapped measurement, but it will not set when taking 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 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 sets (on page E-2) and the charts in this appendix.

Questionable Status Registers

As shown in the status model's Operation event, I/O, and TSP-Link registers (on page E-10), there are seven register sets associated with Questionable Status. Commands are summarized in the Status byte and service request (SRQ) (on page E-14) topic. Note that bits can also be set by using numeric parameter values. For details, see Programming enable and transition registers (on page E-13).

For example, either of the following commands will set 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 E-14). For example, the following command will read the Questionable Status Enable Register:

```
print(status.questionable.enable)
```

For more information about the Questionable Status Registers, refer to Status register sets (on page E-2) and the charts in this appendix.

Questionable Status Register

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 Instrument Summary Register is set.

For more information on the Questionable Status Register, refer to Status register sets (on page E-2) and the charts in this appendix.

Measurement Event Registers

As shown in the status model's Measurement event registers (on page E-5), there are eight register sets associated with measurement event status. Commands are summarized in the Status register sets (on page E-2) topic. Note that bits can also be set by using numeric parameter values. For details, see Programming enable and transition registers (on page E-13) .

For example, either of the following commands will set 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 E-14). For example, the following command will read 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 described as follows:

- **Bit B0, Voltage Limit (VLMT):** Set bit indicates that the voltage limit was exceeded. This bit will be updated only when (1) a measurement is taken or (2) the `smuX.source.compliance` command is invoked.
- **Bit B1, Current Limit (ILMT):** Set bit indicates that the current limit was exceeded. This bit will be updated only when (1) a measurement is taken or (2) the `smuX.source.compliance` command is invoked.
- **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 of the nonvolatile reading buffers.
- **Bit B11, Interlock (INT):** Set bit indicates that interlock 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 E-3) table. For more information about the Measurement Event Registers, refer to Status register sets (on page E-2) and the charts in this appendix.

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
```

Queues

The Model 2651A uses two queues, which are first-in, first-out (FIFO) queues:

- Output queue: Used to hold response messages.
- Error queue: Used to hold error and status messages (see Error summary list (on page 8-2)).

The Model 2651A status model (Status model overview (on page E-4)) shows how the two queues are structured with the other registers.

Output queue

All interfaces share the same output queue. The output queue sets the message available (MAV) bits in the status model. The data in the output queue clears if the mode changes to local mode.

NOTE

You must save the data from the output queue while the instrument is communicating with the remote command interface. All data in the output queue is cleared when the instrument returns to local mode.

Error queue

The error queue holds error and status messages. When an error or status event occurs, a message that defines the error or status is placed in the error queue.

When a message is placed in the error queue, the error available (EAV) bit in the Status Byte Register is set. An error or status message is cleared from the error queue when it is read. The error queue is considered cleared when it is empty. An empty error queue clears the EAV bit in the Status Byte Register.

The commands to control the error queue are listed below. When you read a single message in the error queue, the oldest message is read and then removed from the queue. When the instrument power is turned on, the error queue is initially empty. If there are problems detected when the instrument power is turned on, entries will be placed in the queue. When empty, the error number 0 and “No Error” is placed in the queue.

Messages in the error queue include a code number, message text, severity, and TSP-Link® node number. The messages are listed in Error summary (on page 8-1).

Error queue commands

Error queue command	Description
`errorqueue.clear()`	Clear error queue of all errors.
`errorqueue.count`	Number of messages in the error/event queue.
`errorCode, message, severity, errorNode = errorqueue.next()`	Request error code, text message, severity, and TSP-Link node number.

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 64 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 6-47) for details about the TSP-Link system.

The system summary registers, shown in Status Byte Register overview (on page E-4) and System summary and standard event registers (on page E-6), 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 E-14) for details.

Status model configuration example

In Status configuration (enable) commands (on page E-27), there is an example illustrating the status model configuration for a TSP-Link® system. In this example, a current limit (compliance) event in SMU A of node 15 will set 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 E-27).

When a current limit (compliance) condition occurs in SMU A of Node 15, the following sequence of events will occur:

- Node 15: Bit B1 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 System2 Summary Register 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 will detect the current limit event and can branch to a routine to service the request.

Status configuration (enable) commands

The following commands (sent from the master node) enable the appropriate register bits for the above example:

Node 15 status registers: The following commands enable the current limit events for SMU A 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 following figure).

