

The buffer style is set when you create a user-defined reading buffer. The buffer style cannot be changed after the buffer has been created, so to eliminate memory problems caused by the style, you may need to delete or adjust the capacity of the buffers. Refer to [Creating buffers](#) (on page 6-5) for information on the effects of styles.

The amount of memory used by a sweep configuration is based on the number of source points. The actual memory consumption can vary greatly depending on how often the source-measure unit (SMU) take readings during the sweep, but as a general rule, each source point can be expected to consume at least 24 bytes.

It is possible for the memory used for the run-time environment, sweep configuration, and reading buffers to exceed 32 MB. When this occurs, there is a risk that memory allocation errors will occur and instrument will not execute commands as expected.

About TSP Commands

This section contains an overview of the TSP commands for the instrument. The commands are organized into groups, with a brief description of each group. Each section contains links to the detailed descriptions for each command in the TSP command reference section of this documentation (see [TSP commands](#) (on page 14-8)).

Beeper control

The beeper command allows you to sound the instrument beeper:

[beeper.beep\(\)](#) (on page 14-8)

Digital I/O

The digital I/O port of the instrument can control external circuitry (such as a component handler for binning operations).

The I/O port has 6 lines. Each line can be at TTL logic state 1 (high) or 0 (low). See the pinout diagram in [Digital I/O port configuration](#) (on page 8-14) for additional information.

You can use these commands to read and write to each individual bit, and to read and write to the entire port.

[digio.line\[N\].mode](#) (on page 14-60)
[digio.line\[N\].reset\(\)](#) (on page 14-61)
[digio.line\[N\].state](#) (on page 14-63)
[digio.readport\(\)](#) (on page 14-63)
[digio.writeport\(\)](#) (on page 14-64)

Configuration list

You can use these commands to create and recall configuration lists. A configuration list is a list of stored settings for the source or measurement function. You can restore these settings to change the active state of the instrument.

The measure configuration list commands are:

[smu.measure.configlist.catalog\(\)](#) (on page 14-128)
[smu.measure.configlist.create\(\)](#) (on page 14-129)
[smu.measure.configlist.delete\(\)](#) (on page 14-130)
[smu.measure.configlist.query\(\)](#) (on page 14-131)
[smu.measure.configlist.recall\(\)](#) (on page 14-132)
[smu.measure.configlist.size\(\)](#) (on page 14-133)
[smu.measure.configlist.store\(\)](#) (on page 14-134)

The source configuration list commands are:

[smu.source.configlist.catalog\(\)](#) (on page 14-172)
[smu.source.configlist.create\(\)](#) (on page 14-173)
[smu.source.configlist.delete\(\)](#) (on page 14-174)
[smu.source.configlist.query\(\)](#) (on page 14-175)
[smu.source.configlist.recall\(\)](#) (on page 14-176)
[smu.source.configlist.size\(\)](#) (on page 14-177)
[smu.source.configlist.store\(\)](#) (on page 14-178)

Display

The display functions and attributes allow you to change the format of information on the front-panel display of the instrument. You can also customize your display.

[display.changescreen\(\)](#) (on page 14-66)
[display.clear\(\)](#) (on page 14-67)
[display.delete\(\)](#) (on page 14-67)
[display.input.number\(\)](#) (on page 14-68)
[display.input.option\(\)](#) (on page 14-70)
[display.input.prompt\(\)](#) (on page 14-71)
[display.input.string\(\)](#) (on page 14-72)
[display.lightstate](#) (on page 14-74)
[display.prompt\(\)](#) (on page 14-75)
[display.readingformat](#) (on page 14-76)
[display.settext\(\)](#) (on page 14-77)
[display.waitevent\(\)](#) (on page 14-78)

Event log

You can use the event log to view specific details about LAN triggering events.

- [eventlog.clear\(\) \(on page 14-79\)](#)
- [eventlog.getcount\(\) \(on page 14-80\)](#)
- [eventlog.next\(\) \(on page 14-81\)](#)
- [eventlog.post\(\) \(on page 14-82\)](#)
- [eventlog.save\(\) \(on page 14-84\)](#)

File

You can use the file commands to open and close directories and files, write data, or to read a file on an installed USB flash drive.

The root folder of the USB flash drive has the absolute path:

```
" /usb1 / "
```

NOTE

You can use either the slash (/) or backslash (\) as a directory separator. However, the backslash is also used as an escape character, so if you use it as a directory separator, you will generally need to use a double backslash (\\) when you are creating scripts or sending commands to the instrument.

- [file.close\(\) \(on page 14-85\)](#)
- [file.flush\(\) \(on page 14-86\)](#)
- [file.mkdir\(\) \(on page 14-87\)](#)
- [file.open\(\) \(on page 14-87\)](#)
- [file.read\(\) \(on page 14-88\)](#)
- [file.usbdriveexists\(\) \(on page 14-89\)](#)
- [file.write\(\) \(on page 14-90\)](#)

Instrument identification

These commands store strings that describe the instrument.

- [localnode.access \(on page 14-101\)](#)
- [localnode.gettime\(\) \(on page 14-102\)](#)
- [localnode.linefreq \(on page 14-102\)](#)
- [localnode.model \(on page 14-103\)](#)
- [localnode.password \(on page 14-103\)](#)
- [localnode.prompts \(on page 14-104\)](#)
- [localnode.prompts4882 \(on page 14-105\)](#)
- [localnode.serialno \(on page 14-105\)](#)
- [localnode.settime\(\) \(on page 14-106\)](#)
- [localnode.showevents \(on page 14-107\)](#)
- [localnode.version \(on page 14-108\)](#)

Miscellaneous

The miscellaneous functions and attributes allow you to control instrument features and functions that are not category specific.

[delay\(\)](#) (on page 14-59)
[exit\(\)](#) (on page 14-85)
[localnode.linefreq](#) (on page 14-102)
[localnode.password](#) (on page 14-103)
[localnode.prompts](#) (on page 14-104)
[localnode.prompts4882](#) (on page 14-105)
[opc\(\)](#) (on page 14-110)
[waitcomplete\(\)](#) (on page 14-318)
[upgrade.previous\(\)](#) (on page 14-314)
[upgrade.unit\(\)](#) (on page 14-314)

LAN

LAN commands provide options that allow you to review and configure network settings over the remote interface.

The following commands contain the status of the LAN.

[lan.ipconfig\(\)](#) (on page 14-99)
[lan.lxidomain](#) (on page 14-100)
[lan.macaddress](#) (on page 14-100)

Set up and assert trigger events that are sent over the LAN.

[trigger.lanin\[N\].clear\(\)](#) (on page 14-230)
[trigger.lanin\[N\].edge](#) (on page 14-231)
[trigger.lanin\[N\].overrun](#) (on page 14-231)
[trigger.lanin\[N\].wait\(\)](#) (on page 14-232)
[trigger.lanout\[N\].assert\(\)](#) (on page 14-233)
[trigger.lanout\[N\].connect\(\)](#) (on page 14-234)
[trigger.lanout\[N\].connected](#) (on page 14-234)
[trigger.lanout\[N\].disconnect\(\)](#) (on page 14-235)
[trigger.lanout\[N\].ipaddress](#) (on page 14-236)
[trigger.lanout\[N\].logic](#) (on page 14-236)
[trigger.lanout\[N\].protocol](#) (on page 14-237)
[trigger.lanout\[N\].stimulus](#) (on page 14-238)

GPIB

These commands store the GPIB address and indicate whether GPIB communication is enabled.

[gpib.address](#) (on page 14-98)

Reading buffer

Reading buffers capture measurements, ranges, instrument status, and output states of the instrument.

[buffer.clearstats\(\)](#) (on page 14-9)
[buffer.delete\(\)](#) (on page 14-10)
[buffer.getstats\(\)](#) (on page 14-11)
[buffer.make\(\)](#) (on page 14-14)
[buffer.save\(\)](#) (on page 14-19)
[buffer.saveappend\(\)](#) (on page 14-20)
[bufferVar.capacity](#) (on page 14-23)
[bufferVar.clear\(\)](#) (on page 14-24)
[bufferVar.dates](#) (on page 14-25)
[bufferVar.endindex](#) (on page 14-26)
[bufferVar.fillmode](#) (on page 14-31)
[bufferVar.formattedreadings](#) (on page 14-32)
[bufferVar.fractionalseconds](#) (on page 14-33)
[bufferVar.logstate](#) (on page 14-34)
[bufferVar.n](#) (on page 14-35)
[bufferVar.readings](#) (on page 14-36)
[bufferVar.relativetimesamps](#) (on page 14-37)
[bufferVar.seconds](#) (on page 14-38)
[bufferVar.sourceformattedvalues](#) (on page 14-39)
[bufferVar.sourcestatuses](#) (on page 14-40)
[bufferVar.sourceunits](#) (on page 14-41)
[bufferVar.sourcevalues](#) (on page 14-42)
[bufferVar.startindex](#) (on page 14-43)
[bufferVar.statuses](#) (on page 14-44)
[bufferVar.times](#) (on page 14-46)
[bufferVar.timestamps](#) (on page 14-47)
[bufferVar.units](#) (on page 14-48)
[buffer.write.format\(\)](#) (on page 14-50)
[buffer.write.reading\(\)](#) (on page 14-52)
[printbuffer\(\)](#) (on page 14-112)

Reset

Resets settings to their default settings.

[digio.line\[N\].reset\(\)](#) (on page 14-61)
[reset\(\)](#) (on page 14-116)
[smu.reset\(\)](#) (on page 14-170)
[timer.cleartime\(\)](#) (on page 14-216)
[trigger.blender\[N\].reset\(\)](#) (on page 14-219)
[trigger.timer\[N\].reset\(\)](#) (on page 14-283)
[tsplink.line\[N\].reset\(\)](#) (on page 14-298)
[tspnet.reset\(\)](#) (on page 14-308)

Queries and response messages

You can use the `print()`, `printbuffer()`, and `printnumber()` functions to query the instrument and generate response messages. The format attributes control how the data is formatted for the print functions used.

The `localnode` commands determine if generated errors are automatically sent and if prompts are generated.

[format.asciiprecision](#) (on page 14-90)
[format.byteorder](#) (on page 14-91)
[format.data](#) (on page 14-92)
[localnode.serialno](#) (on page 14-105)
[localnode.settime\(\)](#) (on page 14-106)
[localnode.showevents](#) (on page 14-107)
[localnode.version](#) (on page 14-108)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[printnumber\(\)](#) (on page 14-115)

Scripting

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument efficiently. These commands describe how to create, load, modify, run, and exit scripts.

For detail on using scripts, see [Fundamentals of scripting for TSP](#) (on page 13-4).

[createconfigscript\(\)](#) (on page 14-54)
[exit\(\)](#) (on page 14-85)
[script.delete\(\)](#) (on page 14-117)
[script.load\(\)](#) (on page 14-118)
[scriptVar.run\(\)](#) (on page 14-118)
[scriptVar.save\(\)](#) (on page 14-119)
[scriptVar.source](#) (on page 14-120)

SMU

SMU commands and functions are unique to SMU instruments. You can use the SMU commands to control the instrument source and measure functions.

[localnode.linefreq](#) (on page 14-102)
[smu.interlock.enable](#) (on page 14-120)
[smu.interlock.tripped](#) (on page 14-122)
[smu.terminals](#) (on page 14-203)

SMU measure commands are:

[smu.measure.autorange](#) (on page 14-123)
[smu.measure.autorangehigh](#) (on page 14-124)
[smu.measure.autorangelow](#) (on page 14-125)
[smu.measure.autozero.enable](#) (on page 14-127)
[smu.measure.autozero.once\(\)](#) (on page 14-128)
[smu.measure.configlist.catalog\(\)](#) (on page 14-128)
[smu.measure.configlist.create\(\)](#) (on page 14-129)
[smu.measure.configlist.delete\(\)](#) (on page 14-130)
[smu.measure.configlist.query\(\)](#) (on page 14-131)
[smu.measure.configlist.recall\(\)](#) (on page 14-132)
[smu.measure.configlist.size\(\)](#) (on page 14-133)
[smu.measure.configlist.store\(\)](#) (on page 14-134)
[smu.measure.count](#) (on page 14-136)
[smu.measure.displaydigits](#) (on page 14-139)
[smu.measure.filter.count](#) (on page 14-140)
[smu.measure.filter.enable](#) (on page 14-141)
[smu.measure.filter.type](#) (on page 14-142)
[smu.measure.func](#) (on page 14-143)
[smu.measure.limit\[Y\].audible](#) (on page 14-145)
[smu.measure.limit\[Y\].autoclear](#) (on page 14-146)
[smu.measure.limit\[Y\].clear\(\)](#) (on page 14-147)
[smu.measure.limit\[Y\].enable](#) (on page 14-147)
[smu.measure.limit\[Y\].fail](#) (on page 14-148)
[smu.measure.limit\[Y\].high.value](#) (on page 14-150)
[smu.measure.limit\[Y\].low.value](#) (on page 14-151)
[smu.measure.math.enable](#) (on page 14-152)
[smu.measure.math.format](#) (on page 14-153)
[smu.measure.math.mxb.bfactor](#) (on page 14-154)
[smu.measure.math.mxb.mfactor](#) (on page 14-155)
[smu.measure.math.percent](#) (on page 14-156)
[smu.measure.nplc](#) (on page 14-157)
[smu.measure.offsetcompensation](#) (on page 14-158)
[smu.measure.range](#) (on page 14-159)
[smu.measure.read\(\)](#) (on page 14-160)
[smu.measure.readwithtime\(\)](#) (on page 14-161)
[smu.measure.rel.acquire\(\)](#) (on page 14-162)
[smu.measure.rel.enable](#) (on page 14-163)
[smu.measure.rel.level](#) (on page 14-164)
[smu.measure.sense](#) (on page 14-165)
[smu.terminals](#) (on page 14-203)
[smu.measure.unit](#) (on page 14-168)
[smu.measure.userdelay\[N\]](#) (on page 14-169)

SMU source commands are:

[smu.source.autorange](#) (on page 14-170)
[smu.source.autodelay](#) (on page 14-171)
[smu.source.configlist.catalog\(\)](#) (on page 14-172)
[smu.source.configlist.create\(\)](#) (on page 14-173)
[smu.source.configlist.delete\(\)](#) (on page 14-174)
[smu.source.configlist.query\(\)](#) (on page 14-175)
[smu.source.configlist.recall\(\)](#) (on page 14-176)
[smu.source.configlist.size\(\)](#) (on page 14-177)
[smu.source.configlist.store\(\)](#) (on page 14-178)
[smu.source.delay](#) (on page 14-180)
[smu.source.func](#) (on page 14-181)
[smu.source.hghc](#) (on page 14-182)
[smu.source.level](#) (on page 14-183)
[smu.source.offmode](#) (on page 14-184)
[smu.source.output](#) (on page 14-186)
[smu.source.protect.level](#) (on page 14-187)
[smu.source.protect.tripped](#) (on page 14-188)
[smu.source.range](#) (on page 14-188)
[smu.source.readback](#) (on page 14-190)
[smu.source.sweeplinear\(\)](#) (on page 14-192)
[smu.source.sweeplinearstep\(\)](#) (on page 14-194)
[smu.source.sweeplist\(\)](#) (on page 14-196)
[smu.source.sweeplog\(\)](#) (on page 14-198)
[smu.source.userdelay\[N\]](#) (on page 14-200)
[smu.source.xlimit.level](#) (on page 14-201)
[smu.source.xlimit.tripped](#) (on page 14-202)

Status model

The status model is a set of status registers and queues. You can use the following commands to manipulate and monitor these registers and queues to view and control various instrument events.

[status.clear\(\)](#) (on page 14-204)
[status.condition](#) (on page 14-204)
[status.operation.condition](#) (on page 14-205)
[status.operation.enable](#) (on page 14-206)
[status.operation.event](#) (on page 14-206)
[status.operation.getmap\(\)](#) (on page 14-207)
[status.operation.setmap\(\)](#) (on page 14-208)
[status.preset\(\)](#) (on page 14-209)
[status.questionable.condition](#) (on page 14-209)
[status.questionable.enable](#) (on page 14-210)
[status.questionable.event](#) (on page 14-210)
[status.questionable.getmap\(\)](#) (on page 14-211)
[status.questionable.setmap\(\)](#) (on page 14-212)
[status.request_enable](#) (on page 14-213)
[status.standard.enable](#) (on page 14-214)
[status.standard.event](#) (on page 14-215)

Time

[bufferVar.fractionalseconds](#) (on page 14-33)
[bufferVar.n](#) (on page 14-35)
[bufferVar.relativetimesamps](#) (on page 14-37)
[bufferVar.seconds](#) (on page 14-38)
[bufferVar.times](#) (on page 14-46)
[bufferVar.timestamps](#) (on page 14-47)
[delay\(\)](#) (on page 14-59)
[localnode.gettime\(\)](#) (on page 14-102)
[timer.cleartime\(\)](#) (on page 14-216)
[timer_gettime\(\)](#) (on page 14-217)
[trigger.timer\[N\].clear\(\)](#) (on page 14-279)
[trigger.timer\[N\].count](#) (on page 14-279)
[trigger.timer\[N\].delay](#) (on page 14-281)
[trigger.timer\[N\].delaylist](#) (on page 14-281)
[trigger.timer\[N\].reset\(\)](#) (on page 14-283)
[trigger.timer\[N\].start.fractionalseconds](#) (on page 14-284)
[trigger.timer\[N\].start.generate](#) (on page 14-284)
[trigger.timer\[N\].start.overrun](#) (on page 14-285)
[trigger.timer\[N\].start.seconds](#) (on page 14-286)
[trigger.timer\[N\].start.stimulus](#) (on page 14-286)
[trigger.timer\[N\].wait\(\)](#) (on page 14-288)
[tspnet.timeout](#) (on page 14-310)

Triggering

The triggering commands allow you to set the conditions that the instrument uses to determine when measurements are captured.

[trigger.blender\[N\].clear\(\)](#) (on page 14-217)
[trigger.blender\[N\].orenable](#) (on page 14-218)
[trigger.blender\[N\].overrun](#) (on page 14-218)
[trigger.blender\[N\].reset\(\)](#) (on page 14-219)
[trigger.blender\[N\].stimulus\[M\]](#) (on page 14-220)
[trigger.blender\[N\].wait\(\)](#) (on page 14-221)
[trigger.clear\(\)](#) (on page 14-222)
[trigger.digin\[N\].clear\(\)](#) (on page 14-223)
[trigger.digin\[N\].edge](#) (on page 14-224)
[trigger.digin\[N\].overrun](#) (on page 14-225)
[trigger.digin\[N\].wait\(\)](#) (on page 14-225)
[trigger.digout\[N\].assert\(\)](#) (on page 14-226)
[trigger.digout\[N\].logic](#) (on page 14-227)
[trigger.digout\[N\].pulsewidth](#) (on page 14-228)
[trigger.digout\[N\].release\(\)](#) (on page 14-228)
[trigger.digout\[N\].stimulus](#) (on page 14-229)

[trigger.lanin\[N\].clear\(\)](#) (on page 14-230)
[trigger.lanin\[N\].edge](#) (on page 14-231)
[trigger.lanin\[N\].overrun](#) (on page 14-231)
[trigger.lanin\[N\].wait\(\)](#) (on page 14-232)
[trigger.lanout\[N\].assert\(\)](#) (on page 14-233)
[trigger.lanout\[N\].connect\(\)](#) (on page 14-234)
[trigger.lanout\[N\].connected](#) (on page 14-234)
[trigger.lanout\[N\].disconnect\(\)](#) (on page 14-235)
[trigger.lanout\[N\].ipaddress](#) (on page 14-236)
[trigger.lanout\[N\].logic](#) (on page 14-236)
[trigger.lanout\[N\].protocol](#) (on page 14-237)
[trigger.lanout\[N\].stimulus](#) (on page 14-238)
[trigger.timer\[N\].clear\(\)](#) (on page 14-279)
[trigger.timer\[N\].count](#) (on page 14-279)
[trigger.timer\[N\].delay](#) (on page 14-281)
[trigger.timer\[N\].delaylist](#) (on page 14-281)
[trigger.timer\[N\].enable](#) (on page 14-282)
[trigger.timer\[N\].reset\(\)](#) (on page 14-283)
[trigger.timer\[N\].start.fractionalseconds](#) (on page 14-284)
[trigger.timer\[N\].start.generate](#) (on page 14-284)
[trigger.timer\[N\].start.overrun](#) (on page 14-285)
[trigger.timer\[N\].start.seconds](#) (on page 14-286)
[trigger.timer\[N\].start.stimulus](#) (on page 14-286)
[trigger.timer\[N\].wait\(\)](#) (on page 14-288)
[trigger.tsplinkin\[N\].clear\(\)](#) (on page 14-288)
[trigger.tsplinkin\[N\].edge](#) (on page 14-289)
[trigger.tsplinkin\[N\].overrun](#) (on page 14-290)
[trigger.tsplinkin\[N\].wait\(\)](#) (on page 14-290)
[trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-291)
[trigger.tsplinkout\[N\].logic](#) (on page 14-292)
[trigger.tsplinkout\[N\].pulsewidth](#) (on page 14-292)
[trigger.tsplinkout\[N\].release\(\)](#) (on page 14-293)
[trigger.tsplinkout\[N\].stimulus](#) (on page 14-294)
[trigger.wait\(\)](#) (on page 14-295)

Trigger model

The trigger model commands allow you to control and set the trigger model options.

[trigger.model.abort\(\)](#) (on page 14-239)
[trigger.model.getblocklist\(\)](#) (on page 14-239)
[trigger.model.getbranchcount\(\)](#) (on page 14-240)
[trigger.model.initiate\(\)](#) (on page 14-241)
[trigger.model.load\(\) — Config List](#) (on page 14-241)
[trigger.model.load\(\) — Duration Loop](#) (on page 14-242)
[trigger.model.load\(\) — Empty](#) (on page 14-243)
[trigger.model.load\(\) — GradeBinning](#) (on page 14-244)
[trigger.model.load\(\) — LogicTrigger](#) (on page 14-245)
[trigger.model.load\(\) — LoopUntilEvent](#) (on page 14-246)

[trigger.model.load\(\) — SimpleLoop](#) (on page 14-248)
[trigger.model.load\(\) — SortBinning](#) (on page 14-250)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_ALWAYS](#) (on page 14-253)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_COUNTER](#) (on page 14-254)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_DELTA](#) (on page 14-255)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_LIMIT_CONSTANT](#) (on page 14-256)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC](#) (on page 14-257)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_ON_EVENT](#) (on page 14-259)
[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_ONCE](#) (on page 14-260)
[trigger.model.setblock\(\) — trigger.BLOCK_BUFFER_CLEAR](#) (on page 14-261)
[trigger.model.setblock\(\) — trigger.BLOCK_CONFIG_NEXT](#) (on page 14-262)
[trigger.model.setblock\(\) — trigger.BLOCK_CONFIG_PREV](#) (on page 14-264)
[trigger.model.setblock\(\) — trigger.BLOCK_CONFIG_RECALL](#) (on page 14-265)
[trigger.model.setblock\(\) — trigger.BLOCK_DELAY_CONSTANT](#) (on page 14-267)
[trigger.model.setblock\(\) — trigger.BLOCK_DELAY_DYNAMIC](#) (on page 14-267)
[trigger.model.setblock\(\) — trigger.BLOCK_DIGITAL_IO](#) (on page 14-268)
[trigger.model.setblock\(\) — trigger.BLOCK_LOG_EVENT](#) (on page 14-269)
[trigger.model.setblock\(\) — trigger.BLOCK_MEASURE_DIGITIZE](#) (on page 14-270)
[trigger.model.setblock\(\) — trigger.BLOCK_NOP](#) (on page 14-272)
[trigger.model.setblock\(\) — trigger.BLOCK_NOTIFY](#) (on page 14-273)
[trigger.model.setblock\(\) — trigger.BLOCK_RESET_BRANCH_COUNT](#) (on page 14-274)
[trigger.model.setblock\(\) — trigger.BLOCK_SOURCE_OUTPUT](#) (on page 14-275)
[trigger.model.setblock\(\) — trigger.BLOCK_WAIT](#) (on page 14-275)
[trigger.model.state\(\)](#) (on page 14-278)

TSP-Link

TSP-link commands allow you to control and access the TSP-Link synchronization port. Use these commands to perform basic steady-state digital I/O operations; for example, you can program the instrument to read and write to a specific TSP-Link synchronization line or to the entire port.

[trigger.tsplinkin\[N\].edge](#) (on page 14-289)
[trigger.tsplinkin\[N\].overrun](#) (on page 14-290)
[trigger.tsplinkin\[N\].wait\(\)](#) (on page 14-290)
[trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-291)
[trigger.tsplinkout\[N\].logic](#) (on page 14-292)
[trigger.tsplinkout\[N\].pulsewidth](#) (on page 14-292)
[trigger.tsplinkout\[N\].release\(\)](#) (on page 14-293)
[trigger.tsplinkout\[N\].stimulus](#) (on page 14-294)
[tsplink.group](#) (on page 14-296)
[tsplink.initialize\(\)](#) (on page 14-296)
[tsplink.line\[N\].mode](#) (on page 14-297)
[tsplink.line\[N\].reset\(\)](#) (on page 14-298)
[tsplink.line\[N\].state](#) (on page 14-299)
[tsplink.master](#) (on page 14-300)
[tsplink.node](#) (on page 14-300)
[tsplink.readport\(\)](#) (on page 14-301)
[tsplink.state](#) (on page 14-301)
[tsplink.writeport\(\)](#) (on page 14-302)

TSP-net

TSP-Net provides a simple socket-like programming interface to Test Script Processor (TSP) enabled instruments.

[tspnet.clear\(\)](#) (on page 14-302)
[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.disconnect\(\)](#) (on page 14-304)
[tspnet.execute\(\)](#) (on page 14-305)
[tspnet.idn\(\)](#) (on page 14-306)
[tspnet.read\(\)](#) (on page 14-307)
[tspnet.readavailable\(\)](#) (on page 14-308)
[tspnet.reset\(\)](#) (on page 14-308)
[tspnet.termination\(\)](#) (on page 14-309)
[tspnet.timeout](#) (on page 14-310)
[tspnet.tsp.abort\(\)](#) (on page 14-310)
[tspnet.tsp.abortonconnect](#) (on page 14-311)
[tspnet.tsp.rbtabcopy\(\)](#) (on page 14-312)
[tspnet.tsp.runscript\(\)](#) (on page 14-312)
[tspnet.write\(\)](#) (on page 14-313)

User strings

The `userstring` functions allow you to add user-defined strings to nonvolatile memory, delete the strings and see a list of the user strings in memory.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

[userstring.add\(\)](#) (on page 14-315)
[userstring.catalog\(\)](#) (on page 14-316)
[userstring.delete\(\)](#) (on page 14-316)
[userstring.get\(\)](#) (on page 14-317)

Section 14

TSP command reference

In this section:

TSP command programming notes.....	14-1
Using the TSP command reference	14-4
TSP commands.....	14-8

TSP command programming notes

This section contains general information about using TSP commands.

TSP syntax rules

This section provides rules that explain what you can and cannot do when entering TSP commands.

Upper and lower case

Instrument commands are case sensitive.

Function and attribute names are in lowercase characters.

Parameters and attribute constants can use a combination of lowercase and uppercase characters.
The correct case for a specific command is shown in its command description.

The following example shows the beeper .beep() function, where 2 is the duration in seconds and 2400 is the frequency. Note that the function is in lowercase characters:

```
beeper.beep(2, 2400)
```

The following command changes the display light state to be at level 50. Note that the attribute (display.lightstate) is lower case, but the constant (display.STATE_LCD_50) is a combination of lowercase and uppercase characters:

```
display.lightstate = display.STATE_LCD_50
```

White space

You can send commands with or without white spaces.

For example, the following functions, which set the length and frequency of the instrument beeper, are equivalent:

```
beeper.beep(2,2400)  
beeper.beep(2, 2400)
```

Parameters for functions

All functions must have a set of parentheses () immediately following the function. If there are parameters for the function, they are placed between the parentheses. The parentheses are required even when no parameters are specified.