System summary registers: The following commands enable the required system summary bits for node 15:

```
status.system2.enable = 2
status.system.enable = 1
```

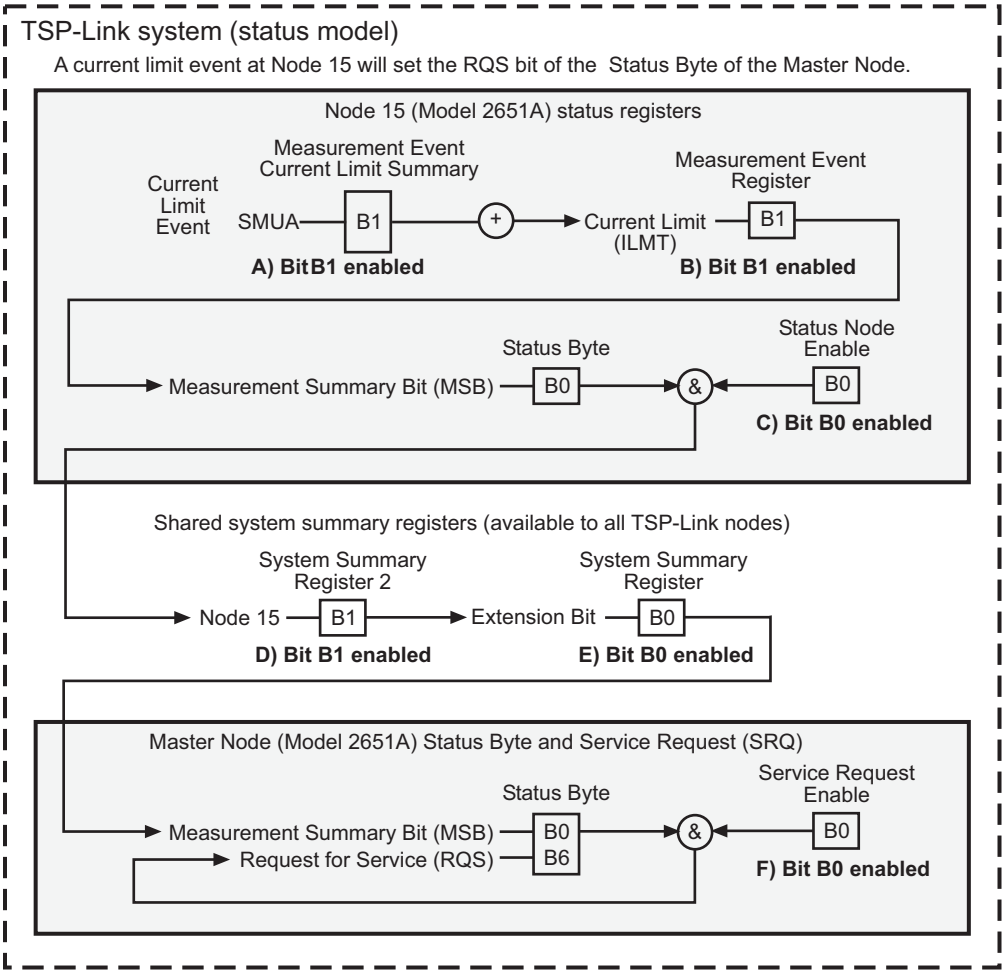
The affected system summary registers for the above commands are indicated by labels D and E (see following figure).

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 following figure).

Figure 140: TSP-Link status model configuration example



Appendix F

Display character codes

In this appendix:

Model 2651A display character codes F-2
Display character dot patterns F-5

Model 2651A display character codes

This topic provides a listing of character codes (decimal values) and their corresponding display.

Display character codes (decimal 0-83)

DECIMAL	DISPLAY	DECIMAL	DISPLAY	DECIMAL	DISPLAY
000	reserved	028	◄	056	8
001	reserved	029	►	057	9
002	reserved	030*	selftest1	058	:
003	reserved	031*	selftest2	059	;
004	reserved	032	(space)	060	<
005	reserved	033	!	061	=
006	reserved	034	"	062	>
007	reserved	035	#	063	?
008	reserved	036	$	064	@
009	reserved	037	%	065	A
010	reserved	038	&	066	B
011	reserved	039	' (apostrophe)	067	C
012	reserved	040	(	068	D
013	reserved	041	)	069	E
014	reserved	042	*	070	F
015	reserved	043	+	071	G
016	µ	044	, (comma)	072	H
017	±	045	-	073	I
018	Ω	046	.	074	J
019	°	047	/	075	K
020*	leftflagbar	048	0	076	L
021*	rightflagbar	049	1	077	M
022*	fullflagbar	050	2	078	N
023*	leftbar	051	3	079	O
024*	rightbar	052	4	080	P
025*	fullbar	053	5	081	Q
026	▲	054	6	082	R
027	▼	055	7	083	S
* The dot patterns for this character can be found in Display character dot patterns (on page F-5).					