The following example shows the beeper.beep() function, where 2 is the duration in seconds and 2400 is the frequency. Note that the parameters are inside the parentheses:

```
beeper.beep(2, 2400)
```

The command below resets commands to their default values (no parameters are needed):

```
reset()
```

Multiple parameters

Multiple parameters must be separated by commas.

For example, the following commands set the beeper to emit a double-beep at 2400 Hz, with a beep sequence of 0.5 s on, a delay of 0.25 s, and then 0.5 s on:

```
beeper.beep(0.5, 2400)  
delay(0.250)  
beeper.beep(0.5, 2400)
```

Time and date values

Time and date values are represented as the number of seconds since some base. The time bases are:

- **UTC 12:00 am Jan 1, 1970:** Some examples of UTC time are reading buffer timestamps, calibration adjustment and verification dates, and the value returned by `os.time()`.
- **Event:** Time referenced to an event, such as the first reading stored in a reading buffer.

Local and remote control

The instrument can be controlled locally or remotely.

When the instrument is controlled locally, you operate the instrument using the front-panel controls.

When it is controlled remotely, you operate the instrument through a controller (usually a computer).

When the instrument is first powered on, it is controlled locally.

The type of control is displayed on the front panel. When the instrument is in local control, the REMOTE LED indicator in the upper right corner is off and the control indicator on the upper left of the screen shows Local.

When the instrument is in remote control, the front-panel REMOTE LED indicator is on and the control indicator at the top left of the screen shows the type of communication interface.

Remote control

When the instrument is controlled remotely, the front-panel controls are disabled. You can still view information on the front-panel display and move between the screens using the keys and touchscreen controls. If you change a selection, however, you are prompted to switch control to local.

The OUTPUT ON/OFF switch is always active. If you press it when the instrument is controlled remotely, the instrument turns the output off (if it is on) and switches to local control.

To switch to remote control:

- Send a command from the computer to the instrument.
- Open communications between the instrument and Test Script Builder.

Local control

To change to local control, you can:

- Choose an option from the screens and try to change the value; select **Yes** on the dialog box that is displayed.
- Send the logout command from the computer.
- Turn the instrument off and on.

Using the TSP command reference

The TSP command reference contains detailed descriptions of each of the TSP commands that you can use to control your instrument. Each command description is broken into subsections. The figure below shows an example of a command description.

Figure 149: Example instrument command description

feature.enable							
This command is an example of a typical TSP command that turns an instrument feature on or off.							
Type	TSP-Link accessible	Affected by	Where saved	Default value			
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	feature.OFF			
Usage							
<pre>state = feature.enable feature.enable = state</pre>							
<code>state</code>		Disable the example feature: <code>feature.OFF</code> Enable the example feature: <code>feature.ON</code>					
Details							
This command is an example of a typical TSP command that enables or disables a feature.							
Example							
<code>feature.enable = feature.ON</code>		Enables the example feature.					
Also see							
exampleUnit.enable (on page 1-73)							

The subsections contain information about the command. The subsections are:

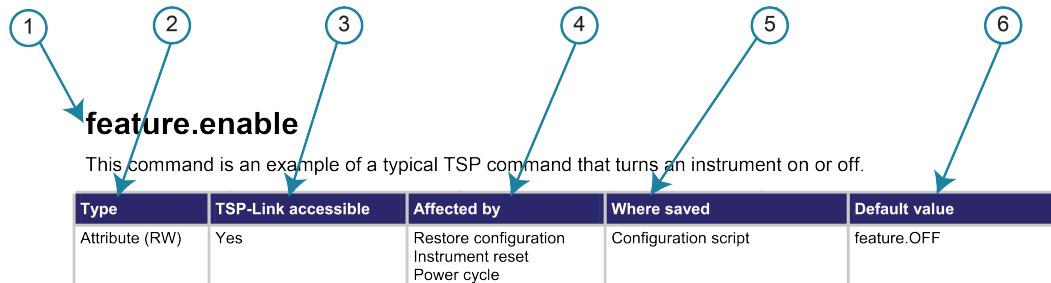
- Command name, brief description, and summary table
- Usage
- Details
- Example
- Also see

The content of each of these subsections is described in the following topics.

Command name, brief description, and summary table

Each instrument command description starts with the command name, followed by a brief description and a table with information for each command. Descriptions of the numbered items in the figure below are provided below.

Figure 150: TSP command name and summary table



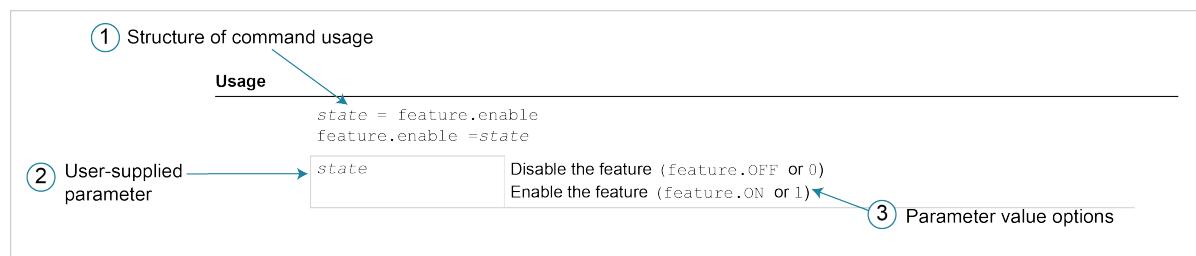
- 1 **Instrument command name.** The beginning of the command description. It is followed by a brief description of what the command does.
- 2 **Type of command.** Commands can be functions, attributes, or constants. If the command is an attribute, it can be read-only (R), read-write (RW), or write-only (W).
- 3 **TSP-Link accessible.** Indicates whether or not the command can be accessed through a TSP-Link network (Yes or No).
- 4 **Affected by.** This column lists commands or actions that can change the value of the command, including:
 - **Power cycle:** The command settings are not saved through a power cycle.
 - **Restore configuration:** If you restore a configuration script, this setting changes to the stored setting.
 - **Instrument reset:** When you reset the instrument, this command is reset to its default value. Reset can be done from the front panel or when you send `reset()` or `*RST`.
 - **Source configuration list:** If you recall a source configuration list, this setting changes to the setting stored in the list.
 - **Measure configuration list:** If you recall a measure configuration list, this setting changes to the setting stored in the list.
 - **Function:** This command changes value when the function is changed (for example, changing from a voltage source to a current source or changing from a current measurement to a resistance measurement).

- 5 **Where saved.** Indicates where the command settings reside once they are used on an instrument. Options include:
 - **Not saved:** Command is not saved and must be typed each time you use it.
 - **Nonvolatile memory:** The command is stored in a storage area in the instrument where information is saved even when the instrument is turned off.
 - **Configuration script:** Command is saved as part of the configuration script.
 - **Source configuration list:** This command is stored in source configuration lists.
 - **Measure configuration list:** This command is stored in measure configuration lists.
- 6 **Default value:** Lists the default value or constant for the command.

Command usage

The Usage section of the remote command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage. All possible command usage options are shown.

Figure 151: TSP usage description



- 1 **Structure of command usage:** Shows how the parts of the command should be organized. If a parameter is shown to the left of the command, it is the return when you print the command. Information to the right is the parameters or other items you need to enter when setting the command.
- 2 **User-supplied parameters:** Indicated by italics. For example, for the function `beeper.beep(duration, frequency)`, replace *duration* with the number of seconds and *frequency* with the frequency of the tone. Send `beeper.beep(2, 2400)` to generate a two-second, 2400 Hz tone.

Some commands have optional parameters. If there are optional parameters, they must be entered in the order presented in the Usage section. You cannot leave out any parameters that precede the optional parameter. Optional parameters are shown as separate lines in usage, presented in the required order with each valid permutation of the optional parameters.

For example:

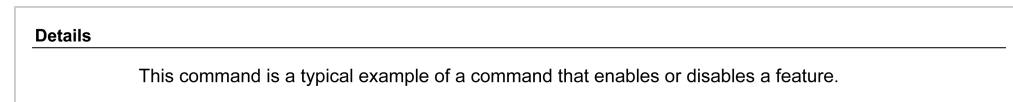
```
printbuffer(startIndex, endIndex, buffer1)
printbuffer(startIndex, endIndex, buffer1, buffer2)
```

- 3 **Parameter value options:** Descriptions of the options that are available for the user-defined parameter.

Command details

This section lists additional information you need to know to successfully use the remote command.

Figure 152: TSP Details description



Example section

The Example section of the remote command description shows examples of how you can use the command.

Figure 153: TSP example code



- 1 Actual example code that you can copy from this table and paste into your own programming application.
- 2 Description of the code and what it does. This may also contain example output of the code.

Related commands and information

The Also see section of the remote command description lists additional commands or sections that are related to the command.

Figure 154: TSP Also see description



TSP commands

The TSP commands available for the instrument are listed in alphabetic order.

available()

This function checks for the presence of specific instrument functionality.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
presence = available(functionality)
```

<i>presence</i>	If the functionality is present, returns <code>true</code> ; if not, returns <code>false</code>
<i>functionality</i>	<p>The functionality to check for:</p> <ul style="list-style-type: none"> ▪ Digital I/O: <code>digio</code> ▪ GPIB communications: <code>gpib</code> ▪ TSP-Link: <code>tsplink</code>

Example

```
print(available(gpib))
```

Returns `true` if GPIB communications are available. Returns `false` if GPIB communications are not available.

Also see

None

beeper.beep()

This function generates an audible tone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

<i>duration</i>	The amount of time to play the tone (0.001 s to 100 s)
<i>frequency</i>	The frequency of the beep (20 Hz to 8000 Hz)

Details

You can use the beeper of the instrument to provide an audible signal at a specific frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

Using this function from a remote interface does not affect audible errors or key click settings that were made from the 2450 front panel.

Example

```
beeper.beep(2, 2400)
```

Generates a 2 s, 2400 Hz tone.

Also see

None

buffer.clearstats()

This function clears the statistical information associated with the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.clearstats()  
buffer.clearstats(bufferVar)
```

bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1 if not specified
-----------	--

Details

This command clears the statistics without clearing the readings.

Example

```
buffer.clearstats()  
buffer.clearstats(testData)
```

Clears statistics for defbuffer1.
Clears statistics for testData.

Also see

[buffer.getstats\(\)](#) (on page 14-11)

buffer.delete()

This function deletes a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.delete(bufferName)
```

bufferName	The name of a user-defined reading buffer
------------	---

Details

You cannot delete the default reading buffers, defbuffer1 and defbuffer2.

After deleting the buffer, call `collectgarbage()` to reclaim the memory the buffer was using.

Example

```
buf400 = buffer.make(400)
smu.measure.read(buf400)
printbuffer(1, buf400.n, buf400.relativetimes)
buffer.delete(buf400)
collectgarbage()

Create a 400-element reading buffer named buf400.
Make measurements and store the readings in buf400.
Print the relative timestamps for each reading in the buffer.
Example output, assuming five readings are stored in the buffer:
0, 0.412850017, 0.821640085, 1.230558058, 1.629523236
Delete buf400.
Use collectgarbage() to unallocate the buffer.
```

Also see

[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

buffer.getstats()

This function returns statistics from a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
statsVar = buffer.getstats()
statsVar = buffer.getstats(bufferVar)
statsVar = buffer.getstats(bufferVar, absStartTime, absStartFractional, absEndTime,
                           absEndFractional)
statsVar = buffer.getstats(bufferVar, relStartTime, relEndTime)
```

<i>statsVar</i>	A table that contains the entries for buffer statistics; see Details for information on the entries
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>
<i>absStartTime</i>	An integer that represents the absolute start time in seconds
<i>absStartFractional</i>	An integer that represents the portion of the absolute start time that is in fractional seconds
<i>absEndTime</i>	An integer that represents the absolute end time in seconds
<i>absEndFractional</i>	An integer that represents the portion of the absolute end time that is in fractional seconds
<i>relStartTime</i>	The start time in seconds relative to the start time of the data in the buffer
<i>relEndTime</i>	The end time in seconds relative to the start time of the data in the buffer

Details

This function returns a table with statistical data about the data that was placed in the reading buffer.

The instrument automatically updates reading buffer statistics as data is added to the reading buffer.

When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make.

The table returned from this function provides statistics at the time the function is called. Although the instrument continues to update the statistics, the table that is returned is not updated. To get fresh statistics, call this function again.

The *statsVar* parameter contains the values described in the following table.

Attribute	When returned	Description
min	$n > 0$	A table that contains data about the minimum reading value that was added to the buffer; the table includes: <ul style="list-style-type: none"> ■ <i>reading</i>: The reading value ■ <i>timestamp</i>: The timestamp of the minimum data point in the buffer ■ <i>seconds</i>: The time in seconds ■ <i>fractionalseconds</i>: The time in fractional seconds
mean	$n > 0$	The average of all readings added to the buffer
stddev	$n > 1$	The standard deviation of all readings that were added to the buffer
n	Always	The number of data points on which the statistics are based
max	$n > 0$	A table that contains data about the maximum reading value that was added to the buffer; the table includes: <ul style="list-style-type: none"> ■ <i>reading</i>: The reading value ■ <i>timestamp</i>: The timestamp of the maximum data point in the buffer ■ <i>seconds</i>: The time in seconds ■ <i>fractionalseconds</i>: The fractional seconds

If *n* equals zero (0), all other values are nil. If *n* equals 1, *stddev* is nil because the standard deviation of a sample size of 1 is undefined.

Use the following command to get values from *statsVar*; a table with the following entries in it: *n*, *min*, *max*, *mean*, and *stddev*:

```
statsVar = buffer.getstats(bufferVar)
```

Use the following commands to print these entries:

```
print(statsVar.n)
print(statsVar.mean)
print(statsVar.stddev)
print(statsVar.min.reading)
print(statsVar.min.timestamp)
print(statsVar.min.seconds)
print(statsVar.min.fractionalseconds)
print(statsVar.max.reading)
print(statsVar.max.seconds)
print(statsVar.max.fractionalseconds)
print(statsVar.max.timestamp)
```

Example

```
reset()
trigger.model.load("SimpleLoop", 12, 0.001, defbuffer1)
trigger.model.initiate()
waitcomplete()

stats = buffer.getstats(defbuffer1)
print(stats)
```

Reset the instrument.

Set up the SimpleLoop trigger-model template to make 12 readings with a 0.001 s delay. Readings are stored in defbuffer1.

Start the trigger model.

Assign the name stats to the table.

Get statistics for the default reading buffer named defbuffer1.

Example output:

```
[ "min" ]={[ "seconds" ]=1561123956, [ "fractionalseconds" ]=0.010184587,
[ "timestamp" ]=1561123956, [ "reading" ]=8.4974199416e-05},
[ "mean" ]=0.000132948335, [ "stddev" ]=4.4270141937e-05,
[ "max" ]={[ "seconds" ]=1561123955, [ "fractionalseconds" ]=0.833083981,
[ "timestamp" ]=1561123955.8, [ "reading" ]=0.0002192359033}, [ "n" ]=12
```

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

buffer.make()

This function creates a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar = buffer.make(bufferSize)
bufferVar = buffer.make(bufferSize, style)
```

bufferVar	The name of the buffer
bufferSize	The maximum number of readings that can be stored in <i>bufferVar</i> ; minimum is 10; 0 to maximize buffer size (see Details)
style	<p>The type of reading buffer to create:</p> <ul style="list-style-type: none"> ■ Store readings with reduced accuracy (6.5 digits) with no formatting information, 1 μs accurate timestamp: <code>buffer .STYLE_COMPACT</code> ■ Store readings with full accuracy with formatting: <code>buffer .STYLE_STANDARD</code> (default) ■ Store the same information as standard, plus additional information: <code>buffer .STYLE_FULL</code> ■ Store external reading buffer data: <code>buffer .STYLE_WRITABLE</code> ■ Store external reading buffer data with two reading values: <code>buffer .STYLE_WRITABLE_FULL</code>

Details

You cannot assign user-defined reading buffers the name `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

If you create a reading buffer that has the same name as an existing user-defined buffer, the existing buffer is overwritten by the new buffer. Any data in the existing buffer is lost.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you create becomes the active buffer.

If you select 0, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The default fill mode of a user-defined buffer is once. You can change it to continuous later.

Once the buffer style is selected, it cannot be changed.

Once you store the first reading in a compact buffer, you cannot change certain measurement settings, including range, display digits, and units; you must clear the buffer first.

Not all remote commands are compatible with the compact, writable, and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

Writable reading buffers are used to bring external data into the instrument. You cannot assign them to collect data from the instrument.

You can change the buffer capacity for an existing buffer through the front panel or by using the `bufferVar.capacity` command.

Example

```
capTest2 = buffer.make(200, buffer.STYLE_FULL)
```

Creates a 200-element reading buffer that stores readings with full accuracy named capTest2.

Also see

- [bufferVar.capacity](#) (on page 14-23)
- [bufferVar.fillmode](#) (on page 14-31)
- [buffer.write.format\(\)](#) (on page 14-50)
- [buffer.write.reading\(\)](#) (on page 14-52)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)
- [Writable reading buffers](#) (on page 6-31)

buffer.math()

This function allows you to run a mathematical expression on a measurement. The expression is applied when the measurement is placed in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.math(readingBuffer, unit, buffer.EXPR_ADD)
buffer.math(readingBuffer, unit, buffer.EXPR_AVERAGE)
buffer.math(readingBuffer, unit, buffer.EXPR_DIVIDE)
buffer.math(readingBuffer, unit, buffer.EXPR_EXPONENT)
buffer.math(readingBuffer, unit, buffer.EXPR_LOG10)
buffer.math(readingBuffer, unit, buffer.EXPR_MULTIPLY)
buffer.math(readingBuffer, unit, buffer.EXPR_NONE)
buffer.math(readingBuffer, unit, buffer.EXPR_POLY, constant0, constant1, constant2,
           constant3, constant4, constant5)
buffer.math(readingBuffer, unit, buffer.EXPR_POWER, constant0)
buffer.math(readingBuffer, unit, buffer.EXPR_RATE)
buffer.math(readingBuffer, unit, buffer.EXPR_RECIPROCAL)
buffer.math(readingBuffer, unit, buffer.EXPR_SQROOT)
buffer.math(readingBuffer, unit, buffer.EXPR_SUBTRACT)
```

<i>readingBuffer</i>	The name of the reading buffer; the reading buffer selected must be set to the style FULL
<i>unit</i>	<p>The units to be applied to the value generated by the expression:</p> <ul style="list-style-type: none"> ▪ DC current: <code>buffer.UNIT_AMP</code> ▪ AC current: <code>buffer.UNIT_AMP_AC</code> ▪ Celsius: <code>buffer.UNIT_CELSIUS</code> ▪ Custom unit 1 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM1</code> ▪ Custom unit 2 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM2</code> ▪ Custom unit 3 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM3</code> ▪ DAC (voltage): <code>buffer.UNIT_DAC</code> ▪ Decibel-milliwatts: <code>buffer.UNIT_DBM</code> ▪ Decibels: <code>buffer.UNIT_DECIBEL</code> ▪ Digital I/O: <code>buffer.UNIT_DIO</code> ▪ Fahrenheit: <code>buffer.UNIT_FAHRENHEIT</code> ▪ Capacitance: <code>buffer.UNIT_FARAD</code> ▪ Frequency: <code>buffer.UNIT_HERTZ</code> ▪ Kelvin: <code>buffer.UNIT_KELVIN</code> ▪ No unit: <code>buffer.UNIT_NONE</code> ▪ Resistance: <code>buffer.UNIT_OHM</code> ▪ Percent: <code>buffer.UNIT_PERCENT</code> ▪ DC voltage ratio: <code>buffer.UNIT_RATIO</code> ▪ Reciprocal: <code>buffer.UNIT_RECIPROCAL</code> ▪ Period: <code>buffer.UNIT_SECOND</code> ▪ Totalizer: <code>buffer.UNIT_TOT</code> ▪ DC voltage: <code>buffer.UNIT_VOLT</code> ▪ AC voltage: <code>buffer.UNIT_VOLT_AC</code> ▪ Power: <code>buffer.UNIT_WATT</code> ▪ <code>buffer.UNIT_X</code>
<i>constant0</i>	The constant to be used for <code>c0</code> in the expression
<i>constant1</i>	The constant to be used for <code>c1</code> in the expression
<i>constant2</i>	The constant to be used for <code>c2</code> in the expression
<i>constant3</i>	The constant to be used for <code>c3</code> in the expression
<i>constant4</i>	The constant to be used for <code>c4</code> in the expression
<i>constant5</i>	The constant to be used for <code>c5</code> in the expression

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style **FULL**. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading
- a = previous reading
- t = timestamp of the reading
- c = constant

Expression	TSP parameter	Formula
Remove math expression	<code>buffer.EXPR_NONE</code>	Not applicable
Add	<code>buffer.EXPR_ADD</code>	$r + p$
Average	<code>buffer.EXPR_AVERAGE</code>	$\frac{(r+a)}{2}$
Divide	<code>buffer.EXPR_DIVIDE</code>	$\frac{r}{a}$
Exponent	<code>buffer.EXPR_EXPONENT</code>	10^r
Log10	<code>buffer.EXPR_LOG10</code>	$\log_{10} r$
Multiply	<code>buffer.EXPR_MULTIPLY</code>	$r * a$
Polynomial	<code>buffer.EXPR_POLY</code>	$c_0 + c_1 \cdot r + c_2 \cdot r^2 + c_3 \cdot r^3 + c_4 \cdot r^4 + c_5 \cdot r^5$
Power	<code>buffer.EXPR_POWER</code>	r^c
Rate of change	<code>buffer.EXPR_RATE</code>	$\frac{(r-r_{-1})}{(t-t_{-1})}$
Reciprocal	<code>buffer.EXPR_RECIPROCAL</code>	$\frac{1}{r}$
Square Root	<code>buffer.EXPR_SQROOT</code>	\sqrt{r}
Subtract	<code>buffer.EXPR_SUBTRACT</code>	$r - a$

Example

```
reset()
mathExp = buffer.make(200, buffer.STYLE_FULL)
smu.measure.func = smu.FUNC_DC_VOLTAGE

buffer.math(mathExp, buffer.UNIT_NONE, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
    print("Reading: ", smu.measure.read(mathExp))
end

display.changescreen(display.SCREEN_READING_TABLE)

print("Extra value reading 1: ", mathExp.extravalues[1])
print("Extra value reading 2: ", mathExp.extravalues[2])
print("Extra value reading 3: ", mathExp.extravalues[3])
```

Reset the instrument.

Make a buffer named `mathExp` set to hold 200 readings with a buffer style of FULL.

Set the measure function to DC voltage.

Set the buffer math expression to multiply readings against the previous readings.

Make three readings.

Display the reading table on the front panel of the instrument, where you can view the extra readings.

Print the extra values (the calculated values).

Example output:

```
Reading: 6.3863430578e-05
Reading: 6.7818055872e-05
Reading: 1.9871571784e-05
Extra value reading 1: 6.3863430578e-05
Extra value reading 2: 4.3310937031e-09
Extra value reading 3: 1.3476513655e-09
```

Also see

[buffer.unit\(\)](#) (on page 14-22)

buffer.save()

This function saves data from the specified reading buffer to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.save(bufferVar, "fileName")
buffer.save(bufferVar, "fileName", timeFormat)
buffer.save(bufferVar, "fileName", timeFormat, start, end)
```

<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>fileName</i>	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
<i>timeFormat</i>	Defines how date and time information from the buffer is saved in the file on the USB flash drive; the options are: <ul style="list-style-type: none"> ■ Save dates, times, and fractional seconds; buffer.SAVE_FORMAT_TIME ■ Save relative timestamps; buffer.SAVE_RELATIVE_TIME ■ Save seconds and fractional seconds; buffer.SAVE_RAW_TIME ■ Save timestamps; buffer.SAVE_TIMESTAMP_TIME
<i>start</i>	Defines the starting point in the buffer to start saving data
<i>end</i>	Defines the ending point in the buffer to stop saving data

Details

The file name must specify the full path (including /usb1/). If included, the file extension must be set to .csv. If no file extension is specified, .csv is added.

Examples of valid destination file names:

```
buffer.save(MyBuffer, "/usb1/myData")
buffer.save(MyBuffer, "/usb1/myData.csv")
```

The 2450 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing CSV files on the flash drive.

Example 1

```
buffer.save(MyBuffer, "/usb1/myData.csv")
```

Save all reading and default time information from a buffer named MyBuffer to a file named myData.csv on the USB flash drive.

Example 2

```
buffer.save(MyBuffer, "/usb1/myDataRel.csv", buffer.SAVE_RELATIVE_TIME)
```

Save all readings and relative timestamps from MyBuffer to a file named myDataRel.csv on the USB flash drive.

Example 3

```
buffer.save(defbuffer1, "/usb1/defbuf1data", buffer.SAVE_RAW_TIME)
```

Save readings and raw time stamps from defbuffer1 to a file named defbuf1data on the USB flash drive.

Also see

[buffer.make\(\)](#) (on page 14-14)
[buffer.saveappend\(\)](#) (on page 14-20)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

buffer.saveappend()

This function appends data from the reading buffer to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.saveappend(bufferVar, "filename")
buffer.saveappend(bufferVar, "filename", timeFormat)
buffer.saveappend(bufferVar, "filename", timeFormat, start, end)
```

bufferVar	Indicates the reading buffer to use; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
fileName	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
timeFormat	Indicates how date and time information from the buffer is saved in the file on the USB flash drive; the options are: <ul style="list-style-type: none"> ▪ Save dates, times, and fractional seconds: <code>buffer.SAVE_FORMAT_TIME</code> ▪ Save relative timestamps: <code>buffer.SAVE_RELATIVE_TIME</code> ▪ Save seconds and fractional seconds: <code>buffer.SAVE_RAW_TIME</code> ▪ Save timestamps: <code>buffer.SAVE_TIMESTAMP_TIME</code>
start	Defines the starting point in the buffer to start saving data
end	Defines the ending point in the buffer to stop saving data

Details

If the file you specify does not exist on the USB flash drive, this command creates the file.

For options that save more than one item of time information, each item is comma-delimited. For example, the default format is date, time, and fractional seconds for each reading.

The file extension .csv is appended to the file name if necessary. Any file extension other than .csv generates an error.

The index column entry in the .csv file starts at 1 for each append operation.

Examples of valid destination file names:

```
buffer.saveappend(bufferVar, "/usb1/myData")
buffer.saveappend(bufferVar, "/usb1/myData.csv")
```

Invalid destination file name examples:

```
buffer.saveappend(bufferVar, "/usb1/myData.")
```

— The period is not followed by csv.

```
buffer.saveappend(bufferVar, "/usb1/myData.txt")
```

— The only allowed extension is .csv. If .csv is not assigned, it is automatically added.

Example 1

```
buffer.saveappend(MyBuffer, "/usb1/myData.csv")
```

Append reading and default time information from a buffer named MyBuffer to a file named myData.csv on the USB flash drive.

Example 2

```
buffer.saveappend(MyBuffer, "/usb1/myDataRel.csv", buffer.SAVE_RELATIVE_TIME)
```

Append readings and relative timestamps from MyBuffer to a file named myDataRel.csv on the USB flash drive.

Example 3

```
reset()
if file.usbdriveexists() == 1 then
    testDir = "TestData1"
    -- Create a directory on the USB drive for the data.
    file.mkdir(testDir)
    -- Build the full file and path.
    fileName = "/usb1/" .. testDir .. "/myTestData.csv"
    -- Open the file where the data will be stored.
    fileNumber = file.open(fileName, file.MODE_WRITE)
    -- Write the string data to a file.
    file.write(fileNumber, "Tested to Company Standard ABC.101\n")
    -- Write the header separator to a file.
    file.write(fileNumber,
        "=====\\n")
    -- Write the string data to a file.
    file.write(fileNumber, "\\t1. Connect HI/LO to respective DUT terminals.\n")
    file.write(fileNumber, "\\t2. Set power supply to 5 VDC @ 1 A.\n")
    file.write(fileNumber, "\\t3. Wait 30 minutes.\n")
    file.write(fileNumber, "\\t4. Capture 100 readings and analyze data.\n\\n\\n")
    -- Write buffering data to a file.
    file.flush(fileNumber)
    -- Close the data file.
    file.close(fileNumber)
end
-- Fix the range to 10 V.
smu.measure.range = 10.0
-- Set the measurement count to 100.
smu.measure.count = 100
-- Set up reading buffers.
-- Ensure the default measurement buffer size matches the count.
defbuffer1.capacity = smu.measure.count
smu.measure.read()
buffer.saveappend(defbuffer1, fileName)
Write string data to a file with information about a test file.
```

Also see

[buffer.make\(\)](#) (on page 14-14)
[buffer.save\(\)](#) (on page 14-19)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

buffer.unit()

This function allows you to create up to three custom units of measure for use in buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.unit(buffer.UNIT_CUSTOMN, unitOfMeasure)
```

N	The number of the custom unit, 1, 2, or 3
unitOfMeasure	A string that defines the custom unit; up to three characters; defaults are x for custom unit 1, y for unit 2, and z for unit 3

Details

You can use custom units of measures in buffer math and writable buffers.

If you specify more than two characters, the additional characters are ignored. Some characters are converted to other symbols:

- u is displayed as μ .
- dC is displayed as $^{\circ}\text{C}$.
- dF is displayed as $^{\circ}\text{F}$.
- RA is displayed as V/V.

This unit is reset when power is cycled. It is not affected by reset.

Example

```
reset()
mathExp = buffer.make(200, buffer.STYLE_FULL)
smu.measure.func = smu.FUNC_DC_VOLTAGE
buffer.unit(buffer.UNIT_CUSTOM1, "fb")
buffer.math(mathExp, buffer.UNIT_CUSTOM1, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
    print("Reading ...x...:", smu.measure.read(mathExp))
end
display.changescreen(display.SCREEN_READING_TABLE)
for x = 1, 3 do
    print("Extra value reading ...x...:", mathExp.extravalues[x])
end
```

Instrument has terminals set to FRONT.
 Reset the instrument.
 Make a buffer named `mathExp` set to hold 200 readings with a buffer style of FULL.
 Set the measure function to DC voltage.
 Set the customer 1 buffer unit to `fb`.
 Set the buffer math expression to multiply readings against the previous readings.
 Make 3 readings.
 Display the reading table on the front panel of the instrument, where you can view the extra readings.
 Print the extra values (the calculated values).
 Example output:
 Reading 1: 0.00015611271869
 Reading 2: 9.0539004907e-05
 Reading 3: 0.30001141669554
 Extra value reading 1: 0.00015611271869
 Extra value reading 2: 1.4134290203e-08
 Extra value reading 3: 1.0336562635e-08

Also see

[buffer.math\(\)](#) (on page 14-15)
[buffer.write.format\(\)](#) (on page 14-50)

bufferVar.capacity

This attribute sets the number of readings a buffer can store.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferCapacity = bufferVar.capacity
bufferVar.capacity = bufferCapacity
```

<code>bufferCapacity</code>	The maximum number of readings the buffer can store; set to 0 to maximize the buffer size (see Details)
<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer

Details

This command allows you to change or view how many readings a buffer can store. Changing the size of a buffer will cause any existing data in the buffer to be lost.

If you select 0, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The overall capacity of all buffers stored in the instrument can be up to 4,500,000 readings for standard reading buffers and 20,000,000 for compact reading buffers.

For more information about buffer capacity, see [Setting reading buffer capacity](#) (on page 6-10).

Example

<pre> reset() testData = buffer.make(500) capTest = buffer.make(300) bufferCapacity = capTest.capacity print(bufferCapacity) print(testData.capacity) testData.capacity = 600 print(testData.capacity) print(defbuffer1.capacity) </pre>	<p>Create two user-defined reading buffers: <code>testData</code> and <code>capTest</code>.</p> <p>Create a variable called <code>bufferCapacity</code> to hold the capacity of the <code>capTest</code> buffer.</p> <p>Print <code>bufferCapacity</code>.</p> <p>Output: 300</p> <p>Print the capacity of <code>testData</code>.</p> <p>Output: 500</p> <p>Changes the capacity of <code>testData</code> to 600.</p> <p>Print the capacity of <code>testData</code>.</p> <p>Output: 600</p> <p>Print the capacity of the default buffer <code>defbuffer1</code>.</p> <p>Output: 10000</p>
---	--

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.clear()

This function clears all readings and statistics from the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`bufferVar.clear()`

<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
------------------------	---

Example

```

reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData)
testData.clear()
print("Readings in buffer after clear ="
      .. testData.n)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData)

```

Create a reading buffer named `testData`, make three readings and store them in `testData`, and then view the readings.

Print number of readings in `testData`.

Output:

`-4.5010112303956e-10, -3.9923108222095e-12, -4.5013931471161e-10`

Clear the readings in `testData`.

Verify that there are no readings in `testData`.

Output:

`Readings in buffer after clear = 0`

Store three new readings in `testData` and view those when complete.

Output:

`4.923509754e-07, 3.332266330e-07, 3.974883867e-07`

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.dates

This attribute contains the dates of readings that are stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
date = bufferVar.dates[N]
```

<code>date</code>	The date of readings stored in <code>bufferVar</code> element <code>N</code>
<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<code>N</code>	The reading number <code>N</code> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

The dates are formatted as month, day, year.

This is not available if the reading buffer style is set to compact.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 1, testData)
trigger.model.initiate()
waitcomplete()
print(testData.dates[1])
printbuffer(1, testData.n, testData.dates)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Print the first reading date.

Example output:

11/27/2017

Prints the dates for readings 1 through the last reading in the buffer.

Example output:

11/27/2017, 11/27/2017,
11/27/2017

Also see

[buffer.delete\(\)](#) (on page 14-10)
[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.endindex

This attribute indicates the last index in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferVar.endindex = endIndex
```

<code>endIndex</code>	Ending index of the buffer
<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer

Details

Use this attribute to find the ending index in a reading buffer.

Example

```

test1 = buffer.make(100)
smu.measure.count = 6
smu.measure.read(test1)
print(test1.startindex, test1.endindex, test1.capacity)
smu.measure.read(test1)
print(test1.startindex, test1.endindex)

Create a buffer named test1 with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer test1.
Get the start index, end index, and capacity of test1.
Output:
1, 6, 100
Make six more measurements and store them in buffer test1.
Get the start index and end index of test1.
Output:
1, 12

```

Also see

[bufferVar.startindex](#) (on page 14-43)
[buffer.make\(\)](#) (on page 14-14)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.extraformattedvalues

This attribute contains the measurement and the unit of measure of the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraFormat = bufferVar.extraformattedvalues[N]
```

extraFormat	The measurement and unit of measure of the extra values for readings
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number N; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the measurement and the unit of measure of an additional value. The reading buffer style must be set to full to use this option.

Example

```
reset()
mathExp = buffer.make(400, buffer.STYLE_FULL)
smu.measure.func = smu.FUNC_DC_VOLTAGE
buffer.math(mathExp, buffer.UNIT_NONE, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
    print("Reading: ", smu.measure.read(mathExp))
end
display.changescreen(display.SCREEN_READING_TABLE)
print("Extra value reading 1: ", mathExp.extraformattedvalues[1])
print("Extra value reading 2: ", mathExp.extraformattedvalues[2])
print("Extra value reading 3: ", mathExp.extraformattedvalues[3])
```

Reset the instrument.

Make a buffer named `mathExp` set to hold 400 readings with a buffer style of FULL.

Set the measure function to DC voltage.

Set the buffer math expression to multiply readings against the previous readings.

Make three readings.

Display the reading table on the front panel of the instrument, where you can view the extra readings.

Print the extra values (the calculated values).

Example output:

```
Reading:      7.1233589551e-06
Reading:      7.1233080234e-06
Reading:      7.2616603575e-06
Extra value reading 1:      +7.1233590 u
Extra value reading 2:      +50.741880 p
Extra value reading 3:      +51.727043 p
```

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.extravalues

This attribute contains the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraValue = bufferVar.extravalues[N]
```

extraValue	The extra values for readings
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number N; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains an additional value, such as the extra value written to a writable buffer. The reading buffer style must be set to full to use this option.

Example

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
    buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
    extBuffer.units)

Creates a 100-point reading buffer named extBuffer. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC,
10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.extraValueUnits

This attribute contains the units of the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraUnits = bufferVar.extraValueUnits[N]
```

extraUnits	The units of the extra values for readings
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number N; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the unit of measure of an additional value. The reading buffer style must be set to full to use this option.

Example

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
    buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.extraValueUnits)
```

Creates a 100-point reading buffer named extBuffer. Style is full writable.

Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.

Write 12 pieces of data into the buffer.

Print the buffer, including the readings and extra value units.

Read the buffer.

Output:

```
1, Watt DC, 2, Watt DC, 3, Watt DC, 4, Watt DC, 5, Watt DC, 6, Watt DC
```

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.fillmode

This attribute determines if a reading buffer is filled continuously or is filled once and stops.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	User-defined buffer: buffer.FILL_ONCE (0) defbuffer1: buffer.FILL_CONTINUOUS (1) defbuffer2: buffer.FILL_CONTINUOUS (1)

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

fillMode	Fill the buffer, then stop: buffer.FILL_ONCE or 0 Fill the buffer continuously: buffer.FILL_CONTINUOUS or 1
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer

Details

When a reading buffer is set to fill once, no data is overwritten in the buffer. When the buffer is filled, no more data is stored in that buffer and new readings are discarded.

When a reading buffer is set to fill continuously, the oldest data is overwritten by the newest data after the buffer fills.

When you change the fill mode of a buffer, any data in the buffer is cleared.

Example

```
reset()
testData = buffer.make(50)
print(testData.fillmode)
testData.fillmode = buffer.FILL_CONTINUOUS
print(testData.fillmode)

Create a reading buffer named testData, configure the instrument to make three measurements, and store the readings in the buffer. Print the fill mode setting for the testData buffer.
Output:
0
Set fill mode to continuous.
Print the fill mode setting for the testData buffer.
Output:
1
```

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.formattedreadings

This attribute contains the stored readings shown as numbers with units and prefixes.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
reading = bufferVar.formattedreadings[N]
```

reading	Buffer reading formatted with numbers and prefixes with units for element <i>N</i>
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This read-only attribute is an array that contains the stored readings. The readings are shown as numbers with prefixes before the units symbol.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.formattedreadings[1])
printbuffer(1, testData.n, testData.formattedreadings)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Print the first reading.
Example output:
+00.0028 nA
Print all readings in the reading buffer.
Example output:
+00.0028 nA, +00.0039 nA, +00.0040 nA

Also see

[bufferVar.readings](#) (on page 14-36)
[buffer.delete\(\)](#) (on page 14-10)
[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.fractionalseconds

This attribute contains the fractional second portion of the timestamp of each reading in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
fractionalSec = bufferVar.fractionalseconds[N]
```

<i>fractionalSec</i>	The fractional second portion of the timestamp to 1 ns resolution
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This read-only attribute is an array of the fractional portion of the timestamp, in seconds, when each reading occurred. Seconds are shown as fractions.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.fractionalseconds[1])
printbuffer(1, 6, testData.fractionalseconds)
```

Create a reading buffer named `testData` and make six measurements.
Print the fractional portion of the timestamp for the first reading in the buffer.
Example output:
0.647118937
Print the fractional portion of the timestamp for the first six readings in the buffer.
Example output:
0.647118937, 0.064543, 0.48196127, 0.89938724, 0.316800064, 0.734218263

Also see

- [bufferVar.seconds](#) (on page 14-38)
- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.logstate

This attribute indicates if information events are logged when the specified reading buffer is at 0% or 100% filled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Power cycle	Not applicable	defbuffer1: buffer.ON (1) defbuffer2: buffer.ON (1) User-created buffer: buffer.OFF (0)

Usage

```
logState = bufferVar.logstate
bufferVar.logstate = logState
```

logState	Do not log information events: buffer.OFF or 0 Log information events: buffer.ON or 1
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer

Details

If this is set to on, when the reading buffer is cleared (0% filled) or full (100% filled), an event is logged in the event log. If this is set to off, reading buffer status is not reported in the event log.

Example

<pre>reset() MyBuffer = buffer.make(500) print(MyBuffer.logstate)</pre>	Create the user-defined buffer MyBuffer. Print the log state of MyBuffer. Output: 0
---	--

Also see

[Using the event log](#) (on page 3-52)

bufferVar.n

This attribute contains the number of readings in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
numberOfReadings = bufferVar.n
```

numberOfReadings	The number of readings stored in the reading buffer
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer

Details

You can call this command to return the number of readings stored in the specified reading buffer.

You can use the *bufferVar.n* attribute in other commands. For example, to print all the readings in a buffer, use the following command:

```
printbuffer(1, bufferVar.n, bufferVar.readings)
```

Where *bufferVar* is the name of the buffer to use.

Example

```
reset()
testData = buffer.make(100)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.n)
print(defbuffer1.n)
print(defbuffer2.n)
```

Create a reading buffer named *testData*, configure the instrument to make three measurements, and store the readings in the buffer.
 Print the number of readings in *testData*.
 Output:
 3
 Print the number of readings in *defbuffer1*.
 Example output:
 0
 Print the number of readings in *defbuffer2*.
 Example output:
 0

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.readings

This attribute contains the readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
reading = bufferVar.readings[N]
```

<i>reading</i>	The value of the reading in the specified reading buffer
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
for x = 1, 3 do printbuffer(x, x, testData.readings, testData.sourcevalues,
    testData.relativetimes) end
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Print the three readings in `testData`, including the measurement, source value, and relative timestamp.
Output:

```
-9.6420389034124e-12, 2, 0
-4.5509945811872e-10, 2, 0.277194856
-9.1078204006445e-12, 2, 0.569614783
```

Also see

[bufferVar.n](#) (on page 14-35)
[buffer.delete\(\)](#) (on page 14-10)
[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.relativetimestamps

This attribute contains the timestamps, in seconds, when each reading occurred, relative to the timestamp of the first entry in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
timestamp = bufferVar.relativetimestamps[N]
```

timestamp	The timestamp, in seconds
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number N; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This read-only attribute is an array of timestamps, in seconds, of when each reading occurred relative to the timestamp of the first entry in the reading buffer. These timestamps are equal to the time that has lapsed for each reading since the first reading was stored in the buffer. Therefore, the relative timestamp for the first entry number in the reading buffer equals 0.

Example

<pre>reset() testData = buffer.make(50) trigger.model.load("SimpleLoop", 3, 0, testData) trigger.model.initiate() waitcomplete() print(testData.relativetimestamps[1]) printbuffer(1, 3, testData.relativetimestamps)</pre>	<p>Create a reading buffer named <code>testData</code>, configure the instrument to make three measurements, and store the readings in the buffer.</p> <p>Print the relative timestamp for the first reading in the buffer.</p> <p>Example output: 0</p> <p>Print the relative timestamp for the reading 1 through 3 in the buffer.</p> <p>Example output: 0, 0.383541, 0.772005</p>
---	--

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.seconds

This attribute contains the timestamp of a reading in seconds, in UTC format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
nonFracSeconds = bufferVar.seconds[N]
```

nonFracSeconds	The nonfractional seconds portion of the timestamp when the reading was stored
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number N; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the nonfractional seconds portion of the timestamp when the reading was stored in Coordinated Universal Time (UTC) format.

The nonfractional seconds portion of the timestamp gives the lowest resolution down to 1 second. To access additional resolution of a timestamp, see *bufferVar.fractionalseconds*.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 6, testData.seconds)
```

Create a reading buffer named `testData`, configure the instrument to make six measurements, and store the readings in the buffer.

Print the seconds portion for readings 1 to 6 in `testData`.

Example output:

```
1362261492, 1362261492,
1362261493, 1362261493,
1362261493, 1362261494
```

Also see

[bufferVar.fractionalseconds](#) (on page 14-33)
[bufferVar.relativetimesamps](#) (on page 14-37)

bufferVar.sourceformattedvalues

This attribute contains the source levels formatted as they appear on the front-panel display when the readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
values = bufferVar.sourceformattedvalues[N]
```

values	The output value of the source when reading <i>N</i> of the specified buffer was acquired
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

The attribute is an array (a Lua table) of the sourced value that was in effect at the time of the reading. The source levels are formatted the same way the readings are formatted when they appear on the front-panel display.

Example

```
reset()
trigger.model.load("SimpleLoop", 6, 0)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.sourceformattedvalues[1])
printbuffer(1,6,defbuffer1.sourceformattedvalues)
```

Example output:
-00.00041 mV
Example output:
-00.00041 mV, +00.00010 mV,
-00.00033 mV, +00.00003 mV,
-00.00028 mV, -00.00045 mV

Also see

- [bufferVar.n](#) (on page 14-35)
- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.sourcestatuses

This attribute contains the source status conditions of the instrument for the reading point.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
statusInfo = bufferVar.sourcestatuses[N]
```

statusInfo	The status value when reading <i>N</i> of the specified buffer was acquired; Source status values are: <ul style="list-style-type: none">▪ Overvoltage protection was active: <code>buffer.STAT_PROTECTION</code>▪ Measured source value was read: <code>buffer.STAT_READBACK</code>▪ Overtemperature condition was active: <code>buffer.STAT_OVER_TEMP</code>▪ Source function level was limited: <code>buffer.STAT_LIMIT</code>▪ Four-wire sense was used: <code>buffer.STAT_SENSE</code>▪ Output was on: <code>buffer.STAT_OUTPUT</code>
bufferVar	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This buffer recall attribute holds an array (a Lua table) of the status values for all the readings in the buffer. The status values are floating-point numbers that encode the status value for each measurement made. See the following table for values.

Buffer status bits for source function when reading was acquired			
Bit	Name	Decimal value	Description
2	STAT_PROTECTION	4	Overvoltage protection was active
3	STAT_READBACK	8	Measured source value was read
4	STAT_OVER_TEMP	16	Overtemperature condition was active
5	STAT_LIMIT	32	Source function level was limited
6	STAT_SENSE	64	Four-wire sense was used
7	STAT_OUTPUT	128	Output was on

Example

```

reset()
testData = buffer.make(50)
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 2, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 2, testData.sourcestatuses)

```

Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.

Turn on the source output.

Print the source status for the readings in `testData`.

Output:

136, 136

Indicating that the status is `buffer.STAT_READBACK` and `buffer.STAT_OUTPUT`.

Also see

[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[buffer.delete\(\)](#) (on page 14-10)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.sourceunits

This attribute contains the units of measure of the source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingUnits = bufferVar.sourceunits[N]
```

readingUnits	The units of measure of the source: <ul style="list-style-type: none"> ■ Volt DC ■ Amp DC
bufferVar	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
N	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

The attribute is an array (a Lua table) of strings indicating the units of measure at the time of the reading.

Example

<pre> reset() testData = buffer.make(50) smu.source.output = smu.ON testData.fillmode = buffer.FILL_CONTINUOUS trigger.model.load("SimpleLoop", 3, 0, testData) smu.source.func = smu.FUNC_DC_CURRENT trigger.model.initiate() waitcomplete() printbuffer(1, testData.n, testData.sourceunits) trigger.model.load("SimpleLoop", 3, 0, testData) smu.source.func = smu.FUNC_DC_VOLTAGE trigger.model.initiate() waitcomplete() printbuffer(1, testData.n, testData.sourceunits) smu.source.output = smu.OFF </pre>	<p>Create a reading buffer named <code>testData</code>, configure the instrument to make three measurements, and store the readings in the buffer.</p> <p>Set the source output to ON.</p> <p>Set the buffer to fill continuously.</p> <p>Set the source function to current.</p> <p>Take three readings.</p> <p>Print the units for the first three readings in the buffer.</p> <p>Output: Amp DC, Amp DC, Amp DC</p> <p>Set the source function to voltage.</p> <p>Take three readings.</p> <p>Print the units for the readings in the buffer.</p> <p>Output: Volt DC, Volt DC, Volt DC</p>
---	---

Also see

- [bufferVar.sourceunits](#) (on page 14-41)
- [bufferVar.sourcevalues](#) (on page 14-42)
- [bufferVar.statuses](#) (on page 14-44)
- [Reading buffers](#) (on page 6-1)

bufferVar.sourcevalues

This attribute contains the source levels being output when readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

`sourceValue = bufferVar.sourcevalues[N]`

<code>sourceValue</code>	The output value of the source when reading N of the specified buffer was acquired
<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
N	The reading number N ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This attribute is like an array (a Lua table) of the sourced value in effect at the time of the reading.

The values returned by this command depend on the source readback state:

- If readback is off, the value is the programmed value
- If readback is on, the value is the actual measured source value

Example

```

reset()
testData = buffer.make(50)
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.level = 1e-6
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 3, testData.sourcevalues)

```

Create a reading buffer named testData, configure the instrument to make three measurements, and store the readings in the buffer.
Set the source value to 1e-6 A.
Print the source values being output when readings in the reading buffer were acquired.
Example output:
9.9999874692e-07,
1.0000017028e-06,
1.0000054544e-06

Also see

[bufferVar.sourcestatuses](#) (on page 14-40)
[bufferVar.sourceunits](#) (on page 14-41)
[bufferVar.statuses](#) (on page 14-44)
[bufferVar.formattedreadings](#) (on page 14-32)
[buffer.delete\(\)](#) (on page 14-10)
[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)
[smu.source.readback](#) (on page 14-190)

bufferVar.startindex

This attribute indicates the starting index in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferVar.startindex = startIndex
```

startIndex	Starting index of the buffer
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer

Details

Use this attribute to find the starting index in a reading buffer.

Example

```
test1 = buffer.make(100)
smu.measure.count = 6
smu.measure.read(test1)
print(test1.startindex, test1.endindex, test1.capacity)
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index, end index, and capacity of `test1`.
Output:
1, 6, 100

Also see

[bufferVar.endindex](#) (on page 14-26)
[buffer.make\(\)](#) (on page 14-14)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.statuses

This attribute contains the status values of readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
statusInformation = bufferVar.statuses[N]
```

<code>statusInformation</code>	The status value when reading N of the specified buffer was acquired; refer to Details
<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
N	The reading number N ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This command is not available if the buffer style is set to compact.

This read-only attribute is an array of status values for the readings in the buffer. The status values are floating-point numbers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description	<i>statusInformation</i> parameter
0x0001	STAT_QUESTIONABLE	1	Measure status questionable	buffer.STAT_QUESTIONABLE
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the 2450, this will always be 0 (main))	buffer.STAT_ORIGIN
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0	buffer.STAT_TERMINAL
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low	buffer.STAT_LIMIT2_LOW
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high	buffer.STAT_LIMIT2_HIGH
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low	buffer.STAT_LIMIT1_LOW
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high	buffer.STAT_LIMIT1_HIGH
0x0100	STAT_START_GROUP	256	First reading in a group	buffer.STAT_START_GROUP

Example

```
reset()
testData = buffer.make(50)
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 2, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 2, testData.statuses)
```

Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.

Turn on the source output.

Print the source status for the readings in `testData`.

Output:

64, 64

Indicating that the status is `buffer.STAT_LIMIT1_LOW`.

Also see

- [buffer.make\(\)](#) (on page 14-14)
- [buffer.delete\(\)](#) (on page 14-10)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [bufferVar.sourcessatuses](#) (on page 14-40)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.times

This attribute contains the time when the instrument made the reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingTime = bufferVar.times[N]
```

readingTime	The time of the reading in hours, minutes, and seconds
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.times[1])
printbuffer(1, 3, testData.times)
```

This example creates a reading buffer named `testData` and makes three measurements.
The `print()` command outputs the time of the first reading.
Output:
`23:09:43`
The `printbuffer()` command outputs the time of readings 1 to 3 in the reading buffer.
Output:
`23:09:43, 23:09:43, 23:09:43`

Also see

[buffer.delete\(\)](#) (on page 14-10)
[buffer.make\(\)](#) (on page 14-14)
[bufferVar.clear\(\)](#) (on page 14-24)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-26)

bufferVar.timestamps

This attribute contains the timestamp when each reading saved in the specified reading buffer occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingTimestamp = bufferVar.timestamps[N]
```

readingTimestamp	The complete timestamp (including date, time, and fractional seconds) of reading number <i>N</i> in the specified reading buffer when the reading was acquired
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the timestamps (date, hours, minutes, seconds, and fractional seconds) of readings stored in the reading buffer.

NOTE

When using the compact buffer style, there is a very small drift between the triggering clock and the timestamp clock, which may result in timestamp truncation and discontinuities over time. The triggering of the measurement remains periodic based on the sample period without any apparent discontinuities.

Example 1

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.timestamps[1])
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Print the first reading date.

Output:

```
03/01/2018 14:46:07.714614838
```

Example 2

```
for x = 1, 3 do printbuffer(x, x, testData.timestamps) end
```

For the buffer created in Example 1, print the timestamps for the readings.

Output:

```
03/01/2018 14:46:07.714614838
03/01/2018 14:46:08.100468838
03/01/2018 14:46:08.487631838
```

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

bufferVar.units

This attribute contains the unit of measure that is stored with readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingUnit = bufferVar.units[N]
```

readingUnit	Volt DC: Voltage measurement Ohm: Resistance measurement Watt DC: Power measurement Amp DC: Current measurement %: Math is set to percent for the measurements X: Math is set to mx+b for the measurements Reciprocal: Math is set to reciprocal for the measurements
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number N; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the unit of measure that is stored with readings in the reading buffer.

Example

```
reset()
testData = buffer.make(50)
testData.fillmode = buffer.FILL_CONTINUOUS
smu.measure.func = smu.FUNC_DC_CURRENT
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.units)
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.units)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Set the buffer to fill continuously.

Set the measure function to current.

Make three readings.

Print the units for the readings.

Output:

Amp DC, Amp DC, Amp DC

Set the measure function to voltage.

Make three readings.

Output:

Volt DC, Volt DC, Volt DC

Also see

- [buffer.delete\(\)](#) (on page 14-10)
- [buffer.make\(\)](#) (on page 14-14)
- [bufferVar.clear\(\)](#) (on page 14-24)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-26)

buffer.write.format()

This function sets the units and number of digits of the readings that are written into the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.write.format(bufferVar, units, displayDigits)
buffer.write.format(bufferVar, units, displayDigits, extraUnits)
buffer.write.format(bufferVar, units, displayDigits, extraUnits, extraDigits)
```

<i>bufferVar</i>	The name of the buffer
<i>units</i>	<p>The units for the first measurement in the buffer index:</p> <ul style="list-style-type: none"> ■ buffer.UNIT_HERTZ ■ buffer.UNIT_KELVIN ■ buffer.UNIT_NONE ■ buffer.UNIT_OHM ■ buffer.UNIT_PERCENT ■ buffer.UNIT_RATIO ■ buffer.UNIT_RECIPROCAL ■ buffer.UNIT_SECOND ■ buffer.UNIT_TOT ■ buffer.UNIT_VOLT ■ buffer.UNIT_VOLT_AC ■ buffer.UNIT_WATT ■ buffer.UNIT_X
<i>displayDigits</i>	<p>The number of digits to use for the first measurement:</p> <ul style="list-style-type: none"> ■ buffer.DIGITS_3_5 ■ buffer.DIGITS_4_5 ■ buffer.DIGITS_5_5 ■ buffer.DIGITS_6_5 ■ buffer.DIGITS_7_5 ■ buffer.DIGITS_8_5
<i>extraUnits</i>	<p>The units for the second measurement in the buffer index; the selections are the same as <i>units</i> (only valid for buffer style WRITABLE_FULL); if not specified, uses the value for <i>units</i></p>
<i>extraDigits</i>	<p>The number of digits to use for the second measurement; the selections are the same as <i>displayDigits</i> (only valid for buffer style WRITABLE_FULL); if not specified, uses the value for <i>displayDigits</i></p>

Details

This command is valid when the buffer style is writable or full writable. When the buffer style is set to full writable, you can include an extra value.

The format defines the units and the number of digits that are reported for the data. This command affects how the data is shown in the reading buffer and what is shown on the front-panel Home, Histogram, Reading Table, and Graph screens.