Display character codes (decimal 84-185)

DECIMAL	DISPLAY	DECIMAL	DISPLAY	DECIMAL	DISPLAY
084	T	118	v	152*	selftest11
085	U	119	w	153*	selftest12
086	V	120	x	154*	selftest13
087	W	121	y	155*	selftest14
088	X	122	z	156*	box1
089	Y	123	{	157*	box2
090	Z	124	|	158*	.5 (one-half)
091	[	125	}	159	¼
092	\	126	~	160	°
093	]	127	▒	161	¹
094	^	128	(space)	162	²
095	_	129*	dot1	163	³
096	' (open single quote)	130*	dot2	164	⁴
097	a	131*	dot12	165	⁵
098	b	132*	dot3	166	⁶
099	c	133*	dot13	167	⁷
100	d	134*	dot23	168	⁸
101	e	135*	dot123	169	⁹
102	f	136*	dot4	170	α
103	g	137*	dot14	171	β
104	h	138*	dot24	172	γ
105	i	139*	dot124	173	δ
106	j	140*	dot34	174	ε
107	k	141*	dot134	175	η
108	l	142*	dot234	176	θ
109	m	143*	dot1234	177	λ
110	n	144*	selftest3	178	π
111	o	145*	selftest4	179	ρ
112	p	146*	selftest5	180	σ
113	q	147*	selftest6	181	τ
114	r	148*	selftest7	182	φ
115	s	149*	selftest8	183	ω
116	t	150*	selftest9	184	Γ
117	u	151*	selftest10	185	Δ

* The dot patterns for this character can be found in Display character dot patterns (on page F-5).

Display character codes (decimal 186-255)

DECIMAL	DISPLAY	DECIMAL	DISPLAY	DECIMAL	DISPLAY
186	Σ	210	â	234	ü
187	Φ	211	ä	235	ú
188	∩	212	á	236	ù
189	∪	213	à	37	Ü
190	÷	214	å	238	ñ
191	≤	215	a̲	239	Ñ
192	≥	216	Ä	240	ÿ
193	≠	217	Å	241*	lightning bolt
194	≡	218	ê	242*	KI
195	≈	219	ë	243*	box3
196	∞	220	é	244*	box4
197	>>	221	è	245*	box5
198	<<	222	É	246*	box6
199	¿	223	î	247*	box7
200	¡	224	ï	248*	box8
201	¢	225	í	249*	box9
202	£	226	ì	250*	tombstone
203	¥	227	ô	251	†
204	$P_{\dagger}$	228	ö	252	↑
205	*ƒ*	229	ó	253	↓
206	Ç	230	ò	254	←
207	ç	231	o̲	255	→
208	æ	232	Ö		
209	Æ	233	û		

* The dot patterns for this character can be found in Display character dot patterns (on page F-5).

Display character dot patterns

Figure 141: Display character dot patterns (1 of 4)

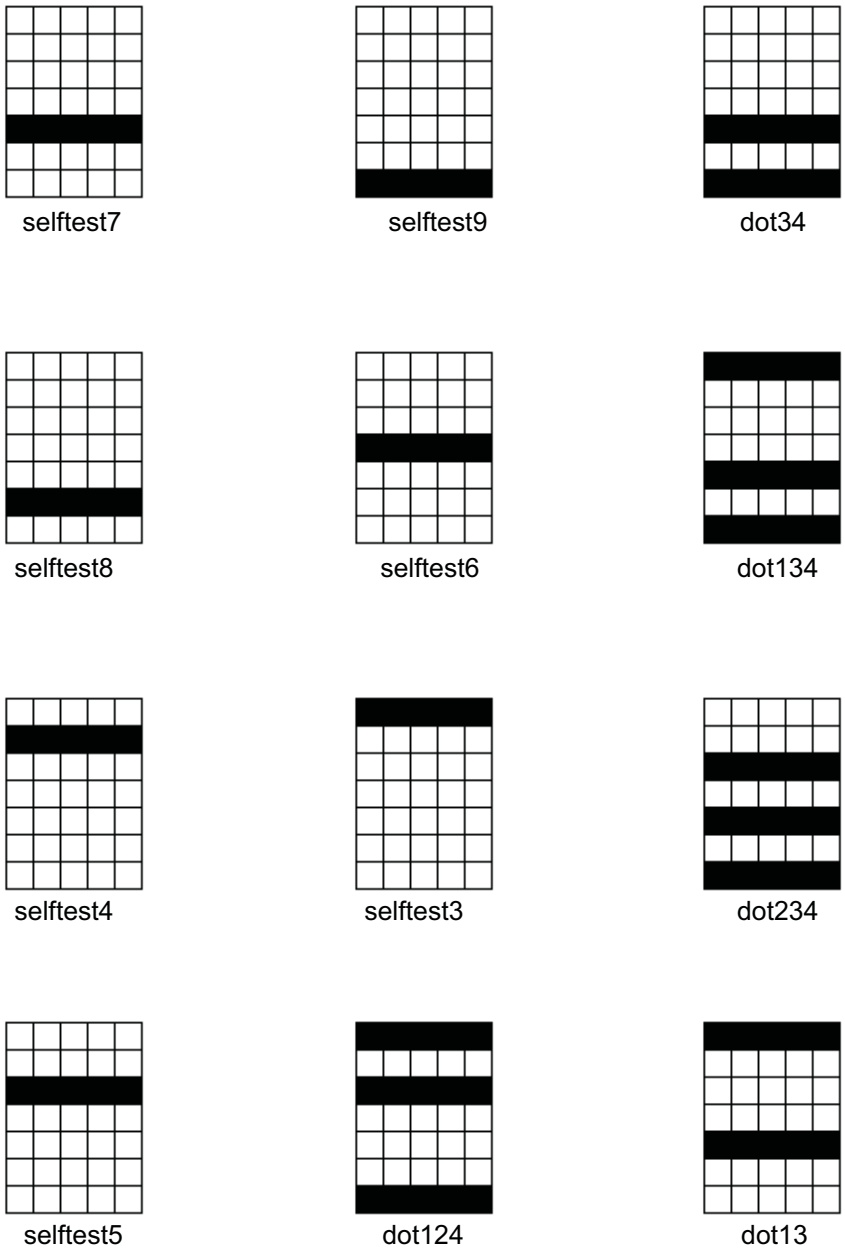


Figure 142: Display character dot patterns (2 of 4)

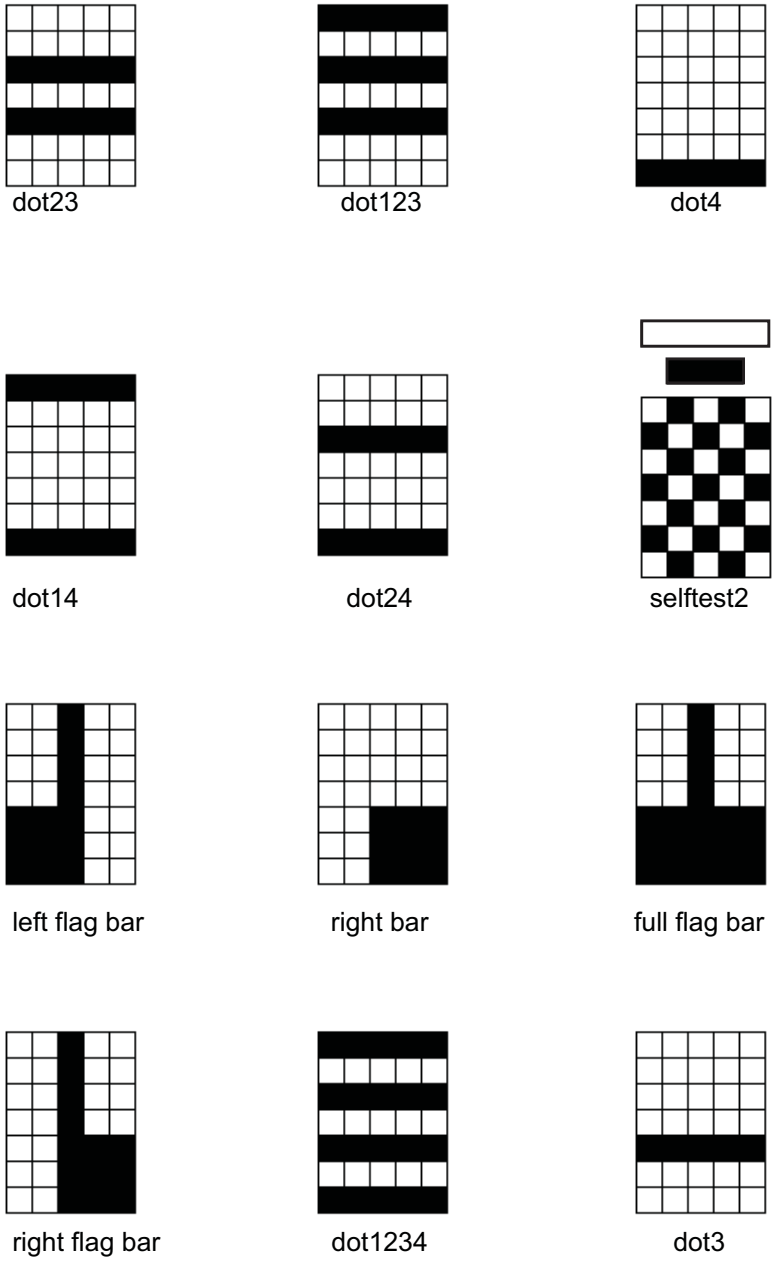