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1)
buffer.write.reading(extBuffer, 2)
buffer.write.reading(extBuffer, 3)
buffer.write.reading(extBuffer, 4)
buffer.write.reading(extBuffer, 5)
buffer.write.reading(extBuffer, 6)
printbuffer(1, 6, extBuffer.readings, extBuffer.units)

Creates a 100-point reading buffer named extBuffer. Style is writable.
Set the data format to show units of watts with 3½ digit resolution.
Write 6 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:
1.000000000e+00, Watt DC, 2.000000000e+00, Watt DC, 3.000000000e+00, Watt DC,
4.000000000e+00, Watt DC, 5.000000000e+00, Watt DC, 6.000000000e+00, Watt DC
```

Example 2

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
                   buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
           extBuffer.units)

Creates a 100-point reading buffer named extBuffer. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in
the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC,
10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Also see

[buffer.make\(\)](#) (on page 14-14)
[buffer.write.reading\(\)](#) (on page 14-52)
[Reading buffers](#) (on page 6-1)
[Writable reading buffers](#) (on page 6-31)

buffer.write.reading()

This function allows you to write readings into the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

For buffers that are set to the writable buffer style:

```
buffer.write.reading(bufferVar, readingValue)
buffer.write.reading(bufferVar, readingValue, seconds)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds, status)
```

For buffers that are set to the full writable buffer style:

```
buffer.write.reading(bufferVar, readingValue, extraValue)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds, fractionalSeconds)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds, fractionalSeconds,
status)
```

bufferVar	The name of the buffer
readingValue	The first value that is recorded in the buffer index
extraValue	A second value that is recorded in the buffer index (only valid for buffer style WRITABLE_FULL)
seconds	An integer that represents the seconds
fractionalSeconds	The portion of the time that represents the fractional seconds
status	Additional information about the reading; see Details

Details

This command writes the data you specify into a reading buffer. The reading buffer must be set to the writable or full writable style, which is set when you make the buffer.

Data must be added in chronological order. If the time is not specified for a reading, it is set to one integer second after the last reading. As you write the data, the front-panel home screen updates and displays the reading you entered.

The *status* parameter provides additional information about the reading. The options are shown in the following table.

Parameter	Description
buffer.STAT_LIMIT1_HIGH	Reading is above high limit 1
buffer.STAT_LIMIT1_LOW	Reading is below low limit 1
buffer.STAT_LIMIT2_HIGH	Reading is above high limit 2
buffer.STAT_LIMIT2_LOW	Reading is below low limit 2
buffer.STAT_ORIGIN	A/D converter from which reading originated; for the 2450, this will always be 0 (main)
buffer.STAT_QUESTIONABLE	Measure status questionable
buffer.STAT_REL	Relative offset
buffer.STAT_SCAN	Scan
buffer.STAT_START_GROUP	First reading in a group
buffer.STAT_TERMINAL	Measure terminal, front is 1, rear is 0

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1)
buffer.write.reading(extBuffer, 2)
buffer.write.reading(extBuffer, 3)
buffer.write.reading(extBuffer, 4)
buffer.write.reading(extBuffer, 5)
buffer.write.reading(extBuffer, 6)
printbuffer(1, 6, extBuffer.readings, extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is writable.
Set the data format to show units of watts with 3½ digit resolution.
Write 6 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:
1, Watt DC, 2, Watt DC, 3, Watt DC, 4, Watt DC, 5, Watt DC, 6, Watt DC

Example 2

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
                   buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
            extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC,
10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC

Also see

[buffer.make\(\)](#) (on page 14-14)
[buffer.write.format\(\)](#) (on page 14-50)
[Reading buffers](#) (on page 6-1)
[Writable reading buffers](#) (on page 6-31)

createconfigscript()

This function creates a setup file that captures most of the present settings of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
createconfigscript("scriptName")
```

scriptName	A string that represents the name of the script that will be created
------------	--

Details

This function does not automatically overwrite existing scripts with the same name. If *scriptName* is set to the name of an existing script, an event message is returned. You must delete the existing script before using the same script name. This includes the `autoexec` script, which runs automatically when the instrument power is turned on. You can set *scriptName* to `autoexec`, but you must delete the existing `autoexec` script first using the `script.delete("autoexec")` command.

Once created, the script that contains the settings can be run and edited like any other script.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press **HOME** and **ENTER** to save an image of the settings with the screen capture feature. Refer to [Save screen captures to a USB flash drive](#) (on page 3-46) for additional information.

Example

```
createconfigscript("myConfigurationScript")
reset()
myConfigurationScript()
```

Capture the present settings of the instrument into a script named myConfigurationScript.
Reset the instrument.
Restore the settings stored in myConfigurationScript.

Also see

[Saving setups](#) (on page 3-47)
[script.delete\(\)](#) (on page 14-117)

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = dataqueue.add(value)
result = dataqueue.add(value, timeout)
```

result	The resulting value of true or false based on the success of the function
value	The data item to add; value can be of any type
timeout	The maximum number of seconds to wait for space in the data queue

Details

You cannot use the *timeout* value when accessing the data queue from a remote node (you can only use the *timeout* value while adding data to the local data queue).

The *timeout* value is ignored if the data queue is not full.

The dataqueue.add() function returns false:

- If the timeout expires before space is available in the data queue
- If the data queue is full and a *timeout* value is not specified

If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
dataqueue.add(10)
dataqueue.add(11, 2)
result = dataqueue.add(12, 3)
if result == false then
    print("Failed to add 12 to the dataqueue")
end
print("The dataqueue contains:")
while dataqueue.count > 0 do
    print(dataqueue.next())
end
```

Clear the data queue.
Each line adds one item to the data queue.
Output:
The dataqueue contains:
10
11
12

Also see

- [dataqueue.CAPACITY](#) (on page 14-56)
- [dataqueue.clear\(\)](#) (on page 14-56)
- [dataqueue.count](#) (on page 14-57)
- [dataqueue.next\(\)](#) (on page 14-58)
- [Using the data queue for real-time communication](#) (on page 9-10)

dataqueue.CAPACITY

This constant is the maximum number of entries that you can store in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
count = dataqueue.CAPACITY
```

count	The variable that is assigned the value of dataqueue.CAPACITY
-------	---

Details

This constant always returns the maximum number of entries that can be stored in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
    dataqueue.add(1)
end
print("There are " .. dataqueue.count
      .. " items in the data queue")
```

This example fills the data queue until it is full and prints the number of items in the queue.
Output:
There are 128 items in the data queue

Also see

- [dataqueue.add\(\)](#) (on page 14-55)
- [dataqueue.clear\(\)](#) (on page 14-56)
- [dataqueue.count](#) (on page 14-57)
- [dataqueue.next\(\)](#) (on page 14-58)
- [Using the data queue for real-time communication](#) (on page 9-10)

dataqueue.clear()

This function clears the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dataqueue.clear()
```

Details

This function forces all dataqueue.add() commands that are in progress to time out and deletes all data from the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
    dataqueue.add(1)
end
print("There are " .. dataqueue.count
      .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
      .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

There are 128 items in the data queue
There are 0 items in the data queue

Also see

- [dataqueue.add\(\)](#) (on page 14-55)
- [dataqueue.CAPACITY](#) (on page 14-56)
- [dataqueue.count](#) (on page 14-57)
- [dataqueue.next\(\)](#) (on page 14-58)
- [Using the data queue for real-time communication](#) (on page 9-10)

dataqueue.count

This attribute contains the number of items in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

count = dataqueue.count	
count	The number of items in the data queue

Details

The count is updated as entries are added with `dataqueue.add()` and read from the data queue with `dataqueue.next()`. It is also updated when the data queue is cleared with `dataqueue.clear()`.

A maximum of `dataqueue.CAPACITY` items can be stored at any one time in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
    dataqueue.add(1)
end
print("There are " .. dataqueue.count
      .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
      .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

There are 128 items in the data queue
There are 0 items in the data queue

Also see

[dataqueue.add\(\)](#) (on page 14-55)
[dataqueue.CAPACITY](#) (on page 14-56)
[dataqueue.clear\(\)](#) (on page 14-56)
[dataqueue.next\(\)](#) (on page 14-58)
[Using the data queue for real-time communication](#) (on page 9-10)

dataqueue.next()

This function removes the next entry from the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dataqueue.next()
value = dataqueue.next(timeout)
```

value	The next entry in the data queue
timeout	The number of seconds to wait for data in the queue

Details

If the data queue is empty, the function waits up to the *timeout* value.

If data is not available in the data queue before the *timeout* expires, the return value is nil.

The entries in the data queue are removed in first-in, first-out (FIFO) order.

If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
for i = 1, 10 do
    dataqueue.add(i)
end
print("There are " .. dataqueue.count
      .. " items in the data queue")

while dataqueue.count > 0 do
    x = dataqueue.next()
    print(x)
end
print("There are " .. dataqueue.count
      .. " items in the data queue")
```

Clears the data queue, adds ten entries, then reads the entries from the data queue. Note that your output may differ depending on the setting of *format.asciiprecision*.

Output:

```
There are 10 items in the data queue
1
2
3
4
5
6
7
8
9
10
There are 0 items in the data queue
```

Also see

[dataqueue.add\(\)](#) (on page 14-55)
[dataqueue.CAPACITY](#) (on page 14-56)
[dataqueue.clear\(\)](#) (on page 14-56)
[dataqueue.count](#) (on page 14-57)
[format.asciiprecision](#) (on page 14-90)
[Using the data queue for real-time communication](#) (on page 9-10)

delay()

This function delays the execution of the commands that follow it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
delay(seconds)
seconds          The number of seconds to delay (0 to 100 ks)
```

Details

The instrument delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the instrument to delay 5 µs to 10 µs (typical) more than the requested delay.

Example 1

beeper.beep(0.5, 2400) delay(0.250) beeper.beep(0.5, 2400)	Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.
--	--

Example 2

dataqueue.clear() dataqueue.add(35) timer.cleartime() delay(0.5) dt = timer.gettime() print("Delay time was " .. dt) print(dataqueue.next())	Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it. Output: Delay time was 0.500099 35
--	---

Also see

None

digio.line[N].mode

This attribute sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	digio.MODE_DIGITAL_IN

Usage

```
lineMode = digio.line[N].mode
digio.line[N].mode = lineMode
```

<i>lineMode</i>	The digital line control type and line mode: Digital control, input: digio.MODE_DIGITAL_IN Digital control, output: digio.MODE_DIGITAL_OUT Digital control, open-drain: digio.MODE_DIGITAL_OPEN_DRAIN Trigger control, input: digio.MODE_TRIGGER_IN Trigger control, output: digio.MODE_TRIGGER_OUT Trigger control, open-drain: digio.MODE_TRIGGER_OPEN_DRAIN Synchronous master: digio.MODE_SYNCHRONOUS_MASTER Synchronous acceptor: digio.MODE_SYNCHRONOUS_ACCEPTOR
<i>N</i>	The digital I/O line: 1 to 6

Details

You can use this command to place each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

The following settings of *lineMode* set the line for direct control as a digital line:

- `digio.MODE_DIGITAL_IN`: The instrument automatically detects externally generated logic levels. You can read an input line, but you cannot write to it.
- `digio.MODE_DIGITAL_OUT`: You can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low.
- `digio.MODE_DIGITAL_OPEN_DRAIN`: Configures the line to be an open-drain signal. The line can serve as an input, an output or both. When a digital I/O line is used as an input in open-drain mode, you must write a 1 to it.

The following settings of `lineMode` set the line as a trigger line:

- `digio.MODE_TRIGGER_IN`: The line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This line state uses the edge setting specified by the `trigger.digin[N].edge` attribute.
- `digio.MODE_TRIGGER_OUT`: The line is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
- `digio.MODE_TRIGGER_OPEN_DRAIN`: Configures the line to be an open-drain signal. You can use the line to detect input triggers or generate output triggers. This line state uses the edge setting specified by the `trigger.digin[N].edge` attribute.

When the line is set as a synchronous acceptor, the line detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting an output trigger releases the latched line.

When the line is set as a synchronous master, the line detects rising-edge triggers as input. For output, the line asserts a TTL-low pulse.

Example

```
digio.line[1].mode = digio.MODE_TRIGGER_OUT Set digital I/O line 1 to be an output trigger line.
```

Also see

- [Digital I/O lines \(on page 8-16\)](#)
- [Digital I/O port configuration \(on page 8-14\)](#)
- [trigger.digin\[N\].edge \(on page 14-224\)](#)

digio.line[N].reset()

This function resets digital I/O line values to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.line[N].reset()
```

N	The digital I/O line: 1 to 6
---	------------------------------

Details

This function resets the following attributes to their default values:

- `digio.line[N].mode`
- `trigger.digin[N].edge`
- `trigger.digout[N].logic`
- `trigger.digout[N].pulsewidth`
- `trigger.digout[N].stimulus`

It also clears `trigger.digin[N].overrun`.

Example

```
-- Set the digital I/O trigger line 3 for a falling edge
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.digout[3].logic = trigger.LOGIC_NEGATIVE
-- Set the digital I/O trigger line 3 to have a pulsewidth of 50 microseconds.
trigger.digout[3].pulsewidth = 50e-6
-- Use digital I/O line 5 to trigger the event on line 3.
trigger.digout[3].stimulus = trigger.EVENT_DIGIO5
-- Print configuration (before reset).
print(digio.line[3].mode, trigger.digout[3].pulsewidth, trigger.digout[3].stimulus)
-- Reset the line back to factory default values.
digio.line[3].reset()
-- Print configuration (after reset).
print(digio.line[3].mode, trigger.digout[3].pulsewidth, trigger.digout[3].stimulus)

Output before reset:
digio.MODE_TRIGGER_OUT    5e-05    trigger.EVENT_DIGIO5

Output after reset:
digio.MODE_TRIGGER_IN    1e-05    trigger.EVENT_NONE
```

Also see

- [digio.line\[N\].mode](#) (on page 14-60)
- [Digital I/O port configuration](#) (on page 8-14)
- [trigger.digin\[N\].overrun](#) (on page 14-225)
- [trigger.digout\[N\].pulsewidth](#) (on page 14-228)
- [trigger.digout\[N\].stimulus](#) (on page 14-229)

digio.line[N].state

This function sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	See Details

Usage

<code>digio.line[N].state = state</code>	
<code>state = digio.line[N].state</code>	
<code>N</code>	The digital I/O line: 1 to 6
<code>state</code>	Set the line low: <code>digio.STATE_LOW</code> or 0 Set the line high: <code>digio.STATE_HIGH</code> or 1

Details

When a reset occurs, the digital line state can be read as high because the digital line is reset to a digital input. A digital input floats high if nothing is connected to the digital line.

This returns the integer equivalent values of the binary states on all six digital I/O lines.

Set the state to `digio.STATE_LOW` to clear the bit; set the state to `digio.STATE_HIGH` to set the bit.

Example

<code>digio.line[1].mode = digio.MODE_DIGITAL_OUT</code>	Sets line 1 (bit B1) of the digital I/O port high.
<code>digio.line[1].state = digio.STATE_HIGH</code>	

Also see

- [digio.line\[N\].mode](#) (on page 14-60)
- [digio.readport\(\)](#) (on page 14-63)
- [digio.writeport\(\)](#) (on page 14-64)
- [Digital I/O port configuration](#) (on page 8-14)
- [trigger.digin\[N\].edge](#) (on page 14-224)

digio.readport()

This function reads the digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

<code>data = digio.readport()</code>	
<code>data</code>	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the digital I/O port. The least significant bit (bit B1) of the binary number corresponds to digital I/O line 1; bit B6 corresponds to digital I/O line 6.

For example, a returned value of 42 has a binary equivalent of 101010, which indicates that lines 2, 4, 6 are high (1), and the other lines are low (0).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

```
data = digio.readport()  
print(data)
```

Assume lines 2, 4, and 6 are set high when the I/O port is read.
Output:
42
This is binary 101010

Also see

[digio.writeport\(\)](#) (on page 14-64)

[Digital I/O port configuration](#) (on page 8-14)

digio.writeport()

This function writes to all digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writeport(data)
```

data	The value to write to the port (0 to 63)
------	--

Details

This function writes to the digital I/O port by setting the binary state of each digital line from an integer equivalent value.

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a *data* value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other three lines are set low).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

You can also enter the *data* parameter as a binary value.

Example 1

```
digio.writeport(63)
```

Sets digital I/O lines 1 through 6 high (binary 111111).

Example 2

```
digio.writeport(0b111111)
```

Sets digital I/O lines 1 through 6 high (digital 63).

Also see

- [digio.readport\(\) \(on page 14-63\)](#)
- [Digital I/O port configuration \(on page 8-14\)](#)
- [Outputting a bit pattern \(on page 8-23\)](#)

display.activebuffer

This attribute determines which buffer is used for measurements that are displayed on the front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	defbuffer1

Usage

```
bufferName = display.activebuffer
display.activebuffer = bufferName
```

bufferName	The name of the buffer to make active
------------	---------------------------------------

Details

The buffer defined by this command is used to store measurements data and is shown in the reading buffer indicator on the home screen of the instrument.

Example

```
display.activebuffer = buffer2
```

Set the front panel to use buffer2 as the active reading buffer.
--

Also see

None

display.changescreen()

This function changes which front-panel screen is displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.changescreen(screenName)
```

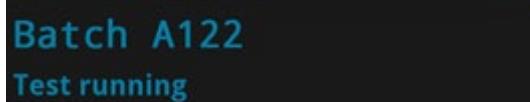
<i>screenName</i>	The screen to display: <ul style="list-style-type: none"> ■ Home screen: <code>display.SCREEN_HOME</code> ■ Home screen with large readings: <code>display.SCREEN_HOME_LARGE_READING</code> ■ Reading table screen: <code>display.SCREEN_READING_TABLE</code> ■ Graph screen (opens last selected tab): <code>display.SCREEN_GRAPH</code> ■ Histogram: <code>display.SCREEN_HISTOGRAM</code> ■ GRAPH swipe screen: <code>display.SCREEN_GRAPH_SWIPE</code> ■ SETTINGS swipe screen: <code>display.SCREEN_SETTINGS_SWIPE</code> ■ SOURCE swipe screen: <code>display.SCREEN_SOURCE_SWIPE</code> ■ STATISTICS swipe screen: <code>display.SCREEN_STATS_SWIPE</code> ■ USER swipe screen: <code>display.SCREEN_USER_SWIPE</code> ■ Open a screen that uses minimal CPU resources: <code>display.SCREEN_PROCESSING</code>
-------------------	---

Example

```
display.clear()
display.settext(display.TEXT1, "Batch A122")
display.settext(display.TEXT2, "Test running")
display.changescreen(display.SCREEN_USER_SWIPE)
```

Clear the USER swipe screen.

Set the first line of the USER swipe screen to read "Batch A122" and the second line to display "Test running".
Display the USER swipe screen.



Also see

[display.settext\(\)](#) (on page 14-77)

display.clear()

This function clears the text from the front-panel USER swipe screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.clear()
```

Example

```
display.clear()
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, "Serial number:")
display.settext(display.TEXT2, localnode.serialno)
```

Clear the USER swipe screen.
Set the first line to read "Serial number:" and the second line to display the serial number of the instrument.

Also see

[display.settext\(\)](#) (on page 14-77)

display.delete()

This function allows you to remove a prompt on the front-panel display that was created with `display.prompt()`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.delete(promptID)
```

<i>promptID</i>	The identifier defined by <code>display.prompt()</code>
-----------------	---

Details

You can use this command to remove the presently displayed prompt.

Example

```
removePrompt3 = display.prompt(display.BUTTONS_NONE, "This prompt will disappear in
3 seconds")
delay(3)
display.delete(removePrompt3)
```

This example displays a prompt that is automatically removed in three seconds.



Also see

[display.prompt\(\)](#) (on page 14-75)

display.input.number()

This function allows you to create a prompt that requests a number from the user on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
numberEntered = display.input.number("dialogTitle")
numberEntered = display.input.number("dialogTitle", numberFormat)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue,
minimumValue)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue,
minimumValue, maximumValue)
```

numberEntered	The number that is entered from the front-panel display; nil if Cancel is pressed on the keypad
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; can be up to 32 characters
numberFormat	The format of the displayed number: <ul style="list-style-type: none"> ■ Allow integers (negative or positive) only: <code>display.NFORMAT_INTEGER</code> (default) ■ Allow decimal values: <code>display.NFORMAT_DECIMAL</code> ■ Display numbers in exponent format: <code>display.NFORMAT_EXPONENT</code> ■ Display numbers with prefixes before the units symbol, such as n, m, or µ: <code>display.NFORMAT_PREFIX</code>
defaultValue	The value that is initially displayed in the displayed keypad
minimumValue	The lowest value that can be entered
maximumValue	The highest value that can be entered

Details

This command prompts the instrument operator to enter a value.

The prompt is displayed until it has been responded to.

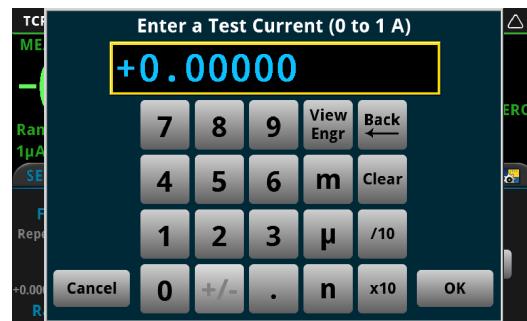
NOTE

On the prompt, the operator can move the cursor in the entry box by touching the screen. The cursor is moved to the spot where the operator touched the screen.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT  
testcurrent = display.input.number("Enter a Test Current (0 to 1 A)",  
    display.NFORMAT_PREFIX, 0, 0, 1)  
smu.source.level = testcurrent
```

This example displays a number pad on the screen that defaults to 0 and allows entries from 0 to 1. The number that the operator enters is assigned to the source current level. If the operator enters a value outside of the range, an error message is displayed.



Also see

- [display.input.option\(\)](#) (on page 14-70)
- [display.input.prompt\(\)](#) (on page 14-71)
- [display.input.string\(\)](#) (on page 14-72)

display.input.option()

This function allows you to create an option dialog box with customizable buttons on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.BUTTON_OPTIONn = display.input.option("dialogTitle", "buttonTitle1",
                                             "buttonTitle2")
display.BUTTON_OPTIONn = display.input.option("dialogTitle", "buttonTitle1",
                                             "buttonTitle2", "buttonTitleN", ... "buttonTitleN")
```

<i>n</i>	The number of the button that is selected from the front-panel display; nil if Cancel is pressed on the keypad; buttons are numbered top to bottom, left to right
<i>dialogTitle</i>	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 32 characters
<i>buttonTitle1</i>	A string that contains the name of the first button; up to 15 characters
<i>buttonTitle2</i>	A string that contains the name of the second button; up to 15 characters
<i>buttonTitleN</i>	A string that contains the names of subsequent buttons, where <i>N</i> is a number from 3 to 10; you can define up to 10 buttons; each button can be up to 15 characters

Details

Buttons are created from top to bottom, left to right. If you have more than five buttons, they are placed into two columns.

The prompt is displayed until it has been responded to. You can only send one input prompt command at a time.

Example

```
optionID = display.input.option("Select an option", "Apple", "Orange", "Papaya",
                               "Pineapple", "Blueberry", "Banana", "Grapes", "Peach", "Apricot", "Guava")
print(optionID)
```

This example displays the following dialog box:



If the user selects Peach, the return is display.BUTTON_OPTION8.

Also see

[display.input.number\(\)](#) (on page 14-68)
[display.input.prompt\(\)](#) (on page 14-71)
[display.input.string\(\)](#) (on page 14-72)

display.input.prompt()

This function allows you to create a prompt that accepts a user response from the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buttonReturn = display.input.prompt(buttonSet, "dialogTitle")
```

buttonReturn	Indicates which button was pressed: <ul style="list-style-type: none"> ■ OK: display.BUTTON_OK ■ Cancel: display.BUTTON_CANCEL ■ Yes: display.BUTTON_YES ■ No: display.BUTTON_NO
buttonSet	The set of buttons to display: <ul style="list-style-type: none"> ■ OK button only: display.BUTTONS_OK ■ Cancel button only: display.BUTTONS_CANCEL ■ OK and Cancel buttons: display.BUTTONS_OKCANCEL ■ Yes and No buttons: display.BUTTONS_YESNO ■ Yes, No, and Cancel buttons: display.BUTTONS_YESNOCANCEL
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 63 characters

Details

This command waits for a user response to the prompt. You can use the text to ask questions that can be used to configure your test.

The prompt is displayed until it has been responded to by the user. You can only send one input prompt command at a time.

Example

```

result = display.input.prompt(display.BUTTONS_YESNO, "Do you want to display the graph
screen?")
if result == display.BUTTON_YES then
    display.changescreen(display.SCREEN_GRAPH)
end

```

This displays the prompt "Do you want to display the graph screen?" on the front-panel display:



If the operator selects Yes, the graph screen is displayed.

Also see

[display.input.number\(\)](#) (on page 14-68)
[display.input.option\(\)](#) (on page 14-70)
[display.input.string\(\)](#) (on page 14-72)

display.input.string()

This function allows you to create a dialog box that requests text from the user through the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```

textEntered = display.input.string("dialogTitle")
textEntered = display.input.string("dialogTitle", textFormat)

```

textEntered	The text that is entered from the front-panel display; nil if Cancel is pressed on the keypad
dialogTitle	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 32 characters
textFormat	<p>The format of the entered text:</p> <ul style="list-style-type: none"> ▪ Allow any characters: <code>display.SFORMAT_ANY</code> (default) ▪ Allow both upper and lower case letters (no special characters): <code>display.SFORMAT_UPPER_LOWER</code> ▪ Allow only upper case letters: <code>display.SFORMAT_UPPER</code> ▪ Allow both upper and lower case letters, no special characters, no spaces, and limited to 32 characters: <code>display.SFORMAT_BUFFER_NAME</code>

Details

This command creates a prompt to the instrument operator to enter a string value.

The prompt is displayed until it has been responded to. You can only send one input prompt command at a time.

Example

```
value = display.input.string("Enter Test Name", display.SFORMAT_ANY)  
print(value)
```

This example displays the prompt "Enter Test Name" and a keyboard that the operator can use to enter a response.



The return is the response from the operator.

Also see

[display.input.number\(\)](#) (on page 14-68)

[display.input.option\(\)](#) (on page 14-70)

[display.input.prompt\(\)](#) (on page 14-71)

display.lightstate

This attribute sets the light output level of the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	display.STATE_LCD_50

Usage

```
brightness = display.lightstate  
display.lightstate = brightness
```

brightness	The brightness of the display: <ul style="list-style-type: none">▪ Full brightness: display.STATE_LCD_100▪ 75% brightness: display.STATE_LCD_75▪ 50% brightness: display.STATE_LCD_50▪ 25% brightness: display.STATE_LCD_25▪ Display off: display.STATE_LCD_OFF▪ Display and all indicators off: display.STATE_BLACKOUT
------------	--

Details

This command changes the light output of the front panel when a test requires different instrument illumination levels.

The change in illumination is temporary. The normal backlight settings are restored after a power cycle. You can use this to reset a display that is already dimmed by the front-panel Backlight Dimmer.

NOTE

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

Example

```
display.lightstate = display.STATE_LCD_50
```

Set the display brightness to 50%.