Figure 143: Display character dot patterns (3 of 4)

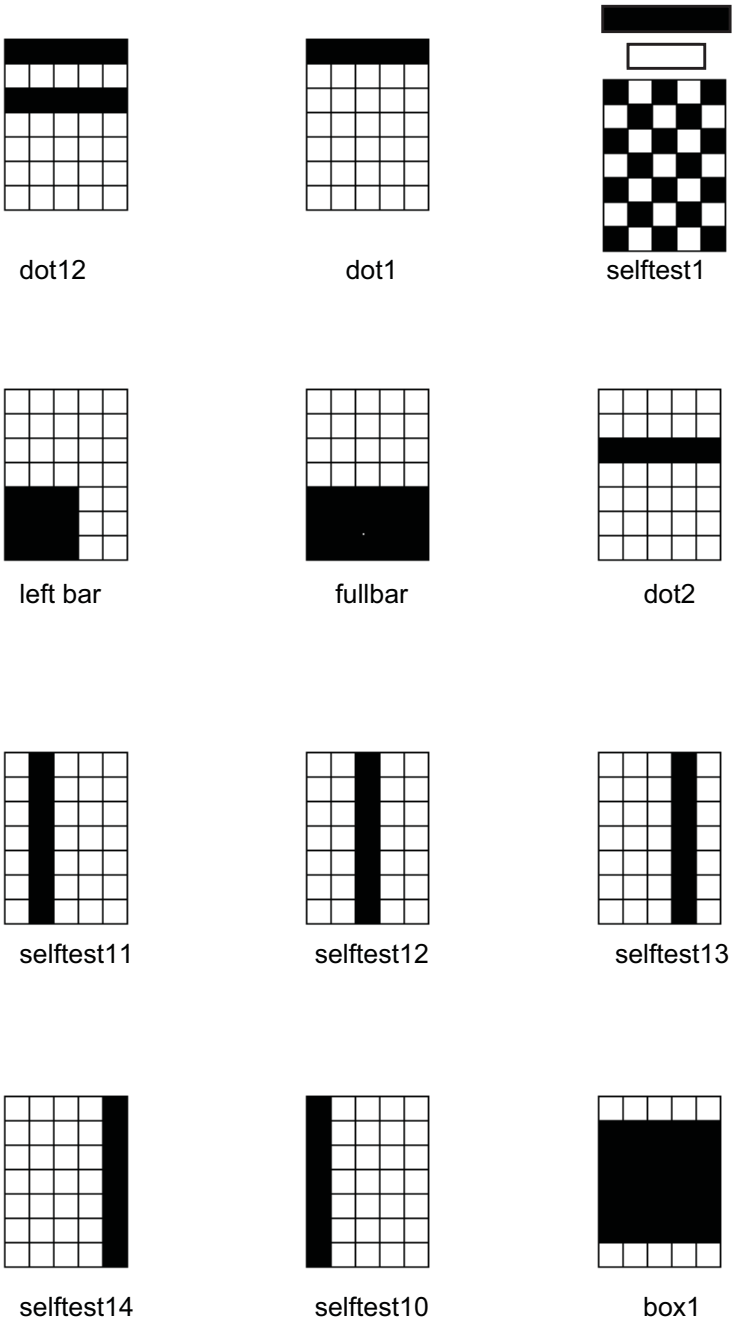
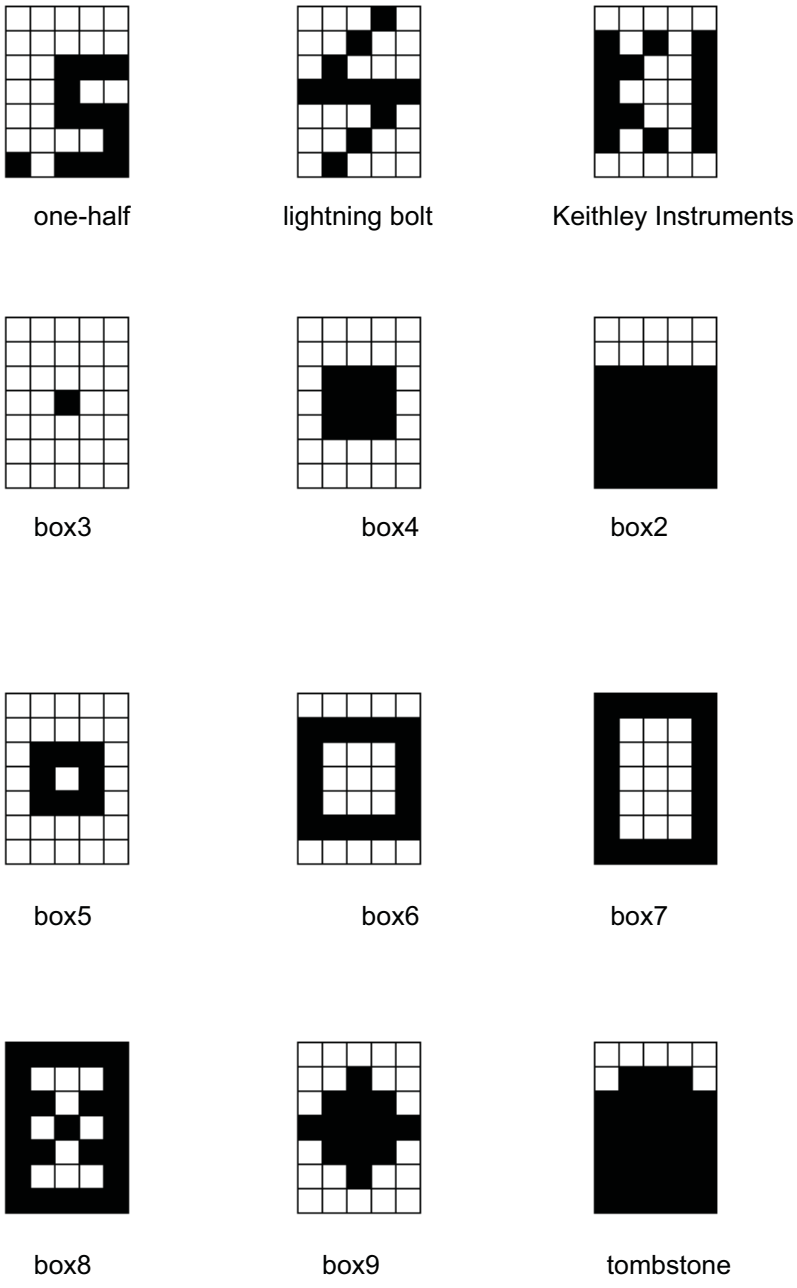


Figure 144: Display character dot patterns (4 of 4)



Index

A

annunciators • 7-55
anonymous script • 6-6
Arrays
  arrays, TSL • 6-29
assigning groups • 6-57
attribute, assigning a value to • 5-2
attributes • 5-2
  reading • 5-3
Auto ohms measurements • 2-31
Auto zero • 2-22
  Front panel • 2-23
  Front panel operation • 2-23
  NPLC caching • 2-22
Autoexec script • 6-7
autorun scripts • 6-7

B

base library functions • 6-31
beeper • 5-3, 7-9
beeper functions and attributes
  beeper.enable • 7-9
bit • See index
  Boolean value • 7-16
  field • 7-13
  index • 7-12, 7-13, 7-14, 7-15, 7-16
  toggle • 7-16
  weighted value, bit • 7-12
bit functions • 7-10, 7-11, 7-12, 7-13, 7-14, 7-15, 7-16, 7-55
bitwise • 7-10, 7-11
branching • 6-23

C

Capabilities
  Source-measure • 2-11
circuits
  Circuit configurations • 2-14, 4-14
    Basic • 2-14
    Contact check • 4-16
    Measure only (V or I) • 4-15
    Source I • 4-14
    Source V • 4-15
clear • 7-55, 7-362
  bit.clear() • See bit.clear()
command
  Command programming
    Time and date values • 7-4
  queries • 5-3
Compliance
  Limit • 4-2
connecting multiple units • 6-48, 6-49
Contact check
  Circuit configuration • 4-16
  Measurements • 2-37
  Overview • 2-37
  Programming example • 2-39
contact information • 1-1
Continuous power operating boundaries • 4-5
Current accuracy • B-14
  Output • B-6, B-14
Current measurement accuracy • B-6, B-8
cursor • 7-57, 7-70

D

Data store
  Overview • 3-6
  Programming examples • 3-17
delay functions
  delay() • 7-43
Digital I/O port • 3-65, 3-88
  +5V output • 3-84
  Bit weighting • 3-87
  Commands • 3-87
  Configuration • 3-83, 3-84
  Controlling I/O lines • 3-85
  Lines • 3-84
  Output enable • 3-88
  Programming examples • 3-88
  Remote operation • 3-87, 3-90
DIGITS • 2-70, 3-17
Display