Also see

[Adjust the backlight brightness and dimmer](#) (on page 3-7)

display.prompt()

This function allows you to create an interactive dialog prompt that displays a custom message on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
promptID = display.prompt(buttonID, "promptText")
```

<i>promptID</i>	A set of characters that identifies the prompt; up to 63 characters
<i>buttonID</i>	The type of prompt to display; choose one of the following options: <ul style="list-style-type: none"> ▪ <code>display.BUTTONS_NONE</code> ▪ <code>display.BUTTONS_OK</code> ▪ <code>display.BUTTONS_CANCEL</code> ▪ <code>display.BUTTONS_OKCANCEL</code> ▪ <code>display.BUTTONS_YESNO</code> ▪ <code>display.BUTTONS_YESNOCANCEL</code>
<i>promptText</i>	A string that contains the text that is displayed above the prompts

Details

This command displays buttons and text on the front panel. You can set up scripts that respond to the buttons when they are selected.

If you send `display.BUTTONS_NONE`, the operator needs to press the EXIT key to clear the message from the front-panel display. You can also use the `display.delete()` command to remove the prompt.

Only one prompt can be active at a time.

When the user presses a button, the button presses are returned as one of the following options:

- **OK:** `display.BUTTON_OK`
- **Cancel:** `display.BUTTON_CANCEL`
- **Yes:** `display.BUTTON_YES`
- **No:** `display.BUTTON_NO`

To capture return values, you need to use `display.waitevent()` to wait for the user button selection.

Example

```

smu.source.sweeplinear("test", 1, 10, 10)
display.prompt(display.BUTTONS_YESNO, "Would you like to start the sweep now? ")
sweepTest, result = display.waitevent()
if result == display.BUTTON_YES then
    trigger.model.initiate()
end
display.prompt(display.BUTTONS_YESNO, "Would you like to switch to the Graph screen? ")
promptID, result = display.waitevent()
if result == display.BUTTON_YES then
    display.changescreen(display.SCREEN_GRAPH)
end
Create a linear sweep.
Display the prompt "Would you like to start the sweep now?"
If the user presses Yes, the sweep starts.
If the user presses No, the sweep does not start and the message is removed.
Display the prompt "Would you like to switch to the Graph screen?"
If the user presses Yes, the Graph screen is displayed.
If the user presses No, the user remains on the present screen.

```

Also see

[display.delete\(\)](#) (on page 14-67)
[display.waitevent\(\)](#) (on page 14-78)

display.readingformat

This attribute determines the format that is used to display measurement readings on the front-panel display of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	display.FORMAT_PREFIX

Usage

```

format = display.readingformat
display.readingformat = format

```

format	Use exponent format: display.FORMAT_EXPONENT Add a prefix to the units symbol, such as k, m, or μ : display.FORMAT_PREFIX
--------	--

Details

This setting persists through `reset()` and power cycles.

When Prefix is selected, prefixes are added to the units symbol, such as k (kilo) or m (milli). When Exponent is selected, exponents are used instead of prefixes. When the prefix option is selected, very large or very small numbers may be displayed with exponents.

Example

display.readingformat = display.FORMAT_EXPONENT	Change front-panel display to show readings in exponential format.
---	--

Also see

[Setting the display format \(on page 3-43\)](#)

display.settext()

This function defines the text that is displayed on the front-panel USER swipe screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.settext(display.TEXT1, "userDisplayText1")
display.settext(display.TEXT2, "userDisplayText2")
```

<i>userDisplayText1</i>	String that contains the message for the top line of the USER swipe screen (up to 20 characters)
<i>userDisplayText2</i>	String that contains the message for the bottom line of the USER swipe screen (up to 32 characters)

Details

This command defines text messages for the USER swipe screen.

If you enter too many characters, the instrument displays a warning event and shortens the message to fit.

You can send use the following codes to create special characters in the message.

Code	Special character
\018	Ω
\019	°
\020	μ
\021	Thin space
\178	Superscript 2 (for example, x ²)
\179	Superscript 3 (for example, x ³)
\185	Δ
\188	1/x
\189	Ratio

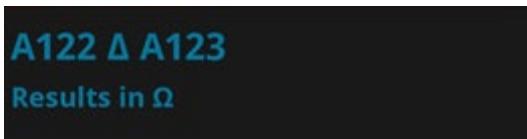
Example

```
display.clear()
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, "A122 \185 A123")
display.settext(display.TEXT2, "Results in \018")
```

Clear the USER swipe screen.

Display the USER swipe screen.

Set the first line to read "A122 Δ A123" and the second line to display ""Results in Ω":



Also see

[display.clear\(\)](#) (on page 14-67)

[display.changescreen\(\)](#) (on page 14-66)

display.waitevent()

This function causes the instrument to wait for a user to respond to a prompt that was created with a prompt command.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
objectID, subID = display.waitevent()
objectID, subID = display.waitevent(timeout)
```

objectID	A number that identifies the object, such as a prompt message, that is displayed on the front panel
subID	The returned value after a button is pressed on the front panel: <ul style="list-style-type: none"> ▪ display.BUTTON_YES ▪ display.BUTTON_NO ▪ display.BUTTON_OK ▪ display.BUTTON_CANCEL
timeout	The amount of time to wait before timing out; time is 0 to 300 s, where the default of 0 waits indefinitely

Details

This command waits until a user responds to a front-panel prompt that was created with the `display.prompt()` command.

Example

```

smu.source.sweeplinear("test", 1, 10, 10)
display.prompt(display.BUTTONS_YESNO, "Would you like to start the sweep now? ")
sweepTest, result = display.waitevent()
if result == display.BUTTON_YES then
    trigger.model.initiate()
end
display.prompt(display.BUTTONS_YESNO, "Would you like to switch to the Graph screen? ")
promptID, result = display.waitevent()
if result == display.BUTTON_YES then
    display.changescreen(display.SCREEN_GRAPH)
end

```

Create a linear sweep.
 Display the prompt "Would you like to start the sweep now?"
 If the user presses Yes, the sweep starts.
 If the user presses No, the sweep does not start and the message is removed.
 Display the prompt "Would you like to switch to the Graph screen?"
 If the user presses Yes, the Graph screen is displayed.
 If the user presses No, the user remains on the present screen.

Also see

[display.input.prompt\(\)](#) (on page 14-71)
[display.prompt\(\)](#) (on page 14-75)

eventlog.clear()

This function clears the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.clear()
```

Details

This command removes all events from the event log, including entries in the front-panel event log.

Also see

[eventlog.next\(\)](#) (on page 14-81)
[eventlog.save\(\)](#) (on page 14-84)
[Using the event log](#) (on page 3-52)

eventlog.getcount()

This function returns the number of unread events in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.getcount()
eventlog.getcount(eventType)
```

eventType	<p>Limits the return to specific event log types; set a cumulative integer value that represents the event log types to:</p> <ul style="list-style-type: none"> ■ Errors only: eventlog.SEV_ERROR or 1 ■ Warnings only: eventlog.SEV_WARN or 2 ■ Errors and warnings only: eventlog.SEV_WARN eventlog.SEV_ERROR or 3 ■ Information only: eventlog.SEV_INFO or 4 ■ Errors and information only: eventlog.SEV_INFO eventlog.SEV_ERROR or 5 ■ Warnings and information only: eventlog.SEV_INFO eventlog.SEV_WARN or 6 ■ All events: eventlog.SEV_ALL or 7
-----------	---

Details

A count finds the number of unread events in the event log. You can specify the event types to return, or return the count for all events.

This command reports the number of events that have occurred since the command was last sent or since the event log was last cleared.

Events are read automatically when `localnode.showevents` is enabled. You can also read them individually with `eventlog.next()`.

Example

```
print(eventlog.getcount(eventlog.SEV_INFO))
```

Displays the present number of unread information messages in the instrument event log.
If there are three information messages in the event log, output is:
3

Also see

[eventlog.clear\(\)](#) (on page 14-79)
[eventlog.next\(\)](#) (on page 14-81)
[localnode.showevents](#) (on page 14-107)
[Using the event log](#) (on page 3-52)

eventlog.next()

This function returns the oldest unread event message from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventNumber, message, severity, nodeID, timeSeconds, timeNanoSeconds =
    eventlog.next()
eventNumber, message, severity, nodeID, timeSeconds, timeNanoSeconds =
    eventlog.next(eventType)
```

eventNumber	The event number
message	A description of the event
severity	The severity of the event: <ul style="list-style-type: none"> ■ Error: 1 ■ Warning: 2 ■ Information: 4
nodeID	The TSP-Link node where the event occurred or 0 if the event occurred on the local node
timeSeconds	The seconds portion of the time when the event occurred
timeNanoSeconds	The fractional seconds portion of the time when the event occurred
eventType	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: <ul style="list-style-type: none"> ■ Errors only: eventlog.SEV_ERROR or 1 ■ Warnings only: eventlog.SEV_WARN or 2 ■ Errors and warnings only: eventlog.SEV_WARN eventlog.SEV_ERROR or 3 ■ Information only: eventlog.SEV_INFO or 4 ■ Errors and information only: eventlog.SEV_INFO eventlog.SEV_ERROR or 5 ■ Warnings and information only: eventlog.SEV_INFO eventlog.SEV_WARN or 6 ■ All events: eventlog.SEV_ALL or 7

Details

When an event occurs on the instrument, it is placed in the event log. The `eventlog.next()` command retrieves an unread event from the event log. Once an event is read, it can no longer be accessed remotely. However, it can be viewed on the front panel. When `localnode.showevents` is enabled, this command never returns an event because those events are automatically read and sent to the remote interface.

To read multiple events, execute this command multiple times.

If there are no entries in the event log, the following is returned:

```
0    No error      0    0    0    0
```

If the event type is not defined, an event of any type is returned.

Example

```
print(eventlog.next(5))
```

Get the oldest error or information event from the event log.

Example output:

```
-285 TSP Syntax error at line 1: unexpected symbol near `0' 1 0 1367806152 652040060
```

Also see

- [eventlog.clear\(\) \(on page 14-79\)](#)
- [eventlog.getcount\(\) \(on page 14-80\)](#)
- [eventlog.save\(\) \(on page 14-84\)](#)
- [Using the event log \(on page 3-52\)](#)

eventlog.post()

This function allows you to post your own text to the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.post( "message" )
eventlog.post( "message", eventType )
```

<code>message</code>	String that contains the message
<code>eventType</code>	The type of event; if no event is defined, defaults to <code>eventlog.SEV_INFO</code> : <ul style="list-style-type: none"> ■ Error: <code>eventlog.SEV_ERROR</code> or 1 ■ Warning: <code>eventlog.SEV_WARN</code> or 2 ■ Information: <code>eventlog.SEV_INFO</code> or 4

Details

You can use this command to create your own event log entries and assign a severity level to them. This can be useful for debugging and status reporting.

From the front panel, you must set the Log Warnings and Log Information options on to have the custom warning and information events placed into the event log.

You can send use the following codes to create special characters in the message.

Code	Special character
\018	Ω
\019	◦
\020	μ
\021	Thin space
\178	Superscript 2 (for example, x^2)
\179	Superscript 3 (for example, x^3)
\185	Δ
\188	1/x
\189	Ratio

These characters are displayed on the front-panel display only.

Example

```
eventlog.clear()
eventlog.post("Results in \018", eventlog.SEVER_ERROR)
print(eventlog.next())
Posts an event that states "Results in Ω".
Output:
1005 User: Results in Ω
```

Also see

[Using the event log](#) (on page 3-52)

eventlog.save()

This function saves the event log to a file on a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.save("filename")
eventlog.save("filename", eventType)
```

<i>filename</i>	A string that represents the name of the file to be saved
<i>eventType</i>	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: <ul style="list-style-type: none"> ■ Errors only: eventlog.SEV_ERROR or 1 ■ Warnings only: eventlog.SEV_WARN or 2 ■ Errors and warnings only: eventlog.SEV_WARN eventlog.SEV_ERROR or 3 ■ Information only: eventlog.SEV_INFO or 4 ■ Errors and information only: eventlog.SEV_INFO eventlog.SEV_ERROR or 5 ■ Warnings and information only: eventlog.SEV_INFO eventlog.SEV_WARN or 6 ■ All events: eventlog.SEV_ALL or 7 (default)

Details

This command saves all event log entries to a USB flash drive.

If you do not define an event type, the instrument saves all event log entries.

The extension .csv is automatically added to the file name.

Example

```
eventlog.save("/usb1/WarningsApril", eventlog.SEV_WARN)
```

Save warning messages to a .csv file on a USB flash drive.

Also see

[eventlog.next\(\) \(on page 14-81\)](#)

exit()

This function stops a script that is presently running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
exit( )
```

Details

Terminates script execution when called from a script that is being executed.

This command does not wait for overlapped commands to complete before terminating script execution. If overlapped commands are required to finish, use the `waitcomplete()` function before calling `exit()`.

Also see

[waitcomplete\(\) \(on page 14-318\)](#)

file.close()

This function closes a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.close(fileNumber)
```

<i>fileNumber</i>	The file number returned from the <code>file.open()</code> function to close
-------------------	---

Details

Note that files are automatically closed when the file descriptors are garbage collected.

Example

<code>file_num = file.open("/usb1/GENTRIGGER" , file.MODE_WRITE)</code>
<code>file.close(file_num)</code>

Open the file GENTRIGGER for writing, then close it.
--

Also see

[file.open\(\) \(on page 14-87\)](#)

file.flush()

This function writes buffering data to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.flush(fileNumber)
```

<i>fileNumber</i>	The file number returned from the <code>file.open()</code> function to flush
-------------------	--

Details

The `file.write()` function may be buffering data instead of writing immediately to the USB flash drive. Use `file.flush()` to flush this data. Data may be lost if the file is not closed or flushed before a script ends.

If there is going to be a time delay before more data is written to a file, flush the file to prevent loss of data because of an aborted test.

Example

```
reset()
-- Fix the range to 10 V
smu.measure.range = 10
-- Set the measurement count to 100
smu.measure.count = 100
-- Set up reading buffers
-- Ensure the default measurement buffer size matches the count
defbuffer1.capacity = 100
smu.measure.read()
testDir = "TestData5"
-- create a directory on the USB drive for the data
file.mkdir(testDir)
fileName = "/usb1/" .. testDir .. "/myTestData.csv"
buffer.save(defbuffer1, fileName)
if file.usbdriveexists() != 0 then
    -- testDir = "TestData3"
    -- Create a directory on the USB drive for the data
    -- file.mkdir(testDir)
    -- Open the file where the data will be stored
    -- fileName = "/usb1/" .. testDir .. "/myTestData.csv"
    -- fileNumber = file.open(fileName, file.MODE_APPEND)
    -- Write header separator to file
    file.write(fileNumber,
    "\n\n=====\n")
    -- Write the string data to a file
    file.write(fileNumber, "Tested to Company Standard ABC.123\n")
    -- Ensure a hurry-up of data written to the file before close or script end
    file.flush(fileNumber)
    -- Close the data file
    file.close(fileNumber)
end
```

This example writes a string that indicates that the readings were made for a certain reason, such as to test to a company standard.

Also see

None

file.mkdir()

This function creates a directory at the specified path on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.mkdir("path")
```

path	A string that contains the path of the directory
------	--

Details

The directory path must be absolute. The name of the directory must not already exist on the flash drive.

Example

file.mkdir("TestData")	Create a new directory named TestData.
------------------------	--

Also see

None

file.open()

This function opens a file on the USB flash drive for later reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileNumber = file.open("fileName", accessType)
```

fileNumber	A number identifying the open file that you use with other file commands to write, read, flush, or close the file after opening
fileName	A string that contains the file name to open, including the full path of file
accessType	The type of action to do: <ul style="list-style-type: none"> ▪ Append the file: file.MODE_APPEND ▪ Read the file: file.MODE_READ ▪ Write to the file: file.MODE_WRITE

Details

The path to the file to open must be absolute.

The root folder of the USB flash drive has the following absolute path:

"/usb1/"

Example

```
file_num = file.open("/usb1/testfile.txt", file.MODE_WRITE)
if file_num != nil then
    file.write(file_num, "This is my test file")
    file.close(file_num)
end
Opens file testfile.txt for writing. If no errors were found while opening, writes This is my test file and closes the file.
```

Also see

None

file.read()

This function reads data from a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileContents = file.read(fileNumber, readAction)
```

fileContents	The contents of the file based on the <i>readAction</i> parameter
fileNumber	The file number returned from the <i>file.open()</i> function to read
readAction	<p>The action:</p> <ul style="list-style-type: none"> ▪ Return the next line; returns <i>nil</i> if the present file position is at the end of the file: <i>file.READ_LINE</i> ▪ Return a string that represents the number found; returns an event string if no number was found; returns <i>nil</i> if the current file position is at the end of file: <i>file.READ_NUMBER</i> ▪ Return the whole file, starting at the present position; returns <i>nil</i> if the present file position is at the end of the file: <i>file.READ_ALL</i>

Details

This command reads data from a file.

Example

```
file_num = file.open("/usb1/testfile.txt", file.MODE_READ)
if file_num != nil then
    file_contents = file.read(file_num, file.READ_ALL)
    file.close(file_num)
end
```

Open *testfile.txt* on the USB flash drive for reading. If it opens successfully, read the entire contents of the file and store it in variable *file_contents*.

Close the file.

Also see

None

file.usbdriveexists()

This function detects if a USB flash drive is inserted into the front-panel USB port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
driveInserted = file.usbdriveexists()

driveInserted      0: No flash drive is detected
                    1: Flash drive is detected
```

Details

You can call this command from a script to verify that a USB flash drive is inserted before attempting to write data to it.

Example

```
local response
local xMax = 10
for x = 1, xMax do
    -- Make xMax readings and store them in defbuffer1.
    smu.measure.read()
end
if (file.usbdriveexists() == 1) then
    response = display.BUTTON_YES
else
    response = display.input.prompt(display.BUTTONS_YESNO, "Insert a USB flash
    drive.\nPress Yes to write data or No to not write data.")
end
if (response == display.BUTTON_YES) then
    if (file.usbdriveexists() == 1) then
        FileNumber = file.open("/usb1/TenReadings.csv", file.MODE_WRITE)
        file.write(FileNumber, "Reading,Seconds\n")
        -- Print out the measured values in a two-column format.
        print("\nIteration:\tReading:\tTime:\n")
        for i = 1, xMax do
            print(i, defbuffer1[i], defbuffer1.relativetimestamps[i])
            file.write(FileNumber, string.format("%g, %g\r\n", defbuffer1.readings[i],
                defbuffer1.relativetimestamps[i]))
        end
        file.close(FileNumber)
    else
        response = display.input.prompt(display.BUTTONS_OK,
            "No drive detected. Allow more time after inserting a drive.")
    end
end
Make measurements.
Verify that a flash drive is inserted into the instrument.
If the flash drive is inserted, write the data to the flash drive.
Print data into a two column format.
If the flash drive is not inserted after selecting Yes, another prompt is displayed.
```

Also see

None

file.write()

This function writes data to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.write(fileNumber, "string")
```

<i>fileNumber</i>	The file number returned from the <code>file.open()</code> function to write
<i>string</i>	A string that contains the data to write to the file

Details

The `file.write()` function may include data that is buffering; it may not be written to the USB flash drive immediately. Use the `file.flush()` function to immediately write buffered data to the drive.

You must use the `file.close()` command to close the file after writing.

Example

```
file_num = file.open("testfile.txt",
                     file.MODE_WRITE)
if file_num != nil then
    file.write(file_num, "This is my test file")
    file.close(file_num)
end
```

Opens file `testfile.txt` for writing. If no errors were found while opening, writes `This is my test file` and closes the file.

Also see

[file.close\(\)](#) (on page 14-85)
[file.flush\(\)](#) (on page 14-86)

format.asciiprecision

This attribute sets the precision (number of digits) for all numbers returned in the ASCII format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	0 (Automatic)

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

<i>precision</i>	A number representing the number of digits to be printed for numbers printed with the <code>print()</code> , <code>printbuffer()</code> , and <code>printnumber()</code> functions; must be a number from 1 to 16; set to 0 to have the instrument select the precision automatically based on the number that is being formatted
------------------	---

Details

This attribute specifies the precision (number of digits) for numeric data printed with the `print()`, `printbuffer()`, and `printnumber()` functions. The `format.asciiprecision` attribute is only used with the ASCII format. The precision value must be a number from 0 to 16.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

<pre>format.asciiprecision = 10 x = 2.54 printnumber(x) format.asciiprecision = 3 printnumber(x)</pre>	Output: 2.54000000e+00 2.54e+00
--	--

Also see

[format.byteorder](#) (on page 14-91)
[format.data](#) (on page 14-92)
[print\(\)](#) (on page 14-111)
[printbuffer\(\)](#) (on page 14-112)
[printnumber\(\)](#) (on page 14-115)

format.byteorder

This attribute sets the binary byte order for the data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	<code>format.LITTLEENDIAN</code>

Usage

```
order = format.byteorder
format.byteorder = order
```

<code>order</code>	Byte order value as follows: <ul style="list-style-type: none"> ▪ Most significant byte first: <code>format.BIGENDIAN</code> ▪ Least significant byte first: <code>format.LITTLEENDIAN</code>
--------------------	---

Details

This attribute selects the byte order in which data is written when you are printing data values with the `printnumber()` and `printbuffer()` functions. The byte order attribute is only used with the `format.REAL32` and `format.REAL64` data formats.

If you are sending data to a computer with a Microsoft Windows operating system, select the `format.LITTLEENDIAN` byte order.

Example

```
x = 1.23
format.data = format.REAL32
format.byteorder = format.LITTLEENDIAN
printnumber(x)
format.byteorder = format.BIGENDIAN
printnumber(x)
```

Output depends on the terminal program you use, but will look something like:
#0¤p??
#0??p¤

Also see

[format.asciiprecision](#) (on page 14-90)
[format.data](#) (on page 14-92)
[printbuffer\(\)](#) (on page 14-112)
[printnumber\(\)](#) (on page 14-115)

format.data

This attribute sets the data format for data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	<code>format.ASCII</code>

Usage

```
value = format.data
format.data = value
```

value	The format to use for data, set to one of the following values: <ul style="list-style-type: none"> ■ ASCII format: <code>format.ASCII</code> ■ Single-precision IEEE Std 754 binary format: <code>format.REAL32</code> ■ Double-precision IEEE Std 754 binary format: <code>format.REAL64</code>
-------	---

Details

You can control the precision of numeric values with the `format.asciiprecision` attribute. If `format.REAL32` or `format.REAL64` is selected, you can select the byte order with the `format.byteorder` attribute.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with #0 and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

Example

<pre>format.asciiprecision = 10 x = 3.14159265 format.data = format.ASCII printnumber(x) format.data = format.REAL64 printnumber(x)</pre>	<p>Output a number represented by <i>x</i> in ASCII using a precision of 10, then output the same number in binary using double precision format.</p> <p>Output:</p> <p>3.141592650e+00 #0ñÔÈSû! @</p>
---	--

Also see

[format.asciiprecision](#) (on page 14-90)
[format.byteorder](#) (on page 14-91)
[printbuffer\(\)](#) (on page 14-112)
[printnumber\(\)](#) (on page 14-115)

fs.chdir()

This function sets the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

<i>workingDirectory</i> = fs.chdir("path")	
<i>workingDirectory</i>	Returned value containing the working path
<i>path</i>	A string indicating the new working directory path

Details

The new working directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist.

Example

<pre>if fs.is_dir("/usb1/temp") == true then fs.chdir("/usb1/temp") testPath = fs.getcwd() print(testPath) else testPath = fs.getcwd() print(testPath) end</pre>	<p>Insert a USB flash drive into the front panel of the instrument. Verify that /usb1/temp is a directory and change it to be the current working directory. Set the variable for the current working directory to be <i>testPath</i>. The return should be: /usb1/temp If /usb1/temp is not a directory, set the variable for the current working directory to be <i>testPath</i>. The return should be: /usb1</p>
--	--

Also see

None

fs.cwd()

This function returns the absolute path of the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.cwd()

path          The absolute path of the current working directory
```

Example

```
if fs.is_dir("/usb1/temp") == true then
    fs.chdir("/usb1/temp")
    testPath = fs.cwd()
    print(testPath)
else
    testPath = fs.cwd()
    print(testPath)
end

Insert a USB flash drive into the front panel of the instrument.
Verify that /usb1/temp is a directory and change it to be the current working directory.
Set the variable for the current working directory to be testPath.
The return should be:
/usb1/temp
If /usb1/temp is not a directory, set the variable for the current working directory to be testPath.
The return should be:
/usb1
```

Also see

None

fs.is_dir()

This function tests whether or not the specified path refers to a directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_dir("path")

status          Whether or not the given path is a directory (true or false)
path           The path of the file system entry to test
```

Details

The file system path may be absolute or relative to the current working system path.

Example 1

```
print("Is directory: ", fs.is_dir("/usb1/"))
```

Because /usb1/ is always the root directory of an inserted flash drive, you can use this command to verify that USB flash drive is inserted.

Example 2

```
if fs.is_dir("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.

Check to see if the temp directory exists.

If it does not exist, create a directory named temp.

Also see

[fs.is_file\(\)](#) (on page 14-95)

fs.is_file()

Tests whether the specified path refers to a file (as opposed to a directory).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_file("path")
```

status	true if the given path is a file; otherwise, false
--------	--

path	The path of the file system entry to test
------	---

Details

The file system path may be absolute or relative to the current working system path.

Example

```
rootDirectory = "/usb1/"
print("Is file: ", fs.is_file(rootDirectory))
```

Insert a USB flash drive into the front panel of the instrument.

Set rootDirectory to be the USB port.

Check to see if rootDirectory is a file. Because rootDirectory was set up as a directory, the return is false.

Also see

[fs.is_dir\(\)](#) (on page 14-94)

fs.mkdir()

This function creates a directory at the specified path.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.mkdir("newPath")
```

path	The returned path of the new directory
newpath	Location (path) of where to create the new directory

Details

The directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

Example

```
if fs.is_dir("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.
Check to see if the `temp` directory exists.
If it does not exist, create a directory named `temp`.

Also see

[fs.rmdir\(\)](#) (on page 14-97)

fs.readdir()

This function returns a list of the file system entries in the directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
files = fs.readdir("path")
```

files	A table containing the names of all the file system entries in the specified directory
path	The directory path

Details

The directory path may be absolute or relative to the current working directory.

This command is nonrecursive. For example, entries in subfolders are not returned.

An error is logged to the error queue if the given path does not exist or does not represent a directory.

Example 1

```

rootDirectory = "/usb1/"
entries = fs.readdir(rootDirectory)
count = table.getn(entries)
print("Found a total of "..count.." files and directories")
for i = 1, count do
    print(entries[i])
end

```

Insert a USB flash drive into the front panel of the instrument.

Set `rootDirectory` to be the USB port.

Set `entries` as the variable for the file system entries in `rootDirectory`.

Return the number of files and directories in the directory.

Also see

None

`fs.rmdir()`

This function removes a directory from the file system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
fs.rmdir("path")
```

<code>path</code>	The path of the directory to remove
-------------------	-------------------------------------

Details

This path may be absolute or relative to the present working directory.

An error is logged to the error queue if the given path does not exist, or does not represent a directory, or if the directory is not empty.

Example

```

rootDirectory = "/usb1/"
tempDirectoryName = "temp"
if fs.is_dir(rootDirectory..tempDirectoryName) == false then
    fs.mkdir(rootDirectory..tempDirectoryName)
end
fs.rmdir(rootDirectory..tempDirectoryName)

```

Insert a USB flash drive into the front panel of the instrument.

Set `rootDirectory` to be the USB port.

Set `tempDirectoryName` to be equivalent to `temp`.

Check to see if `tempDirectoryName` exists.

If it does not exist, create a directory named `temp`.

Remove the directory.

Also see

[fs.mkdir\(\)](#) (on page 14-96)

gpib.address

This attribute contains the GPIB address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Nonvolatile memory	18

Usage

```
address = gpib.address  
gpib.address = address
```

address	The GPIB address of the instrument (1 to 30)
---------	--

Details

The address can be set to any address value from 1 to 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow sufficient time for the command to be processed before attempting to communicate with the instrument again.

The `reset()` function does not affect the GPIB address.

Example

```
gpib.address = 26  
address = gpib.address  
print(address)
```

Sets the GPIB address and reads the address. Output: 26

Also see

[GPIB setup](#) (on page 2-10)

lan.ipconfig()

This function specifies the LAN configuration for the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No	Rear panel LAN reset	Nonvolatile memory	lan.MODE_AUTO

Usage

```
method, ipV4Address, subnetMask, gateway = lan.ipconfig()
lan.ipconfig(method)
lan.ipconfig(method, "ipV4Address")
lan.ipconfig(method, "ipV4Address", "subnetMask")
lan.ipconfig(method, "ipV4Address", "subnetMask", "gateway")
```

method	The method for configuring LAN settings; it can be one of the following values: lan.MODE_AUTO: The instrument automatically assigns LAN settings lan.MODE_MANUAL: You specify the LAN settings
ipV4Address	LAN IP address; must be a string specifying the IP address in dotted decimal notation
subnetMask	The LAN subnet mask; must be a string in dotted decimal notation
gateway	The LAN default gateway; must be a string in dotted decimal notation

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used.