  DISPLAY PATTERNS test • A-4
  Resolution • 2-71
  unit serial number • 1-3
Display operations
  Adding menu entries • 3-80
  Character codes • 3-74
  Clearing • 3-73
  Deleting menu entries • 3-81
  Functions and attributes • 3-70
  Indicators • 3-78

Input prompting • 3-76
Keycodes • 3-81
Key-press codes • 3-81
Load test menu • 3-79
Load test menu • 3-79
LOCAL lockout • 3-79
Measurement functions • 3-72
Menu • 3-76
Messages • 3-72
Resolution • 3-72
Running a test • 3-81
Text messages • 3-74
Triggering • 3-81

E

environmental conditions • B-2
Error messages
Effects on scripts • 8-1
Reading • 8-2
Summary • 8-1
errorqueue functions and attribute
errorqueue.clear() • 7-77
errorqueue.count • 7-77, 7-78
examples
Ohms programming example • 2-37, 2-39
passing parameter • 5-3
variable assignment • 5-3
exit functions
exit() • 7-83

F

Factory defaults
Restoring • B-4
Factory scripts
KISweep • 5-22
Modifying • 5-21
Filters • 3-3
Enabling • 3-4
Response time • 3-4
Types of • 3-3
Firmware • 9-2
Upgrading • 9-2, A-4
Using TSB for upgrade • A-4
format attributes
format.asciiprecision • 7-87
format.byteorder • 7-88
format.data • 7-88
front panel
Source-measure procedure • 2-24, 2-27
Tests • A-2
functions • 6-18
Fuse
Line, replacement • A-1

G

get • 7-12, 7-13, 7-55, 7-57, 7-59, 7-93, 7-143
gpib attribute
gpib.address • 7-95
groups
assigning • 6-57
coordinating remote • 6-58
different test scripts • 6-56
reassigning • 6-57

H

High-capacitance mode • 3-65
Enabling • 3-68
Overview • 3-65

I

ICL
general device control • 6-63
TSP-specific device control • 6-63
index • 7-12, 7-13, 7-14, 7-15, 7-16
Indicators • 3-78

K

Key-press codes • 3-81
KEYS test • A-3
key-value pairs • 5-20, 7-373, 7-374, 7-375

L

LAN
Assigning the Method • C-1
Connecting to • C-1
Domain name system • C-1
Duplex mode • C-11
MAC address • C-13
Overview • C-1
Point-to-point connection • C-1
Setting the IP address • C-1
Setting the method • C-1