Example

```
lan.ipconfig(lan.MODE_AUTO)
print(lan.ipconfig())
lan.ipconfig(lan.MODE_MANUAL, "192.168.0.7", "255.255.240.0", "192.168.0.3")
print(lan.ipconfig())
Set the IP configuration method to automatic. Request the IP configuration. Example output:
lan.MODE_AUTO 134.63.78.136 255.255.254.0 134.63.78.1
Set the IP configuration method to manual. Request the IP configuration. Output:
lan.MODE_MANUAL 192.168.0.7 255.255.240.0 192.168.0.3
```

Also see

None

lan.lxidomain

This attribute contains the LXI domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0

Usage

```
domain = lan.lxidomain
lan.lxidomain = domain
```

domain	The LXI domain number (0 to 255)
--------	----------------------------------

Details

This attribute sets the LXI domain number.

All outgoing LXI packets are generated with this domain number. All inbound LXI packets are ignored unless they have this domain number.

Example

print(lan.lxidomain)	Displays the LXI domain.
----------------------	--------------------------

Also see

None

lan.macaddress

This attribute describes the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	Not applicable	Not applicable	Not applicable

Usage

```
MACaddress = lan.macaddress
```

MACaddress	The MAC address of the instrument
------------	-----------------------------------

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets.

Example

print(lan.macaddress)	Returns the MAC address. For example:
-----------------------	---------------------------------------

Also see

[lan.ipconfig\(\)](#) (on page 14-99)

localnode.access

This attribute contains the type of access users have to the instrument through different interfaces.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.ACCESS_FULL

Usage

```
accessType = localnode.access
localnode.access = accessType
```

<code>accessType</code>	The type of access: <ul style="list-style-type: none"> ■ Full access for all users from all interfaces: <code>localnode.ACCESS_FULL</code> ■ Allows access by one remote interface at a time with logins required from other interfaces: <code>localnode.ACCESS_EXCLUSIVE</code> ■ Allows access by one remote interface at a time with passwords required on all interfaces: <code>localnode.ACCESS_PROTECTED</code> ■ Allows access by one interface (including the front panel) at a time with passwords required on all interfaces: <code>localnode.ACCESS_LOCKOUT</code>
-------------------------	---

Details

When access is set to full, the instrument accepts commands from any interface with no login or password.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. You do not need a password with this access.

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

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

Under any access type, if a script is running on one remote interface when a command comes in from another remote interface, the command is ignored and the message "FAILURE: A script is running, use ABORT to stop it" is generated.

Example

```
localnode.access = localnode.ACCESS_LOCKOUT
login admin
logout
```

Set the instrument access to locked out.
 Log into the interface using the default password.
 Log out of the interface.

Also see

[localnode.password](#) (on page 14-103)

localnode.gettime()

This function retrieves the instrument date and time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.gettime()
```

Details

The time is returned in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

```
print(os.date('%c', gettime()))
```

Example output:

Tue Dec 5 03:44:37 2017

Also see

[localnode.settime\(\)](#) (on page 14-106)

localnode.linefreq

This attribute contains the power line frequency setting that is used for NPLC calculations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle	Not applicable	Not applicable

Usage

```
frequency = localnode.linefreq
```

frequency

The detected line frequency: 50 or 60

Details

The instrument automatically detects the power line frequency when the instrument is powered on. Power line frequency can be 50 Hz or 60 Hz.

Example

```
frequency = localnode.linefreq
print(frequency)
```

Reads the line frequency setting.

Also see

None

localnode.model

This attribute stores the model number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
model = localnode.model
```

model	The model number of the instrument
-------	------------------------------------

Example

print(localnode.model)	Outputs the model number of the local node. For example: 2450
------------------------	--

Also see

[localnode.serialno](#) (on page 14-105)

localnode.password

This attribute stores the instrument password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	No	Rear-panel LAN reset	Nonvolatile memory	"admin"

Usage

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

password	A string that contains the instrument password (maximum 30 characters)
----------	--

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

If you forget the password, you can reset the password to the default:

1. On the front panel, press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **Password Reset**.

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

Example

localnode.password = "N3wpa55w0rd"	Changes the password to N3wpa55w0rd.
------------------------------------	--------------------------------------

Also see

[localnode.access](#) (on page 14-101)

localnode.prompts

This attribute determines if the instrument generates prompts in response to command messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.DISABLE

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting	Do not generate prompts: localnode.DISABLE Generate prompts: localnode.ENABLE
-----------	--

Details

When the prompting mode is enabled, the instrument generates prompts when the instrument is ready to take another command. Because the prompt is not generated until the previous command completes, enabling prompts provides handshaking with the instrument to prevent buffer overruns.

When prompting is enabled, the instrument might generate the following prompts:

- **TSP>**. The standard prompt, which indicates that the previous command completed normally.
- **TSP?**. The prompt that is issued if there are unread entries in the event log when the prompt is issued. Like the TSP> prompt, it indicates that processing of the command is complete. It does not mean the previous command generated an error, only that there were still errors in the event log when command processing completed.
- **>>>**. The continuation prompt, which occurs when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the present command.

Commands do not generate prompts. The instrument generates prompts in response to command completion.

Prompts are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable prompts when the LAN connection is active, they will not be enabled for a subsequent USB connection.

NOTE

Do not disable prompting when using Test Script Builder. Test Script Builder requires prompts and sets the prompting mode automatically. If you disable prompting, the instrument will stop responding when you communicate using Test Script Builder because it is waiting for a common complete prompt from Test Script Builder.

Example

localnode.prompts = localnode.ENABLE	Enable prompting.
--------------------------------------	-------------------

Also see

[tslink.initialize\(\)](#) (on page 14-296)

localnode.prompts4882

This attribute enables and disables the generation of prompts for IEEE Std 488.2 common commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.ENABLE

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

<i>prompting</i>	IEEE Std 488.2 prompting mode: <ul style="list-style-type: none"> ■ Disable prompting: localnode.DISABLE ■ Enable prompting: localnode.ENABLE
------------------	---

Details

When this attribute is enabled, the IEEE Std 488.2 common commands generate prompts if prompting is enabled with the `localnode.prompts` attribute. If `localnode.prompts4882` is enabled, limit the number of `*trg` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When this attribute is disabled, IEEE Std 488.2 common commands will not generate prompts. When using the `*trg` command with a script that executes `trigger.wait()` repeatedly, disable prompting to avoid problems associated with the command interface input queue filling.

Example

```
localnode.prompts4882 = localnode.DISABLE
```

Disables IEEE Std 488.2 common command prompting.

Also see

[localnode.prompts](#) (on page 14-104)

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
```

<i>serialno</i>	The serial number of the instrument
-----------------	-------------------------------------

Details

This indicates the instrument serial number.

Example

```
display.clear()
display.settext(display.TEXT2, "Serial #: " ..localnode.serialno)
display.changescreen(display.SCREEN_USER_SWIPE)
Clears the instrument display.
Places the serial number of this instrument on the bottom line of the USER swipe screen display. Displays the
USER swipe screen.
```

Also see

[localnode.model](#) (on page 14-103)
[localnode.version](#) (on page 14-108)

localnode.settime()

This function sets the date and time of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.settime()
localnode.settime(year, month, day, hour, minute, second)
localnode.settime(hour, minute, second)
localnode.settime(os.time({year, month, day}))
localnode.settime(os.time({year = year, month = month, day = day, hour = hour, min =
minute, sec = second}))
```

year	Year; must be more than 1970
month	Month (1 to 12)
day	Day (1 to 31)
hour	Hour in 24-hour time format (0 to 23)
minute	Minute (0 to 59)
second	Second (0 to 59)

Details

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

When called without a parameter (the first form), the function returns the current time.

Example 1

localnode.settime(2017, 12, 5, 15, 48, 20) print(localnode.settime())	Sets the date and time to December 5, 2017 at 3:48:20 pm and verifies the time. Output: Tue Dec 5 15:48:20 2017
--	---

Example 2

```
systemTime = os.time({year = 2018,
                     month = 3,
                     day = 31,
                     hour = 14,
                     min = 25})
localnode.settime(systemTime)
print(os.date('%c', gettime()))
```

Sets the date and time to Mar 31, 2018 at 2:25 pm.
Output:
Sat Mar 31 14:25:00 2018

Also see

[localnode.gettime\(\)](#) (on page 14-102)

localnode.showevents

This attribute sets whether or not the instrument automatically outputs generated events to the remote interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	0 (no events sent)

Usage

```
errorMode = localnode.showevents
localnode.showevents = errorMode
```

errorMode	The errors that are returned: <ul style="list-style-type: none"> ■ No events: 0 ■ Errors only: 1 (eventlog.SEVER_ERROR) ■ Warnings only: 2 (eventlog.SEVER_WARN) ■ Errors and warnings: 3 (eventlog.SEVER_ERROR eventlog.SEVER_WARN) ■ Information only: 4 (eventlog.SEVER_INFO) ■ Information and errors: 5 (eventlog.SEVER_INFO eventlog.SEVER_ERROR) ■ Warnings and information: 6 (eventlog.SEVER_INFO eventlog.SEVER_WARN) ■ All events: 7 (eventlog.SEVER_ALL)
-----------	---

Details

Enable this attribute to have the instrument output generated events to the remote interface.

Events are output after a command message is executed but before prompts are issued (if prompts are enabled with `localnode.prompts`).

If this attribute is disabled, use `eventlog.next()` to retrieve unread events from the event log.

Events are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable show events when the GPIB connection is active, they will not be enabled for a subsequent USB connection.

Example

```
localnode.showevents = eventlog.SEVER_ERROR | eventlog.SEVER_INFO
trigger.digin[3].edge = trigger.EDGE_EITHER
Send generated error and warning messages.
Example output if the edge cannot be sent to either:
1805, Settings conflict: setting input edge when line 3 set for digital
```

Also see

[eventlog.clear\(\)](#) (on page 14-79)
[localnode.prompts](#) (on page 14-104)

localnode.version

This attribute stores the firmware version of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
version = localnode.version
```

version	Instrument version level
---------	--------------------------

Details

This attribute indicates the version number of the firmware that is presently running in the instrument.

Example

print(localnode.version)	Outputs the present version level. Example output: 1.0.0a
--------------------------	--

Also see

[localnode.model](#) (on page 14-103)
[localnode.serialno](#) (on page 14-105)

node[N].execute()

This function starts test scripts on a remote TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
node[N].execute( "scriptCode" )
```

N	The node number of this instrument (1 to 63)
scriptCode	A string containing the source code

Details

This command is only applicable to TSP-Link systems. You can use this command to use the remote master node to run a script on the specified node. This function does not run test scripts on the master node; only on the subordinate node when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

Example 1

<code>node[2].execute(sourcecode)</code>	Runs script code on node 2. The code is in a string variable called <code>sourcecode</code> .
--	---

Example 2

<code>node[3].execute("x = 5")</code>	Runs script code in string constant ("x = 5") to set <code>x</code> equal to 5 on node 3.
---------------------------------------	---

Example 3

<code>node[32].execute(TestDut.source)</code>	Runs the test script stored in the variable <code>TestDut</code> (previously stored on the master node) on node 32.
---	---

Also see

[tsplink.group](#) (on page 14-296)

node[N].getglobal()

This function returns the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = node[N].getglobal("name")
```

<code>value</code>	The value of the variable
<code>N</code>	The node number of this instrument (1 to 63)
<code>name</code>	The global variable name

Details

This function retrieves the value of a global variable from the run-time environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link® network.

Example

<code>print(node[5].getglobal("test_val"))</code>	Retrieves and outputs the value of the global variable named <code>test_val</code> from node 5.
---	---

Also see

[node\[N\].setglobal\(\)](#) (on page 14-110)

node[N].setglobal()

This function sets the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
node[N].setglobal("name", value)
```

N	The node number of this instrument (1 to 63)
name	The global variable name to set
value	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-Link®.

Example

node[3].setglobal("x", 5)	Sets the global variable x on node 3 to the value of 5.
---------------------------	---

Also see

[node\[N\].getglobal\(\)](#) (on page 14-109)

opc()

This function sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
opc( )
```

Details

This function causes the operation complete bit in the Status Event Status Register to be set when all previously started local overlapped commands are complete.

Note that each node independently sets its operation complete bits in its own status model. Any nodes that are not actively performing overlapped commands set their bits immediately. All remaining nodes set their own bits as they complete their own overlapped commands.

Example

```
opc()
waitcomplete()
print("1")
```

Output:
1

Also see

[*OPC](#) (on page 15-6)
[Status model](#) (on page 16-1)
[waitcomplete\(\)](#) (on page 14-318)

print()

This function generates a response message.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like other scripting environments, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments is separated with a tab character.

Numbers are printed using the `format.asciiprecision` attribute. If you want use Lua formatting, print the return value from the `tostring()` function.

Example 1

```
x = 10
print(x)
```

Example of an output response message:
10

Note that your output might be different if you set your ASCII precision setting to a different value.

Example 2

```
x = true
print(tostring(x))
```

Example of an output response message:
true

Also see

[format.asciiprecision](#) (on page 14-90)

printbuffer()

This function prints data from tables or reading buffer subtables.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

<i>startIndex</i>	Beginning index of the buffer to print; this must be more than one and less than <i>endIndex</i>
<i>endIndex</i>	Ending index of the buffer to print; this must be more than <i>startIndex</i> and less than the index of the last entry in the tables
<i>bufferVar</i>	Name of first table or reading buffer subtable to print; may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>bufferVar2</i>	Second table or reading buffer subtable to print; may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>bufferVarN</i>	The last table or reading buffer subtable to print; may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
...	One or more tables or reading buffer subtables separated with commas

Details

If *startIndex* is set to less than 1 or if *endIndex* is more than the size of the index, 9.910000e+37 is returned for each value outside the allowed index and an event is generated.

If overlapped commands use the specified reading buffers and the commands are not complete (at least to the specified index), this function outputs data as it becomes available.

When there are outstanding overlapped commands to acquire data, *n* refers to the index that the last entry in the table has after all the readings have completed.

If you pass a reading buffer instead of a reading buffer subtable, the default subtable for that reading buffer is used.

This command generates a single response message that contains all data.

The output of printbuffer() is affected by the data format selected by `format.data`. If you set `format.data` to `format.REAL32` or `format.REAL64`, you have fewer options for buffer elements. With these formats, the only buffer elements available are `readings`, `relativetimes`, and `extravalues`. If you request a buffer element that is not permitted for the selected data format, the instrument returns 9.91e37.

You can use the *bufferVar* attributes that are listed in the following table with the print buffer command. For example, if `testData` is the buffer, you can use `testData.dates` attribute to print the date of each reading in the `testData` buffer.

You can use *bufferVar.n* to retrieve the number of readings in the specified reading buffer.

Attribute	Description
<i>bufferVar.readings</i>	The readings stored in a specified reading buffer. See bufferVar.readings (on page 14-36).
<i>bufferVar.dates</i>	The dates of readings stored in the reading buffer. See bufferVar.dates (on page 14-25).
<i>bufferVar.statuses</i>	The status values of readings in the reading buffer. See bufferVar.statuses (on page 14-44).
<i>bufferVar.formattedreadings</i>	The stored readings formatted as they appear on the front-panel display. See bufferVar.formattedreadings (on page 14-32).
<i>bufferVar.sourceformattedvalues</i>	The source levels formatted as they appear on the front-panel display when the readings in the reading buffer were acquired. See bufferVar.sourceformattedvalues (on page 14-39).
<i>bufferVar.sourcevalues</i>	The source levels that were being output when readings in the reading buffer were acquired. See bufferVar.sourcevalues (on page 14-42).
<i>bufferVar.sourcessatuses</i>	The source status conditions of the instrument for the reading point. See bufferVar.sourcessatuses (on page 14-40).
<i>bufferVar.times</i>	The time when the instrument made the readings. See bufferVar.times (on page 14-46).
<i>bufferVar.timestamps</i>	The timestamps of readings stored in the reading buffer. See bufferVar.timestamps (on page 14-47).
<i>bufferVar.relativetimesamps</i>	The timestamps, in seconds, when each reading occurred relative to the timestamp of reading buffer entry number 1. See bufferVar.relativetimesamps (on page 14-37).
<i>bufferVar.sourceunits</i>	The units of measure of the source. See bufferVar.sourceunits (on page 14-41).
<i>bufferVar.seconds</i>	The nonfractional seconds portion of the timestamp when the reading was stored in UTC format. See bufferVar.seconds (on page 14-38).
<i>bufferVar.fractionalseconds</i>	The fractional portion of the timestamp (in seconds) of when each reading occurred. See bufferVar.fractionalseconds (on page 14-33).
<i>bufferVar.units</i>	The unit of measure that is stored with readings in the reading buffer. See bufferVar.units (on page 14-48).

Example 1

```
reset()
testData = buffer.make(200)
format.data = format.ASCII
format.asciiprecision = 6
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.readings, testData.units,
            testData.relativetimestamps)
```

Reset the instrument.

Set the data format and ASCII precision.

Use trigger model SimpleLoop to create a 6-count loop with no delays that stores data in the reading buffer testBuffer.

Start the trigger model, wait for the commands to complete, and output the readings.

Use of testData.n (*bufferVar.n*) indicates that the instrument should output all readings in the reading buffer. In this example, testBuffer.n equals 6.

Example of output data:

```
1.10458e-11, Amp DC, 0.00000e+00, 1.19908e-11, Amp DC, 1.01858e-01, 1.19908e-11, Amp DC,
2.03718e-01, 1.20325e-11, Amp DC, 3.05581e-01, 1.20603e-11, Amp DC, 4.07440e-01, 1.20325e-11,
Amp DC, 5.09299e-01
```

Example 2

```
for x = 1, testData.n do
    printbuffer(x,x,testData, testData.units, testData.relativetimestamps)
end
```

Using the same buffer created in Example 1, output the readings, units and relative timestamps on a separate line for each reading.

```
1.10458e-11, Amp DC, 0.00000e+00
1.19908e-11, Amp DC, 1.01858e-01
1.19908e-11, Amp DC, 2.03718e-01
1.20325e-11, Amp DC, 3.05581e-01
1.20603e-11, Amp DC, 4.07440e-01
1.20325e-11, Amp DC, 5.09299e-01
```

Also see

- [bufferVar.n](#) (on page 14-35)
- [bufferVar.readings](#) (on page 14-36)
- [format.asciiprecision](#) (on page 14-90)
- [format.byteorder](#) (on page 14-91)
- [format.data](#) (on page 14-92)
- [printnumber\(\)](#) (on page 14-115)

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

<pre>format.asciiprecision = 10 x = 2.54 printnumber(x) format.asciiprecision = 3 printnumber(x, 2.54321, 3.1)</pre>	<p>Configure the ASCII precision to 10 and set <code>x</code> to 2.54. Read the value of <code>x</code> based on these settings. Change the ASCII precision to 3. View how the change affects the output of <code>x</code> and some numbers. Output: 2.54000000e+00 2.54e+00, 2.54e+00, 3.10e+00</p>
--	--

Also see

- [format.asciiprecision](#) (on page 14-90)
- [format.byteorder](#) (on page 14-91)
- [format.data](#) (on page 14-92)
- [print\(\)](#) (on page 14-111)
- [printbuffer\(\)](#) (on page 14-112)

reset()

This function resets commands to their default settings and clears the buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
reset()
reset(system)
```

<i>system</i>	If the node is the master, the entire system is reset: true Only the local group is reset: false
---------------	---

Details

The `reset()` command in its simplest form resets the entire TSP-enabled system, including the controlling node and all subordinate nodes.

If you want to reset a specific instrument, use the `node[N].reset()` command. Also use the `node[N].reset()` command to reset an instrument on a subordinate node.

When no value is specified for `system`, the default value is `true`.

You can only reset the entire system using `reset(true)` if the node is the master. If the node is not the master node, executing this command generates an error event.

Example

<code>reset(true)</code>	If the node is the master node, the entire system is reset; if the node is not the master node, an error event is generated.
--------------------------	--

Also see

[Resets](#) (on page 3-51)

script.catalog()

This function returns an iterator that can be used in a `for` loop to iterate over all the scripts stored in nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in script.catalog() do body end
```

<i>name</i>	String representing the name of the script returned from <code>script.catalog()</code>
<i>body</i>	Code that implements the body of the <code>for</code> loop to process the return names from the <code>catalog</code> command

Details

This function accesses the catalog of scripts stored in nonvolatile memory, which allows you to process all scripts in nonvolatile memory. The entries are enumerated in no particular order.

Each time the body of the function executes, *name* takes on the name of one of the scripts stored in nonvolatile memory. The `for` loop repeats until all scripts have been iterated.

Example

```
for name in script.catalog() do
    print(name)
end
```

Retrieve the catalog listing for user scripts.
`print(name)` represent the body parameter
shown in the Usage.

Also see

None

script.delete()

This function deletes a script from the run-time memory and nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.delete("scriptName")
```

<i>scriptName</i>	A string that represents the name of the script
-------------------	---

Details

When a script is deleted, the global variable referring to this script is also deleted.

You must delete an existing script before you can use the name of that script again. Scripts are not automatically overwritten.

Example

<code>script.delete("test8")</code>	Deletes a user script named <code>test8</code> from nonvolatile memory and the global variable named <code>test8</code> .
-------------------------------------	--

Also see

[Deleting a user script using a remote interface](#) (on page 13-10)
[scriptVar.save\(\)](#) (on page 14-119)

script.load()

This function creates a script from a specified file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.load("file")
scriptVar = script.load("file")
```

<i>file</i>	A string that contains the path and file name of the script file to load; if <i>scriptVar</i> is not defined, this name is used as the global variable name for this script
<i>scriptVar</i>	The created script; a global variable with this name is used to reference the script

Details

The name that is used for *scriptVar* must not already exist as a global variable. In addition, the *scriptVar* name must be a global reference and not a local variable, table, or array.

For external scripts, the root folder of the USB flash drive has the absolute path /usb1/.

Example

```
test8 = script.load("/usb1/testSetup.tsp")
```

Loads the script with the file name testSetup.tsp that is on the USB flash drive and names it test8.

Also see

None

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.run()
scriptVar()
```

<i>scriptVar</i>	The name of the variable that references the script
------------------	---

Details

The *scriptVar.run()* function runs the script referenced by *scriptVar*. You can also run the script by using *scriptVar()*.

Example

```
test8.run()
```

Runs the script referenced by the variable test8.

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory or to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.save()
scriptVar.save("filename")
```

scriptVar	The name of variable that references the script
filename	A string that contains the file name to use when saving the script to a USB flash drive

Details

The `scriptVar.save()` function saves a script to nonvolatile memory or a USB flash drive. The root folder of the USB flash drive has the absolute path `/usb1/`.

If no `filename` is specified, the script is saved to internal nonvolatile memory. If a `filename` is given, the script is saved to the USB flash drive.

If you set `scriptVar` to `autoexec`, the script is run when the instrument powers up. You must delete the existing `autoexec` script before saving the new one. Note that performing a system reset does not delete the `autoexec` script.

You can add the file extension, but it is not required. The only allowed extension is `.tsp` (see Example 2).

Example 1

test8.save()	Saves the script referenced by the variable test8 to nonvolatile memory.
--------------	--

Example 2

test8.save("/usb1/myScript.tsp")	Saves the script referenced by the variable test8 to a file named <code>myScript.tsp</code> on your USB flash drive.
----------------------------------	--

Also see

[Working with scripts](#) (on page 13-6)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
code = scriptVar.source
```

code	The body of the script
scriptVar	The name of the variable that references the script that contains the source code

Details

The body of the script is a single string with lines separated by the new line character.

Example

```
print(test7.source)
Assuming a script named test7 was created on the instrument, this example retrieves the source code.
Output:
reset()
display.settext(display.TEXT1, "Text on line 1")
display.settext(display.TEXT2, "Text on line 2")
```

Also see

[scriptVar.save\(\)](#) (on page 14-119)

smu.interlock.enable

This attribute determines if the output can be turned on when the interlock is not engaged.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	None	Nonvolatile memory	smu.OFF

Usage

```
state = smu.interlock.enable
smu.interlock.enable = state
```

state	Allow the output to be turned on when the interlock is not engaged: smu.OFF Only allow the output to be turned on if the interlock is engaged: smu.ON
-------	--

Details

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

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

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

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

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

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

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

To change the Interlock setting:

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

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

WARNING

The 2450 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

Example

```
smu.interlock.enable = smu.ON  
The source output is disabled unless the interlock is engaged.
```

Also see

[smu.interlock.tripped](#) (on page 14-122)

smu.interlock.tripped

This attribute indicates that the interlock has been tripped.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
interlockStatus = smu.interlock.tripped
```

interlockStatus	The status of the interlock: <ul style="list-style-type: none">■ smu.OFF: The interlock is not asserted and the 200 V range is limited; lower voltage ranges are available■ smu.ON: The interlock signal is asserted and all voltage ranges are available
-----------------	--

Details

This command gives you the status of the interlock. When the safety interlock signal is asserted, all voltage ranges of the instrument are available. However, when the safety interlock signal is not asserted, the 200 V range is hardware limited to a nominal output of less than ± 42 V.

When the interlock is not asserted:

- The front-panel INTERLOCK indicator is off.
- High voltage ranges are disabled.
- If you attempt to turn on the source with a voltage more than ± 21 V, an event message is generated.

Example

```
print(smu.interlock.tripped)
```

If the interlock is not asserted, returns smu.OFF.

If the interlock is asserted, returns smu.ON.

Also see

[smu.interlock.enable](#) (on page 14-120)

smu.measure.autorange

This attribute determines if the measurement range is set manually or automatically for the selected measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
autoRange = smu.measure.autorange
smu.measure.autorange = autoRange
autoRange = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_AUTO)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_AUTO, autoRange)
```

autoRange	Set the measurement range manually: smu.OFF Set the measurement range automatically: smu.ON
function	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

Autorange selects the best range in which to measure the signal that is applied to the input terminals of the instrument. When autorange is enabled, the range increases at 100 percent of range. The range decreases occur when the reading is <10 percent of nominal range.

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

smu.measure.func = smu.FUNC_DC_CURRENT smu.measure.autorange = smu.ON	Set the measurement function to current. Set the range to be set automatically.
--	--

Also see

[Ranges](#) (on page 4-38)

[smu.measure.range](#) (on page 14-159)

smu.measure.autorangehigh

When autorange is selected, this attribute represents the highest measurement range that is used when the instrument selects the measurement range automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute Resistance: (RW) Voltage and current: (R)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Resistance: 200e6 Ω

Usage

```
highRange = smu.measure.autorangehigh
smu.measure.autorangehigh = highRange
highRange = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_HIGH)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_HIGH, highRange)
```

<i>highRange</i>	The highest measurement range that is used when the range is set automatically: <ul style="list-style-type: none"> ▪ Current: 1e-8 A to 1 A ▪ Resistance: 2 Ω to 200e6 Ω ▪ Voltage: 0.02 V to 200 V
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement (read only): <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement (read only): <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command can be written to and read for resistance measurements. For current and voltage measurements, it can only be read.

For current and voltage measurements, the upper limit is controlled by the current or voltage limit.

For resistance measurements, you can use this command when automatic range selection is enabled to put an upper bound on the range that is used for resistance measurements.

The upper limit must be more than the lower limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

<pre>smu.measure.func = smu.FUNC_DC_VOLTAGE smu.measure.autorange = smu.ON print(smu.measure.autorangehigh)</pre>	Sets the measurement function to voltage and turn autorange on. Check the high range for voltage measurements.
---	--

Also see

[Ranges](#) (on page 4-38)
[reset\(\)](#) (on page 14-116)
[smu.measure.autorange](#) (on page 14-123)
[smu.reset\(\)](#) (on page 14-170)

smu.measure.autorangelow

This attribute selects the lower limit for measurements of the selected function when the range is selected automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: 10e-9 A Voltage: 20 mV Resistance: 20 Ω

Usage

```
lowRange = smu.measure.autorangerangelow
smu.measure.autorangerangelow = lowRange
lowRange = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_LOW)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_LOW, lowRange)
```

<i>lowRange</i>	The lower limit: <ul style="list-style-type: none"> ■ Current: 1e-8 to 1 A ■ Resistance: 2 to 200e6 Ω ■ Voltage: 0.02 to 200 V
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ■ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ■ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

You can use this command when automatic range selection is enabled. It prevents the instrument from selecting a range that is below this limit. Because the lowest ranges generally require longer settling times, setting the low limit that is appropriate for your application but above the lowest possible range can make measurements require less settling time.