Setting the subnet mask • C-1
Troubleshooting • C-1
libraries, standard • 6-30
Line
Fuse replacement • A-1
Power • B-2
Load test menu • 3-79
local group • 7-152
logical
logical AND operation • 7-10
logical OR operation • 7-10
loop control • 6-25
Low range limits • 2-67

M

MAC address • C-13
maintenance • A-1
  Fuse replacement • A-1
makegetter functions
  makegetter() • 7-141
  makesetter() • 7-141
master node • 7-152
master node overview • 6-56
math
  library functions • 6-35
measure
  V or I • 4-15
  Voltage or current • 2-27
multiple units, connecting • 6-48, 6-49

N

named scripts
  overview • 6-4
  RUN • 6-6
node
  master overview • 6-56
nonblocking • 7-55, 7-70, 7-71
NPLC caching • 2-24

O

Ohms
  Calculations • 2-31
  Measurement procedure • 2-31
  Measurements • 2-27
  Programming example • 2-35
  Remote programming • 2-34
  Sense selection • 2-34
  Sensing • 2-32
opc functions
  opc() • 7-145
Operating boundaries • 4-5
  Continuous power • 4-5
  Source or sink • 4-5
Operation
  Considerations • 2-14
Output current accuracy • B-6, B-14
  Limits • B-6
Output enable • 3-88
  Control • 3-89
Output voltage accuracy • B-13
  Limits • B-13
Overheating protection • 4-3
  equations • 4-3
overlapped operations in remote groups, coordinating • 6-58
overwrite a bit field • 7-15

P

parallel test scripts • 6-57
power
  blinking • 7-57
  Calculations • 2-36, 4-3
  Equations • 4-3
  Measurement procedure • 2-37
  Measurements • 2-35
  Programming example • 2-37
  Remote programming • 2-36
Power-on setup • 2-40, 2-41
Precedence • 6-22
print functions
  print() • 7-145
  printbuffer() • 7-146
  printnumber() • 7-147
programming
  interaction • 6-37
  script model • 6-3
Pulse energy limitations • 4-18

Q

queries • 5-3
queues • E-2, E-25
  Error • E-25

R

range
  Auto range limits • 2-67
  Available ranges • 2-65
  Commands • 2-69
  Low limits • 2-67
  Output value • B-5
  Programming • 2-69
  Programming example • 2-70
reading buffer
  removing stale values • 6-60
Reading buffers • 3-6
  Attributes • 3-15
  Defined buffer example • 3-17
  Displaying readings • 3-10
  Dual buffer example • 3-18
  Dynamically allocated • 3-16
  Dynamically allocated buffer example • 3-19
  Location number • 3-9
  Overview • 3-6
  Remote state • 3-10
  Saving • 3-8
  Status • 3-16
  Storage • 3-8
  Storage control attributes • 3-11
  Timestamp • 3-10
readings

Maximum • 2-66
Recalling • 3-9
Recommended test equipment • B-2
Recommended verification equipment • B-2
registers
Enable and transition • E-18
Measurement event • E-23
Operation event • E-20
Programming example • E-24
Questionable event • E-21
Reading • E-14
Serial polling and SRQ • E-17
Service request enable • E-16
Standard event • E-19
Status sets • E-18
System summary event • E-19
REL
Defining a value • 3-2
Enabling and disabling • 3-1
Front panel • 3-1
Remote programming • 3-2
Remote command interface
Selecting • C-14
Remote programming
Ohms • 2-34
Power • 2-36
rel • 3-2
Reset
digio trigger • 7-50
lan • 7-115
localnode • 7-138
reset • 7-152
status • 7-313
timer • 7-334
Restoring factory defaults • B-4
run-time environment
script, restoring • 6-43

S

Script Editor • 6-37
scripts
autoexec • 6-7
autorun scripts • 6-7
deleting • 6-45
function, using • 6-20
interactive • 6-3
name attribute • 7-161
named • 6-4, 6-6
parallel test, running • 6-57
restoring in run-time environment • 6-43
running • 6-6, 6-57
Script Editor • 6-37
test scripts across the TSP-Link network • 6-59
unnamed • 6-6
user • 6-3, 6-6, 6-8
Sensing
Ohms • 2-32
serial number • 1-3
serial polling • E-17
Setups
Power-on • 2-40
Sink • 4-5
Operation • 2-27
sound • 7-9
Source • 4-5
Source I measure I • 4-13
Source V measure V • 4-13
Source-measure
Capabilities • 2-11
Front panel operation • 2-24
Programming example • 2-27
Speed
Command • 2-72
Configuration menu • 2-72
Programming example • 2-72
Remote programming • 2-72