The lower limit must be less than the upper limit.

Though you can send any value when you send this command, the instrument selects the next highest range value. For example, if you send 15 for the lowest voltage range, the instrument will be set to the 20 V range as the low limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
smu.measure.autorangelow = 2
```

Sets the low range for voltage measurements to 2 V.

Also see

[Ranges](#) (on page 4-38)

[smu.measure.autorange](#) (on page 14-123)

smu.measure.autorangerebound

This attribute determines if the instrument restores the measure range to match the limit range after making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.autorangerebound
smu.measure.autorangerebound = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE_REBOUND)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE_REBOUND, state)
```

state	Do not restore the measurement range after each measurement: smu.OFF Restore the measurement range after each measurement: smu.ON
function	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

The effective source limit is the lesser of either the programmed source limit or 105% of the active measure range. If you use fixed measure ranges, the instrument prevents you from selecting different limit and measure ranges. However, if measure autorange is selected, it is possible for the autorange process to cause the ranges to differ because the instrument may go down to a range that is lower than the one on which the source limit is programmed. This causes the effective source limit to drop to 105% of the newly selected measure range. For example, the output may be limited if you make a measurement that causes the range to be lowered and then increase the source level or make a change to the device under test or an external source. In this situation, the source voltage or current is limited to conform with the lower measurement range until another measurement causes the instrument to select a higher range to accommodate the new source value. If no further measurement is made, the source limit may remain limited to the present measurement range indefinitely.

When autorange rebound is enabled, it prevents the source from being limited to a value that is below the source limit. After an autoranged measurement is made, the measure range is restored to match the limit range once the autoranged measurement is complete. This ensures that the source does not limit at less than the full limit setting.

Example

smu.measure.func = smu.FUNC_DC_CURRENT smu.measure.autorange = smu.ON smu.measure.autorangerebound = smu.ON	Set the measurement function to current. Set the range to be set automatically. Set the measure range to be automatically restored to match the source limit value after each measurement.
---	--

Also see

[Ranges](#) (on page 4-38)
[smu.measure.autorange](#) (on page 14-123)

smu.measure.autozero.enable

This attribute enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
state = smu.measure.autozero.enable
smu.measure.autozero.enable = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_AUTO_ZERO)
smu.measure.setattribute(function, smu.ATTR_MEAS_AUTO_ZERO, state)
```

<i>state</i>	The status of autozero; set to one of the following values: <ul style="list-style-type: none"> ▪ Disable autozero: smu.OFF ▪ Enable autozero: smu.ON
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The 2450 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

The time to make the reference measurements is in addition to the normal measurement time. If timing is critical, such as in sweeps, you can disable autozero to avoid this time penalty.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

smu.measure.func = smu.FUNC_DC_VOLTAGE smu.measure.autozero.enable = smu.OFF	Set autozero off for voltage measurements.
---	--

Also see

[smu.measure.autozero.once\(\)](#) (on page 14-128)
[smu.measure.nplc](#) (on page 14-157)

smu.measure.autozero.once()

This function causes the instrument to refresh the reference and zero measurements once.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.autozero.once()
```

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting for the selected function.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

smu.measure.autozero.once()	Do a one-time refresh of the reference and zero measurements.
-----------------------------	---

Also see

[Automatic reference measurements](#) (on page 4-44)
[smu.measure.autozero.enable](#) (on page 14-127)

smu.measure.configlist.catalog()

This function returns the name of one measure configuration list that is stored on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.catalog()
```

Details

You can use this command to retrieve the names of measure configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no measure configuration lists have been created for the instrument.

Example

<code>print(smu.measure.configlist.catalog())</code>	Request the name of one measure configuration list that is stored in the instrument. Send the command again until it returns <code>nil</code> to get all stored lists.
<code>print(smu.measure.configlist.catalog())</code>	If there are two configuration lists on the instrument. Example output: <code>testMeasList</code>
<code>print(smu.measure.configlist.catalog())</code>	<code>myMeasList</code>
<code>print(smu.measure.configlist.catalog())</code>	<code>nil</code>

Also see

[Configuration lists](#) (on page 4-84)
[createconfigscript\(\)](#) (on page 14-54)
[smu.measure.configlist.create\(\)](#) (on page 14-129)

smu.measure.configlist.create()

This function creates an empty measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.create("listName")
```

<code>listName</code>	A string that represents the name of a measure configuration list
-----------------------	---

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the `store` command.

Configuration lists are not saved when the instrument is turned off. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Configuration list names must be unique. For example, the name of a source configuration list cannot be the same as the name of a measure configuration list.

Example

<code>smu.measure.configlist.create("MyMeasList")</code>
Create a measure configuration list named <code>MyMeasList</code> .

Also see

[Configuration lists](#) (on page 4-84)
[smu.measure.configlist.catalog\(\)](#) (on page 14-128)
[smu.measure.configlist.delete\(\)](#) (on page 14-130)
[smu.measure.configlist.query\(\)](#) (on page 14-131)
[smu.measure.configlist.recall\(\)](#) (on page 14-132)
[smu.measure.configlist.size\(\)](#) (on page 14-133)
[smu.measure.configlist.store\(\)](#) (on page 14-134)

smu.measure.configlist.delete()

This function deletes a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.delete("listName")
smu.measure.configlist.delete("listName", index)
```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

smu.measure.configlist.delete("myMeasList")	Delete a measure configuration list named myMeasList.
smu.measure.configlist.delete("myMeasList", 2)	Delete configuration index 2 from the measure configuration list named myMeasList.

Also see

[Configuration lists](#) (on page 4-84)
[createconfigscript\(\)](#) (on page 14-54)
[smu.measure.configlist.create\(\)](#) (on page 14-129)

smu.measure.configlist.query()

This function returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.query("listName", index)
smu.measure.configlist.query("listName", index, "fieldSeparator")
```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list
<i>fieldSeparator</i>	String that represents the separator for the data; use one of the following: <ul style="list-style-type: none"> ▪ Comma (default): , ▪ Semicolon: ; ▪ New line: \n

Details

This command recalls data for one configuration index.

For additional information about the information this command recalls when using a configuration list query command, see [Settings stored in a measure configuration index](#) (on page 4-99).

Example

```
print(smu.measure.configlist.query("testMeasList", 2, "\n"))

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Example output:
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 1.000000e-08
smu.measure.autorange = smu.ON
smu.measure.autorangelow = 1.000000e-08
smu.measure.autozero.enable = smu.ON
smu.measure.displaydigits = smu.DIGITS_5_5
smu.measure.filter.enable = smu.OFF
smu.measure.filter.count = 10
smu.measure.filter.type = smu.FILTER_REPEAT_AVG
smu.measure.limit[1].autoclear = smu.ON
smu.measure.limit[1].enable = smu.OFF
smu.measure.limit[1].high.value = 1.000000e+00
smu.measure.limit[1].low.value = -1.000000e+00
smu.measure.limit[2].autoclear = smu.ON
smu.measure.limit[2].enable = smu.OFF
smu.measure.limit[2].high.value = 1.000000e+00
smu.measure.limit[2].low.value = -1.000000e+00
smu.measure.math.enable = smu.OFF
smu.measure.math.format = smu.MATH_PERCENT
smu.measure.math.mxb.bfactor = 0.000000e+00
smu.measure.math.mxb.mfactor = 1.000000e+00
```

```

smu.measure.math.percent = 1.000000e+00
smu.measure.userdelay[1] = 0.000000e+00
smu.measure.userdelay[2] = 0.000000e+00
smu.measure.userdelay[3] = 0.000000e+00
smu.measure.userdelay[4] = 0.000000e+00
smu.measure.userdelay[5] = 0.000000e+00
smu.measure.nplc = 1.000000e+00
smu.measure.offsetcompensation = smu.OFF
smu.measure.sense = smu.SENSE_2WIRE
smu.terminals = smu.TERMINALS_FRONT
smu.measure.unit = smu.UNIT_AMP
smu.measure.rel.enable = smu.OFF
smu.measure.rel.level = 0.000000e+00

```

Also see

- [Configuration lists](#) (on page 4-84)
[createconfigscript\(\)](#) (on page 14-54)
[smu.measure.configlist.create\(\)](#) (on page 14-129)

smu.measure.configlist.recall()

This function recalls a configuration index in a measure configuration list and an optional source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```

smu.measure.configlist.recall("listName")
smu.measure.configlist.recall("listName", index)
smu.measure.configlist.recall("listName", index, "sourceListName")
smu.measure.configlist.recall("listName", index, "sourceListName", sourceIndex)

```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the measure configuration list
<i>sourceListName</i>	A string that represents the name of a source configuration list
<i>sourceIndex</i>	A number that defines a specific configuration index in the source configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a measure configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified measure configuration list.

You can optionally specify a source configuration list and index to recall with the measure settings. If you do not specify a source index, the source index defaults to match the measure index. Specify a measure and source list together with this command to allow the instrument to coordinate the application of the settings in the two lists appropriately. If you do not need the source and measure configuration lists coordinated, you can specify just the measure configuration list with this command and use the [smu.source.configlist.recall\(\)](#) (on page 14-176) command to recall source settings separately in your application.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, an error message is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. A single index stores the settings associated with a single measure function.

NOTE

To recall a source configuration list separately (not with this command), recall the source configuration list before the measure configuration list. This order ensures that dependencies between source and measure settings will be properly handled.

This command recalls data for one configuration index from the specified measure configuration list and from the source configuration list (if specified).

For additional information about the information this command recalls when using a configuration list query command, see [Settings stored in a measure configuration index](#) (on page 4-99).

Example

<code>smu.measure.configlist.recall("MyMeasList")</code>	Because an index was not specified, this command recalls configuration index 1 from a configuration list named MyMeasList.
<code>smu.measure.configlist.recall("MyMeasList" , 5)</code>	Recalls configuration index 5 in a configuration list named MyMeasList.

Also see

- [Configuration lists](#) (on page 4-84)
- [createconfigscript\(\)](#) (on page 14-54)
- [smu.measure.configlist.create\(\)](#) (on page 14-129)
- [smu.measure.configlist.store\(\)](#) (on page 14-134)

smu.measure.configlist.size()

This function returns the size (number of configuration indexes) of a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

<code>indexCount = smu.measure.configlist.size("listName")</code>	
<code>indexCount</code>	A number that represents the total count of indexes stored in the specified measure configuration list
<code>listName</code>	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration indexes) of a measure configuration list.

The size of the list is equal to the number of configuration indexes in a configuration list.

Example

```
print(smu.measure.configlist.size("testMeasList"))
Returns the number of configuration indexes in a measure configuration list named testMeasList.
Example output:
1
```

Also see

[Configuration lists](#) (on page 4-84)
[createconfigscript\(\)](#) (on page 14-54)
[smu.measure.configlist.create\(\)](#) (on page 14-129)

smu.measure.configlist.store()

This function stores the active measure settings into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.configlist.store("listName")
smu.measure.configlist.store("listName", index)
```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active measure settings to a configuration index in a configuration list. If the index parameter is not provided, the new settings are appended to the end of the list. The index only stores the active settings for a single active measure function.

Configuration lists are not saved when the instrument is turned off or reset. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Refer to [Settings stored in a measure configuration index](#) (on page 4-99) for a complete list of measure settings that the instrument stores.

Example

smu.measure.configlist.store("MyConfigList")	Stores the active settings of the instrument to the end of the configuration list MyConfigList.
smu.measure.configlist.store("MyConfigList", 5)	Stores the active settings of the instrument to configuration index 5 in the measure configuration list MyConfigList.

Also see

[Configuration lists](#) (on page 4-84)
[createconfigscript\(\)](#) (on page 14-54)
[smu.measure.configlist.create\(\)](#) (on page 14-129)

smu.measure.configlist.storefunc()

This function allows you to store the settings for a measure function into a measure configuration list whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
smu.measure.configlist.storefunc("listName", function)
smu.measure.configlist.storefunc("listName", function, index)
```

<i>listName</i>	Name of the configuration list in which to store the function settings
<i>function</i>	The measure function settings to save into the configuration list: <ul style="list-style-type: none"> ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE
<i>index</i>	The number of the configuration list index in which to store the settings

Details

You must create the configuration list before using this command.

If *index* is not specified, the settings are stored to the next available index in the configuration list.

Example

```
smu.measure.configlist.create("MyMeasList")
smu.measure.configlist.storefunc("MyMeasList", smu.FUNC_DC_VOLTAGE)

Create a measure configuration list named MyMeasList.
Store the attributes for the DC Voltage settings in index 1.
```

Also see

[smu.measure.configlist.create\(\)](#) (on page 14-129)
[smu.measure.setattribute\(\)](#) (on page 14-166)

smu.measure.count

This attribute sets the number of measurements to make when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
count = smu.measure.count
smu.measure.count = count
count = smu.measure.getattribute(function, smu.ATTR_MEAS_COUNT)
smu.measure.setattribute(function, smu.ATTR_MEAS_COUNT, count)
```

<i>count</i>	Number of measurements (1 to 300,000 or buffer capacity)
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command sets the number of measurements that are made when a measurement is requested. This command does not affect the trigger model.

When `smu.measure.count` or if the function for `smu.measure.setattribute` is the active function, this command sets the count for all measure functions. When you send `smu.measure.setattribute` for a function that is not active, only the count for the specified function is changed.

If you set the count to a value that is larger than the capacity of the reading buffer and the buffer fill mode is set to continuous, the buffer wraps until the number of readings specified have occurred. The earliest readings in the count are overwritten. If the buffer is set to fill once, readings stop when the buffer is filled, even if the count is not complete.

NOTE

To get better performance from the instrument, use the SimpleLoop trigger-model template instead of using the count command.

Example 1

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1
smu.measure.count = 200

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

-- Turn on output and initiate readings
smu.source.output = smu.ON
smu.measure.read(defbuffer1)

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
    print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

-- Discharge the capacitor to 0 V and turn off the output
smu.source.level = 0
delay(2)
smu.source.output = smu.OFF
```

This example uses `smu.measure.count` to do a capacitor test. This outputs 200 readings that are similar to the following output:

```
Rdg # Time (s)      Current (A)
1   0     8.5718931952528e-11
2   0.151875    1.6215984111057e-10
3   0.303727    1.5521139928865e-10
...
198 29.91579194    1.5521250951167e-10
199 30.067648716   1.4131290582142e-10
200 30.219497716   1.5521067764368e-10
```

Example 2

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

-- Turn on output and initiate readings
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
    print(i, defbuffer1.relativetimes[i], defbuffer1[i])
end

-- Discharge the capacitor to 0 V and turn off the output
smu.source.level = 0
delay(2)
smu.source.output = smu.OFF
```

This example uses the SimpleLoop trigger-model template to do a capacitor test. This also outputs 200 readings that are similar to the following:

Rdg #	Time (s)	Current (A)
1	0	8.5718931952528e-11
2	0.151875	1.6215984111057e-10
3	0.303727	1.5521139928865e-10
.	.	.
198	29.91579194	1.5521250951167e-10
199	30.067648716	1.4131290582142e-10
200	30.219497716	1.5521067764368e-10

Also see

[smu.measure.read\(\)](#) (on page 14-160)
[trigger.model.load\(\) — SimpleLoop](#) (on page 14-248)

smu.measure.displaydigits

This attribute determines the number of digits that are displayed for measurements on the front panel for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.DIGITS_5_5

Usage

```
digits = smu.measure.displaydigits
smu.measure.displaydigits = digits
digits = smu.measure.getattribute(function, smu.ATTR_MEAS_DIGITS)
smu.measure.setattribute(function, smu.ATTR_MEAS_DIGITS, digits)
```

<i>digits</i>	6½ display digits: smu.DIGITS_6_5 5½ display digits: smu.DIGITS_5_5 4½ display digits: smu.DIGITS_4_5 3½ display digits: smu.DIGITS_3_5
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

The change in digits occurs the next time a measurement is made.

To change the number of digits returned in a remote command reading, use `format.asciiprecision`.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.displaydigits = smu.DIGITS_6_5
```

Set the measurement function to voltage with a front-panel display resolution of 6½.

Also see

[format.asciiprecision](#) (on page 14-90)

smu.measure.filter.count

This attribute sets the number of measurements that are averaged when filtering is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	10

Usage

```
filterCount = smu.measure.filter.count
smu.measure.filter.count = filterCount
filterCount = smu.measure.getattribute(function, smu.ATTR_MEAS_FILTER_COUNT)
smu.measure.setattribute(function, smu.ATTR_MEAS_FILTER_COUNT, filterCount)
```

<i>filterCount</i>	The number of readings required for each filtered measurement (1 to 100)
<i>function</i>	<p>The measurement function:</p> <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. When the filter count is larger, more filtering is done and the data is less noisy.

This command is set for the selected function.

Example

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.filter.count = 10
smu.measure.filter.type = smu.FILTER_MOVING_AVG
smu.measure.filter.enable = smu.ON
```

Set the measurement function to current.
 Set the averaging filter type to moving average, with a filter count of 10.
 Enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 5-23)
[smu.measure.filter.enable](#) (on page 14-141)
[smu.measure.filter.type](#) (on page 14-142)

smu.measure.filter.enable

This attribute enables or disables the averaging filter for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
filterState = smu.measure.filter.enable
smu.measure.filter.enable = filterState
filterState = smu.measure.getattribute(function, smu.ATTR_MEAS_FILTER_ENABLE)
smu.measure.setattribute(function, smu.ATTR_MEAS_FILTER_ENABLE, filterState)
```

<i>filterState</i>	The filter status: <ul style="list-style-type: none"> ■ Disable the filter: smu.OFF ■ Enable the filter: smu.ON
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

This command enables or disables the averaging filter. When this is enabled, the reading returned by the instrument is an averaged value, taken from multiple measurements. The settings of the filter count and filter type for the selected measure function determines how the reading is averaged.

Example

<pre>smu.measure.func = smu.FUNC_DC_CURRENT smu.measure.filter.count = 10 smu.measure.filter.type = smu.FILTER_MOVING_AVG smu.measure.filter.enable = smu.ON</pre>	Set the measurement function to current. Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.
--	---

Also see

[Filtering measurement data](#) (on page 5-23)
[smu.measure.filter.count](#) (on page 14-140)
[smu.measure.filter.type](#) (on page 14-142)

smu.measure.filter.type

This attribute sets the type of averaging filter that is used for the selected measure function when the measurement filter is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.FILTER_REPEAT_AVG

Usage

```
filterType = smu.measure.filter.type
smu.measure.filter.type = filterType
filterType = smu.measure.getattribute(function, smu.ATTR_MEAS_FILTER_TYPE)
smu.measure.setattribute(function, smu.ATTR_MEAS_FILTER_TYPE, filterType)
```

<i>filterType</i>	The filter type to use when filtering is enabled; set to one of the following values: <ul style="list-style-type: none"> ▪ Moving average filter: <code>smu.FILTER_MOVING_AVG</code> ▪ Repeat filter: <code>smu.FILTER_REPEAT_AVG</code>
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

You can select one of two types of averaging filters: repeating average or moving average.

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

NOTE

When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results, but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

Example

<code>smu.measure.func = smu.FUNC_DC_CURRENT smu.measure.filter.count = 10 smu.measure.filter.type = smu.FILTER_MOVING_AVG smu.measure.filter.enable = smu.ON</code>	Set the measurement function to current. Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.
--	---

Also see

[Filtering measurement data](#) (on page 5-23)
[smu.measure.filter.count](#) (on page 14-140)
[smu.measure.filter.enable](#) (on page 14-141)

smu.measure.func

This attribute selects the active measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.FUNC_DC_CURRENT

Usage

```
mFunction = smu.measure.func
smu.measure.func = mFunction
```

<i>mFunction</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>
------------------	--

Details

Set this command to the type of measurement you want to make.

Reading this command returns the measure function that is presently active.

When you select a function, settings for other commands that are related to the function become active. For example, assume that:

- You selected the current function and set the math function to reciprocal.
- You changed to the voltage function and set the math function to percent.

If you return to the current function, the math function returns to reciprocal. If you then switch from the current function to the voltage function, the math function returns to percent. All attributes that begin with `smu.measure.` are saved with the active measure function unless otherwise indicated in the command description.

Example

```

smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.math.format = smu.MATH_PERCENT
smu.measure.math.enable = smu.ON
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.math.format = smu.MATH_RECIPROCAL
smu.measure.math.enable = smu.ON
print(smu.measure.math.format)
smu.measure.func = smu.FUNC_DC_VOLTAGE
print(smu.measure.math.format)

```

Sets the instrument to measure voltage and set the math format to percent and enable the math functions.

Set the instrument to measure resistance and set the math format to reciprocal and enable the math functions.

Print the math format while the resistance measurement function is selected. The output is:

smu.MATH_RECIPROCAL

Change the function to voltage. Print the math format. The output is:

smu.MATH_PERCENT

Also see

[Making resistance measurements](#) (on page 4-25)
[Source and measure using TSP commands](#) (on page 4-32)

smu.measure.getattribute()

This function returns the setting for a function attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

<code>value = smu.measure.getattribute(function, setting)</code>	
<code>value</code>	The attribute value
<code>function</code>	The measurement function: <ul style="list-style-type: none">▪ Voltage measurement: smu.FUNC_DC_VOLTAGE▪ Current measurement: smu.FUNC_DC_CURRENT▪ Ohms measurement: smu.FUNC_RESISTANCE
<code>setting</code>	The attribute for the function; refer to smu.measure.setattribute() (on page 14-166) for available settings

Details

You can retrieve one attribute at a time.

Example

```

print(smu.measure.getattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_RANGE))
print(smu.measure.getattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_NPLC))
print(smu.measure.getattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_DIGITS))

```

Retrieve the range, NPLC, and digits settings for the DC voltage function.

Example return:

0.02
1
smu.DIGITS_4_5

Also see

[smu.measure.setattribute\(\)](#) (on page 14-166)

smu.measure.limit[Y].audible

This attribute determines if the instrument beeper sounds when a limit test passes or fails.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.AUDIBLE_NONE

Usage

```
state = smu.measure.limit[Y].audible
smu.measure.limit[Y].audible = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_AUDIBLE_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_AUDIBLE_Y, state)
```

state	When the beeper sounds: <ul style="list-style-type: none"> ■ Never: smu.AUDIBLE_NONE ■ On test failure: smu.AUDIBLE_FAIL ■ On test pass: smu.AUDIBLE_PASS
Y	Limit number: 1 or 2
function	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

The tone and length of beeper cannot be adjusted.

Example

See [smu.measure.limit\[Y\].fail](#) (on page 14-148) for an example of how to use this command.

Also see

[smu.measure.limit\[Y\].enable](#) (on page 14-147)

smu.measure.limit[Y].autoclear

This attribute indicates if the test result for limit Y should be cleared automatically or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
value = smu.measure.limit[Y].autoclear
smu.measure.limit[Y].autoclear = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y, value)
```

<i>value</i>	The auto clear setting: <ul style="list-style-type: none"> ■ Disable: smu.OFF ■ Enable: smu.ON
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

When auto clear is set to on, limit conditions are cleared automatically after each measurement. If you are making a series of measurements, the instrument shows the limit test result of the last measurement for the pass or fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the auto clear setting to off. When this is set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The auto clear setting affects both the high and low limits.

Example

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.limit[1].autoclear = smu.ON
```

Turns on autoclear for limit 1 when measuring DC current.

Also see

[smu.measure.limit\[Y\].clear\(\)](#) (on page 14-147)

smu.measure.limit[Y].clear()

This function clears the results of the limit test defined by *Y* for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.measure.limit[Y].clear()
Y           Limit number: 1 or 2
```

Details

Use this command to clear the test results of limit *Y* when the limit auto clear option is turned off. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, turn the auto clear option on.