SRQ (Service Request) • E-14
standard
libraries • 6-30
standard libraries • 6-30
state • 6-60
Status byte and service request (SRQ) • E-14
Commands • E-17
status model • E-1
Clearing registers and queues • E-12
Programming registers and queues • E-12
Queues • E-2
Status byte and SRQ • E-1, E-14
Status register sets • E-2
TSP-Link system • E-26
status register sets • E-2, E-18
string library functions • 6-33
style • 7-57, 7-70
substring • 6-33
Sweep
Waveforms • 4-3

T

tables • 6-29
Telnet
Configuring • C-15
Test considerations • B-5
test scripts across the TSP-Link network • 6-59
Tests
Front panel • A-2
time • 7-168, 7-169
Timestamp • 3-10
Triggering • 9-2
Configuring attributes • 2-27
Local mode • 2-27

TSP-Link • 6-53
TSB Embedded • 6-37
TSP
installing software • 6-37
TSP-Link • 6-47
Abort • 6-53
Accessing nodes • 6-52
communicating between TSP-enabled instruments • 6-62
Initialization • 6-50
Master • 6-48
Node numbers • 6-50
Reset • 6-51
reset() command • 6-53
Slaves • 6-48
synchronization lines
Connecting to • 3-89
Digital I/O • 3-89
Remote commands • 3-90
Triggering • 6-53

U

unnamed scripts • 6-6
USB
Saving the reading buffer to • 3-9
user scripts
creating alternative • 6-3
modifying • 6-8
running • 6-6
User setups
Recalling • 2-40
Saving • 2-39
Saving from a command interface • 2-41
userstring functions • 6-56
UTC • 7-168

V

variables • 6-16
verification
Limits • B-3
Test considerations • B-5
Test equipment • B-2
Test procedures • B-4
Test requirements • B-1
Test summary • B-4
Verify menu • C-1
voltage
Accuracy • B-6
Accuracy limit • B-14
Measure • 2-27
Measurement accuracy • B-14
Measurement accuracy limits • B-14

W

waitcomplete functions
waitcomplete() • 7-375
Warm-up • 2-8
Warm-up period • B-2

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.
All other trademarks and trade names are the property of their respective companies.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY • www.keithley.com

12/06