Example

<pre>smu.measure.func = smu.FUNC_DC_CURRENT smu.measure.limit[2].clear()</pre>	Clears the test result for the high and low limit 2 for current measurements.
--	--

Also see

[smu.measure.limit\[Y\].autoclear](#) (on page 14-146)

smu.measure.limit[Y].enable

This attribute enables or disables a limit test on the measurement from the selected measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.limit[Y].enable
smu.measure.limit[Y].enable = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_ENABLE_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_ENABLE_Y, state)
```

<i>state</i>	Disable the test: smu.OFF Enable the test: smu.ON
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

This command enables or disables a limit test for the selected measurement function. When this attribute is enabled, the limit Y testing occurs on each measurement made by the instrument. Limit Y testing compares the measurements to the high-limit and low-limit values. If a measurement falls outside these limits, the test fails.

Example

<code>smu.measure.func = smu.FUNC_DC_VOLTAGE</code>	<code>smu.measure.limit[1].enable = smu.ON</code>	Enable testing for limit 1 when measuring voltage.
---	---	--

Also see

- [smu.measure.limit\[Y\].autoclear](#) (on page 14-146)
- [smu.measure.limit\[Y\].clear\(\)](#) (on page 14-147)
- [smu.measure.limit\[Y\].fail](#) (on page 14-148)
- [smu.measure.limit\[Y\].high.value](#) (on page 14-150)
- [smu.measure.limit\[Y\].low.value](#) (on page 14-151)

smu.measure.limit[Y].fail

This attribute queries the results of a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Function change	Not applicable	Not applicable

Usage

```
result = smu.measure.limit[Y].fail
result = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_FAIL_Y)
```

<i>result</i>	The results of the limit test for limit Y: <ul style="list-style-type: none"> ■ smu.FAIL_NONE: Test passed; measurement under or equal to the high limit ■ smu.FAIL_HIGH: Test failed; measurement exceeded high limit ■ smu.FAIL_LOW: Test failed; measurement exceeded low limit ■ smu.FAIL_BOTH: Test failed; measurement exceeded both limits
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

This command queries the result of a limit test for the selected measurement function.

The response message indicates if the limit test passed or how it failed (on the high or low limit).

If autoclear is set to off, reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command. To automatically clear the results, set auto clear on.

If auto clear is set to on and you are making a series of measurements, the last measurement limit determines the fail indication for the limit. If auto clear is turned off, the results return a test fail if any of one of the readings failed.

To use this attribute, you must set the limit state to on.

If the readings are stored in a reading buffer, you can use the `bufferVar.statuses` command to see the results.

Example

This example enables limits 1 and 2 for voltage, measurements. Limit 1 is checking for readings to be between 3 and 5 V, while limit 2 is checking for the readings to be between 1 and 7 V. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
reset()
-- set the instrument source current
smu.source.func = smu.FUNC_DC_CURRENT
-- set the instrument to measure voltage
smu.measure.func = smu.FUNC_DC_VOLTAGE
-- set the range to 10 V
smu.measure.range = 10
-- set the nplc to 0.1
smu.measure.nplc = 0.1
-- disable auto clearing for limit 1
smu.measure.limit[1].autoclear = smu.OFF
-- set high limit on 1 to fail if reading exceeds 5 V
smu.measure.limit[1].high.value = 5
-- set low limit on 1 to fail if reading is less than 3 V
smu.measure.limit[1].low.value = 3
--- set the beeper to sound if the reading exceeds the limits for limit 1
smu.measure.limit[1].audible = smu.AUDIBLE_FAIL
-- enable limit 1 checking for voltage measurements
smu.measure.limit[1].enable = smu.ON
-- disable auto clearing for limit 2
smu.measure.limit[2].autoclear = smu.OFF
-- set high limit on 2 to fail if reading exceeds 7 V
smu.measure.limit[2].high.value = 7
-- set low limit on 2 to fail if reading is less than 1 V
smu.measure.limit[2].low.value = 1
-- enable limit 2 checking for voltage measurements
smu.measure.limit[2].enable = smu.ON
-- set the measure count to 50
smu.measure.count = 50
-- create a reading buffer that can store 100 readings
LimitBuffer = buffer.make(100)
-- make 50 readings and store them in LimitBuffer
smu.measure.read(LimitBuffer)
-- Check if any of the 50 readings were outside of the limits
print("limit 1 results = " .. smu.measure.limit[1].fail)
print("limit 2 results = " .. smu.measure.limit[2].fail)
-- clear limit 1 conditions
smu.measure.limit[1].clear()
-- clear limit 2 conditions
smu.measure.limit[2].clear()
```

```

Example output that shows all readings are within limit values (all readings between 3 V and 5 V):
limit 1 results = smu.FAIL_NONE
limit 2 results = smu.FAIL_NONE
Example output showing at least one reading failed limit 1 high values (a 6 V reading would cause this
condition or a reading greater than 5 V but less than 7 V):
limit 1 results = smu.FAIL_HIGH
limit 2 results = smu.FAIL_NONE
Example output showing at least one reading failed limit 1 and 2 low values (a 0.5 V reading would cause this
condition or a reading less than 1 V):
limit 1 results = smu.FAIL_LOW
limit 2 results = smu.FAIL_LOW

```

Also see

[bufferVar.statuses](#) (on page 14-44)
[Limit testing and binning](#) (on page 4-67)
[smu.measure.limit\[Y\].enable](#) (on page 14-147)

smu.measure.limit[Y].high.value

This attribute specifies the upper limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```

highLimit = smu.measure.limit[Y].high.value
smu.measure.limit[Y].high.value = highLimit
highLimit = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_HIGH_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_HIGH_Y, highLimit)

```

<i>highLimit</i>	The value of the high limit (-9.99999e+11 to +9.99999e+11)
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command sets the high limit for the limit *Y* test for the selected measurement function. When limit *Y* testing is enabled, the instrument generates a fail indication when the measurement value is more than this value.

Example

See the example in [smu.measure.limit\[Y\].fail](#) (on page 14-148).

Also see

[smu.measure.limit\[Y\].enable](#) (on page 14-147)

smu.measure.limit[Y].low.value

This attribute specifies the lower limit for limit tests.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	-1

Usage

```
lowLimit = smu.measure.limit[Y].low.value
smu.measure.limit[Y].low.value = lowLimit
lowLimit = smu.measure.getattribute(function, smu.ATTR_MEAS_LIMIT_LOW_Y)
smu.measure.setattribute(function, smu.ATTR_MEAS_LIMIT_LOW_Y, lowLimit)
```

<i>lowLimit</i>	The low limit value of limit <i>Y</i> (-9.99999E+11 to 9.99999E+11)
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ■ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ■ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command sets the lower limit for the limit *Y* test for the selected measure function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

Example

See the example in [smu.measure.limit\[Y\].fail](#) (on page 14-148).

Also see

[smu.measure.limit\[Y\].enable](#) (on page 14-147)

smu.measure.math.enable

This attribute enables or disables math operations on measurements for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
value = smu.measure.math.enable
smu.measure.math.enable = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_ENABLE)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_ENABLE, value)
```

<i>value</i>	The math enable setting: <ul style="list-style-type: none"> ■ Disable: smu.OFF ■ Enable: smu.ON
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: smu.FUNC_DC_VOLTAGE ■ Current measurement: smu.FUNC_DC_CURRENT ■ Ohms measurement: smu.FUNC_RESISTANCE

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.math.format = smu.MATH_PERCENT
smu.measure.math.enable = smu.ON
```

When voltage measurements are made, the math format is enabled and set to percent.

Also see

[Calculations that you can apply to measurements](#) (on page 4-50)
[smu.measure.math.format](#) (on page 14-153)

smu.measure.math.format

This attribute specifies which math operation is performed on measurements when math operations are enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.MATH_PERCENT

Usage

```
operation = smu.measure.math.format
smu.measure.math.format = operation
operation = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_FORMAT)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_FORMAT, operation)
```

operation	Math operation to be performed on measurements: <ul style="list-style-type: none"> ▪ <code>y = mx+b</code>: <code>smu.MATH_MXB</code> ▪ Percent: <code>smu.MATH_PERCENT</code> ▪ Reciprocal: <code>smu.MATH_RECIPROCAL</code>
function	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This specifies which math operation is performed on measurements for the measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Displays measurements as the percentage of deviation from a specified reference constant.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is $1/x$, where x is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.math.format = smu.MATH_RECIPROCAL
smu.measure.math.enable = smu.ON
```

Enables the reciprocal math operation on voltage measurements.

Also see

- [Calculations that you can apply to measurements](#) (on page 4-50)
- [smu.measure.math.enable](#) (on page 14-152)
- [smu.measure.math.mxb.mfactor](#) (on page 14-155)
- [smu.measure.math.percent](#) (on page 14-156)

smu.measure.math.mxb.bfactor

This attribute specifies the offset, b, for the $y = mx + b$ operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
value = smu.measure.math.mxb.bfactor
smu.measure.math.mxb.bfactor = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_MXB_BF)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_MXB_BF, value)
```

value	The offset for the $y = mx + b$ operation; the valid range is $-1e12$ to $+1e12$
function	<p>The measurement function:</p> <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This attribute specifies the offset (b) for an $mx + b$ operation.

The $mx + b$ math operation lets you manipulate normal display readings (x) mathematically based on the calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example

<code>smu.measure.func = smu.FUNC_DC_VOLTAGE</code> <code>smu.measure.math.format = smu.MATH_MXB</code> <code>smu.measure.math.mxb.mfactor = 0.80</code> <code>smu.measure.math.mxb.bfactor = 50</code> <code>smu.measure.math.enable = smu.ON</code>	Set the measurement function to voltage. Set the math operation to mx+b. Set the scale factor for the mx +b operation to 0.80. Set the offset factor to 50. Enable the math function.
---	---

Also see

[Calculations that you can apply to measurements](#) (on page 4-50)

[smu.measure.math.enable](#) (on page 14-152)

[smu.measure.math.mxb.mfactor](#) (on page 14-155)

smu.measure.math.mxb.mfactor

This attribute specifies the scale factor, m, for the $y = mx + b$ math operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
value = smu.measure.math.mxb.mfactor
smu.measure.math.mxb.mfactor = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_MXB_MF)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_MXB_MF, value)
```

value	The scale factor; the valid range is -1e12 to +1e12
function	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command sets the scale factor (m) for an $mx + b$ operation for the selected function.

The $mx + b$ math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example

<code>smu.measure.func = smu.FUNC_DC_VOLTAGE</code> <code>smu.measure.math.format = smu.MATH_MXB</code> <code>smu.measure.math.mxb.mfactor = 0.80</code> <code>smu.measure.math.mxb.bfactor = 50</code> <code>smu.measure.math.enable = smu.ON</code>	Set the measurement function to voltage. Set the math operation to mx+b. Set the scale factor for the mx+b operation to 0.80. Set the offset factor to 50. Enable the math function.
---	--

Also see

[Calculations that you can apply to measurements](#) (on page 4-50)

[smu.measure.math.enable](#) (on page 14-152)

[smu.measure.math.mxb.bfactor](#) (on page 14-154)

smu.measure.math.percent

This attribute specifies the reference constant that is used when math operations are set to percent.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
value = smu.measure.math.percent
smu.measure.math.percent = value
value = smu.measure.getattribute(function, smu.ATTR_MEAS_MATH_PERCENT)
smu.measure.setattribute(function, smu.ATTR_MEAS_MATH_PERCENT, value)
```

<i>value</i>	The reference used when the math operation is set to percent; the range is -1e12 to +1e12
<i>function</i>	<p>The measurement function:</p> <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This is the constant that is used when the math operation is set to percent for the selected measurement function.

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

$$\text{Percent} = \left(\frac{\text{input} - \text{reference}}{\text{reference}} \right) \times 100\%$$

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.math.format = smu.MATH_PERCENT
smu.measure.math.percent = 50
smu.measure.math.enable = smu.ON
```

Set the measurement function to voltage.
 Set the math operations to percent.
 Set the percentage value to 50 for voltage measurements.
 Enable math operations.

Also see

- [Calculations that you can apply to measurements](#) (on page 4-50)
- [smu.measure.math.enable](#) (on page 14-152)
- [smu.measure.math.format](#) (on page 14-153)

smu.measure.nplc

This command sets the time that the input signal is measured for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
nplc = smu.measure.nplc
smu.measure.nplc = nplc
nplc = smu.measure.getattribute(function, smu.ATTR_MEAS_NPLC)
smu.measure.setattribute(function, smu.ATTR_MEAS_NPLC, nplc)
```

<i>nplc</i>	The number of power line cycles: 0.01 to 10
<i>function</i>	<p>The measurement function:</p> <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified as the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50). For 60 Hz, if you set the NPLC to 0.1, the measure time is 1.667 ms.

This command is set for the measurement of specific functions (current, resistance, or voltage).

The shortest amount of time results in the fastest reading rate, but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits, but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

Example

<pre>smu.measure.func = smu.FUNC_DC_VOLTAGE smu.measure.nplc = 0.5</pre>	<p>Set the measurement function to DC Voltage. Set the NPLC value to 0.5, which is 0.0083 seconds (0.5/60).</p>
--	---

Also see

[Using NPLCs to adjust speed and accuracy](#) (on page 5-9)

smu.measure.offsetcompensation

This attribute determines if offset compensation is used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.offsetcompensation
smu.measure.offsetcompensation = state
state = smu.measure.getattribute(function, smu.ATTR_MEAS_OFFSET_COMP)
smu.measure.setattribute(function, smu.ATTR_MEAS_OFFSET_COMP, state)
```

state	Disable offset compensation: smu.OFF Enable offset compensation: smu.ON
function	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

The voltage offsets caused by the presence of thermoelectric EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

This feature is only available for resistance measurements or when the `smu.measure.unit` is set to `smu.UNIT_OHM`.

Example

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.sense = smu.SENSE_4WIRE
smu.measure.offsetcompensation = smu.ON
smu.source.output = smu.ON
print(smu.measure.read())
smu.source.output = smu.OFF
```

Sets the measurement function to resistance. Set the instrument for 4-wire measurements and turn offset compensation on. Turn on the source, make a measurement, and turn the source off.
Example output:
81592000

Also see

None

smu.measure.range

This attribute determines the positive full-scale measure range.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: 100e-6 Resistance: 200,000 Voltage: 0.02

Usage

```
rangeValue = smu.measure.range
smu.measure.range = rangeValue
rangeValue = smu.measure.getattribute(function, smu.ATTR_MEAS_RANGE)
smu.measure.setattribute(function, smu.ATTR_MEAS_RANGE, rangeValue)
```

<i>rangeValue</i>	Set to the maximum expected value to be measured: <ul style="list-style-type: none"> ▪ Current: 1 nA to 1 A ▪ Resistance: 2 Ω to 200 MΩ ▪ Voltage: 0.02 V to 200 V
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

You can assign any real number using this command. The instrument selects the closest fixed range that is large enough to measure the entered number. For example, for current measurements, if you expect a reading of approximately 9 mA, set the range to 9 mA to select the 10 mA range. When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using.

This command is primarily intended to eliminate the time that is required by the instrument to automatically search for a range.

When a range is fixed, any signal greater than the entered range generates an overrange condition. When an overrange condition occurs, the front panel displays "Overflow" and the remote interface returns `9.9e+37`.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range, regardless of measurement range setting. However, the setting for the measure range is retained, and when the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

If you change the range while the output is off, the instrument does not update the hardware settings, but if you read the range setting, the return is the setting that will be used when the output is turned on. If you set a range while the output is on, the new setting takes effect immediately.

NOTE

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the selected measurement function (if supported by that function).

The range for measure functions defaults to autorange for all measure functions. If you switch from a fixed range to autorange, autorange is set to off. The range remains at the fixed range until a measurement is made, at which time the range is set to accommodate the new measurement.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 0.5
```

Set the source function to current.
Set the measurement function to voltage.
Instrument selects the 2 V measurement range.

Also see

[Ranges](#) (on page 4-38)
[smu.measure.autorange](#) (on page 14-123)

smu.measure.read()

This function makes measurements, places them in a reading buffer, and returns the last reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = smu.measure.read()
reading = smu.measure.read(bufferName)
```

reading	The last reading of the measurement process
bufferName	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; if no buffer is defined, it defaults to defbuffer1

Details

This function makes a measurement using the present function setting, stores the reading in a reading buffer, and returns the last reading.

The `smu.measure.count` attribute determines how many measurements are performed. You can also use the trigger model Simple Loop.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

NOTE

To make a power reading, use the `smu.measure.unit` command and set the units to `smu.UNIT_WATT` for the voltage or current measurement function.

Example

```
voltMeasBuffer = buffer.make(10000)
smu.measure.func = smu.FUNC_DC_VOLTAGE
print(smu.measure.read(voltMeasBuffer))
```

Create a buffer named `voltMeasBuffer`. Set the instrument to measure voltage. Make a measurement that is stored in the `voltMeasBuffer` and is also printed.

Also see

[buffer.make\(\)](#) (on page 14-14)
[Reading buffers](#) (on page 6-1)
[smu.measure.count](#) (on page 14-136)
[smu.measure.unit](#) (on page 14-168)
[trigger.model.load\(\) — SimpleLoop](#) (on page 14-248)

smu.measure.readwithtime()

This function initiates measurements and returns the last actual measurement and time information in UTC format without using the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading, seconds, fractional = smu.measure.readwithtime()
smu.measure.readwithtime(bufferName)
```

<code>reading</code>	The last reading of the measurement process
<code>seconds</code>	Seconds in UTC format
<code>fractional</code>	Fractional seconds
<code>bufferName</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>

Details

This command initiates measurements using the present function setting, stores the readings in a reading buffer, and returns the last reading.

The `smu.measure.count` attribute determines how many measurements are performed.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

Example

```
print(smu.measure.readwithtime(defbuffer1))
```

Print the last measurement and time information from defbuffer1 in UTC format, which will look similar to:
-1.405293589829e-11 1400904629 0.1950935

Also see

[smu.measure.count](#) (on page 14-136)
[trigger.model.load\(\) — SimpleLoop](#) (on page 14-248)

smu.measure.rel.acquire()

This function acquires a measurement and stores it as the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
relativeValue = smu.measure.rel.acquire()
```

relativeValue	The internal measurement acquired for the relative offset value
---------------	---

Details

This command triggers the instrument to make a new measurement for the selected function. This measurement is then stored as the new relative offset level.

When you send this command, the instrument does not apply any math, limit test, or filter settings to the measurement, even if they are set. It is a measurement that is made as if these settings are disabled.

If an error event occurs during the measurement, nil is returned and the relative offset level remains at the last valid setting.

You must change to the function for which you want to acquire a value before sending this command.

The instrument must have relative offset enabled to use the acquired relative offset value.

After executing this command, you can use the `smu.measure.rel.level` attribute to see the last relative level value that was acquired or that was set.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
rel_value = smu.measure.rel.acquire()
smu.measure.rel.enable = smu.ON
```

Acquires a relative offset level value for voltage measurements and turns the relative offset feature on.

Also see

[smu.measure.rel.enable](#) (on page 14-163)
[smu.measure.rel.level](#) (on page 14-164)

smu.measure.rel.enable

This attribute enables or disables the application of a relative offset value to the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
relEnable = smu.measure.rel.enable
smu.measure.rel.enable = relEnable
relEnable = smu.measure.getattribute(function, smu.ATTR_MEAS_REL_ENABLE)
smu.measure.setattribute(function, smu.ATTR_MEAS_REL_ENABLE, relEnable)
```

<i>relEnable</i>	Relative measurement control: <ul style="list-style-type: none"> ■ Disable relative offset: <code>smu.OFF</code> ■ Enable relative offset: <code>smu.ON</code>
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ■ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ■ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value calculated when you acquire the relative offset value.

Each returned measured relative reading is the result of the following calculation:

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

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
rel_value = smu.measure.rel.acquire()
smu.measure.rel.enable = smu.ON
```

Acquires a relative offset level value for voltage measurements and turns the relative offset feature on.

Also see

- [Relative offset \(on page 4-47\)](#)
- [smu.measure.rel.acquire\(\) \(on page 14-162\)](#)
- [smu.measure.rel.level \(on page 14-164\)](#)

smu.measure.rel.level

This attribute contains the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
relValue = smu.measure.rel.level
smu.measure.rel.level = relValue
relValue = smu.measure.getattribute(function, smu.ATTR_MEAS_REL_LEVEL)
smu.measure.setattribute(function, smu.ATTR_MEAS_REL_LEVEL, relValue)
```

relValue	Relative offset value for measurements: <ul style="list-style-type: none"> ▪ Current (A): -1.05 to 1.05 ▪ Resistance (Ω): -2.10e6 to 2.10e6 ▪ Voltage (V): -210 to 210
function	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ▪ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ▪ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

This command specifies the relative offset value that can be applied to new measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

You can set this value, or have the instrument acquire a value. If the instrument acquires the value, read this setting to return the value that was measured internally.

NOTE

If you have math, limits, or filter operations selected, you can set the relative offset value to include the adjustments made by these operations. To include these operations, set `smu.measure.rel.level` to `smu.measure.read()`. The adjustments from these operations are not used if you use the `smu.measure.rel.acquire()` function to set the relative offset level.

Example

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.rel.level = smu.measure.read()
smu.measure.rel.enable = smu.ON
```

Sets the measurement function to current, performs a current measurement, uses it as the relative offset value, and enables the relative offset for current measurements.

Also see

- [Relative offset \(on page 4-47\)](#)
- [smu.measure.rel.acquire\(\) \(on page 14-162\)](#)
- [smu.measure.rel.enable \(on page 14-163\)](#)

smu.measure.sense

This attribute selects local (2-wire) or remote (4-wire) sensing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.SENSE_2WIRE

Usage

```
sensing = smu.measure.sense
smu.measure.sense = sensing
sensing = smu.measure.getattribute(function, smu.ATTR_MEAS_SENSE)
smu.measure.setattribute(function, smu.ATTR_MEAS_SENSE, sensing)
```

<i>sensing</i>	Two-wire sensing: smu.SENSE_2WIRE Four-wire sensing: smu.SENSE_4WIRE
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

This command determines if 2-wire (local) or 4-wire (remote) sensing is used.

When you use 4-wire sensing, voltages are measured at the device under test (DUT). For the source voltage, if the sensed voltage is lower than the programmed amplitude, the voltage source increases the voltage until the sensed voltage is the same as the programmed amplitude. This compensates for IR drop in the output test leads.

Using 4-wire sensing with voltage measurements eliminates any voltage drops that may be in the test leads between the 2450 and the DUT.

When you are using 2-wire sensing, voltage is measured at the output connectors.

When you are measuring resistance, you can enable 4-wire sensing to make 4-wire resistance measurements.

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

If the output is on when you change the sense setting, the output is turned off.

Example

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

Set the measurement function to resistance.
Set the sense to 4-wire remote.

Also see

[Two-wire local sense connections](#) (on page 4-9)
[Four-wire remote sense connections](#) (on page 4-10)

smu.measure.setattribute()

This function allows you to set up a measure function whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
smu.measure.setattribute(function, setting, value)
```

<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE
<i>setting</i>	The attribute for the function; refer to Details and the tables following the examples
<i>value</i>	The attribute value

Details

The lists following the Example and Also See information show the attributes that are available for each function, with links to the descriptions of the corresponding TSP command descriptions. The options for each attribute are the same as the settings for the TSP commands.

Example

```
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_REL_LEVEL, 0.55)
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_LIMIT_HIGH_1, 0.64)
smu.measure.setattribute(smu.FUNC_DC_VOLTAGE, smu.ATTR_MEAS_LIMIT_LOW_1, 0.32)
smu.measure.configlist.create("MyMeasList")
smu.measure.configlist.storefunc("MyMeasList", smu.FUNC_DC_VOLTAGE)

Configure the DC Voltage function settings for the relative offset level, limit 1 high value, and limit 1 low value
whether or not DC Voltage is the active function.

Create a configuration list named MyMeasList.

Store the settings for the DC Voltage function in MyMeasList at index 1.
```

Also see

[smu.measure.getattribute\(\)](#) (on page 14-144)

Measure options

[Autorange](#) (on page 14-123): smu.ATTR_MEAS_RANGE_AUTO

[Autorange high limit](#) (on page 14-124): smu.ATTR_MEAS_RANGE_HIGH

[Autorange low limit](#) (on page 14-125): smu.ATTR_MEAS_RANGE_LOW

[Autorange rebound](#) (on page 14-126): smu.ATTR_MEAS_RANGE_REBOUND

[Autozero enable](#) (on page 14-127): smu.ATTR_MEAS_AUTO_ZERO

[Count](#) (on page 14-136): smu.ATTR_MEAS_COUNT

[Display digits](#) (on page 14-139): `smu.ATTR_MEAS_DIGITS`
[NPLC](#) (on page 14-157): `smu.ATTR_MEAS_NPLC`
[Offset compensation](#) (on page 14-158): `smu.ATTR_MEAS_OFFSET_COMP`
[Range](#) (on page 14-159): `smu.ATTR_MEAS_RANGE`
[Relative offset enable](#) (on page 14-163): `smu.ATTR_MEAS_REL_ENABLE`
[Relative offset level](#) (on page 14-164): `smu.ATTR_MEAS_REL_LEVEL`
[Sense \(2-wire or 4-wire\)](#) (on page 14-165): `smu.ATTR_MEAS_SENSE`
[Unit](#) (on page 14-168): `smu.ATTR_MEAS_UNIT`
[User delays](#) (on page 14-169): `smu.ATTR_MEAS_USER_DELAY_N`, where *N* is 1 to 5

Filter options

[Filter count](#) (on page 14-140): `smu.ATTR_MEAS_FILTER_COUNT`
[Filter enable](#) (on page 14-141): `smu.ATTR_MEAS_FILTER_ENABLE`
[Filter type](#) (on page 14-142): `smu.ATTR_MEAS_FILTER_TYPE`

Math options (measure)

[Enable math](#) (on page 14-152): `smu.ATTR_MEAS_MATH_ENABLE`
[b \(offset\) value](#) (on page 14-154): `smu.ATTR_MEAS_MATH_MXB_BF`
[m \(scalar\) value](#) (on page 14-155): `smu.ATTR_MEAS_MATH_MXB_MF`
[Math format](#) (on page 14-153): `smu.ATTR_MEAS_MATH_FORMAT`
[Percent](#) (on page 14-156): `smu.ATTR_MEAS_MATH_PERCENT`

Limit options (measure)

[Limit 1 audible](#) (on page 14-145): `smu.ATTR_MEAS_LIMIT_AUDIBLE_1`
[Limit 1 auto clear](#) (on page 14-146): `smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_1`
[Limit 1 enable](#) (on page 14-147): `smu.ATTR_MEAS_LIMIT_ENABLE_1`
[Limit 1 fail](#) (on page 14-148): `smu.ATTR_MEAS_LIMIT_FAIL_1`
[Limit 1 high value](#) (on page 14-150): `smu.ATTR_MEAS_LIMIT_HIGH_1`
[Limit 1 low value](#) (on page 14-151): `smu.ATTR_MEAS_LIMIT_LOW_1`
[Limit 2 audible](#) (on page 14-145): `smu.ATTR_MEAS_LIMIT_AUDIBLE_2`
[Limit 2 auto clear](#) (on page 14-146): `smu.ATTR_MEAS_LIMIT_AUTO_CLEAR_2`
[Limit 2 enable](#) (on page 14-147): `smu.ATTR_MEAS_LIMIT_ENABLE_2`
[Limit 2 fail](#) (on page 14-148): `smu.ATTR_MEAS_LIMIT_FAIL_2`
[Limit 2 high value](#) (on page 14-150): `smu.ATTR_MEAS_LIMIT_HIGH_2`
[Limit 2 low value](#) (on page 14-151): `smu.ATTR_MEAS_LIMIT_LOW_2`

smu.measure.unit

This attribute sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: smu.UNIT_AMP Resistance: smu.UNIT_OHM Voltage: smu.UNIT_VOLT

Usage

```
unitOfMeasure = smu.measure.unit
smu.measure.unit = unitOfMeasure
unitOfMeasure = smu.measure.getattribute(function, smu.ATTR_MEAS_UNIT)
smu.measure.setattribute(function, smu.ATTR_MEAS_UNIT, unitOfMeasure)
```

<i>unitOfMeasure</i>	The units of measure to be displayed for the measurement: <ul style="list-style-type: none"> ▪ Current: smu.UNIT_AMP (only available for current measurements) ▪ Resistance: smu.UNIT_OHM (available for voltage, current, or resistance measurements) ▪ Volts: smu.UNIT_VOLT (only available for voltage measurements) ▪ Power: smu.UNIT_WATT (only available for voltage or current measurements)
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ▪ Voltage measurement: smu.FUNC_DC_VOLTAGE ▪ Current measurement: smu.FUNC_DC_CURRENT ▪ Ohms measurement: smu.FUNC_RESISTANCE

Details

The change in measurement units is displayed when the next measurement is made.

Example

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.unit = smu.UNIT_WATT
```

Changes the front-panel display and buffer readings for voltage measurements to be displayed as power readings in watts.

Also see

[smu.measure.func](#) (on page 14-143)

smu.measure.userdelay[N]

This attribute sets a user-defined delay that you can use in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0 (0 s)

Usage

```
delayTime = smu.measure.userdelay[N]
smu.measure.userdelay[N] = delayTime
delayTime = smu.measure.getattribute(function, smu.ATTR_MEAS_USER_DELAY_N)
smu.measure.setattribute(function, smu.ATTR_MEAS_USER_DELAY_N, delayTime)
```

<i>delayTime</i>	The delay (0 for no delay, or 167 ns to 10 ks)
<i>N</i>	The user delay to which this time applies (1 to 5)
<i>function</i>	The measurement function; see Functions
<i>function</i>	The measurement function: <ul style="list-style-type: none"> ■ Voltage measurement: <code>smu.FUNC_DC_VOLTAGE</code> ■ Current measurement: <code>smu.FUNC_DC_CURRENT</code> ■ Ohms measurement: <code>smu.FUNC_RESISTANCE</code>

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.

The delay is specific to the selected function.

Example

```
smu.measure.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()

Set user delay for measure 1 to 5 s.
Set trigger block 1 to turn the source output on.
Set trigger block 2 to a dynamic delay that calls measure user delay 1.
Set trigger block 3 to make a measurement.
Set trigger block 4 to turn the source output off.
Set trigger block 5 to branch to block 1 ten times.
Start the trigger model.
```

Also see

[trigger.model.setblock\(\) — trigger.BLOCK_DELAY_DYNAMIC](#) (on page 14-267)

smu.reset()

This function turns off the output and resets the commands that begin with `smu.` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.reset( )
```

Details

This function turns off the output and resets the commands that begin with `smu.` to their default settings. It also deletes source and measure configuration lists.

Example

<code>smu.reset()</code>	Turns off the output and resets the SMU commands to their default settings.
---------------------------	---

Also see

[reset\(\)](#) (on page 14-116)

smu.source.autorange

This attribute determines if the range is selected manually or automatically for the selected source function or voltage source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
sourceAutorange = smu.source.autorange
smu.source.autorange = sourceAutorange
sourceAutorange = smu.source.getattribute(function, smu.ATTR_SRC_RANGE_AUTO)
smu.source.setattribute(function, smu.ATTR_SRC_RANGE_AUTO, sourceAutorange)
```

<code>sourceAutorange</code>	Disable automatic source range: <code>smu.OFF</code> Enable automatic source range: <code>smu.ON</code>
<code>function</code>	The source function: <ul style="list-style-type: none"> ▪ Current source: <code>smu.FUNC_DC_CURRENT</code> ▪ Voltage source: <code>smu.FUNC_DC_VOLTAGE</code>

Details

This command indicates the state of the range for the selected source. When automatic source range is disabled, the source range is set manually.

When automatic source range is enabled, the instrument selects the range that is most appropriate for the value that is being sourced. The output level controls the range. If you read the range after the output level is set, the instrument returns the range that the instrument chose as appropriate for that source level.

If the source range is set to a specific value from the front panel or a remote command, the setting for automatic range is set to disabled.

Only available for the current and voltage functions.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.autorange = smu.ON
```

Set the source function to current.
Set the instrument to select the source range automatically.

Also see

[smu.source.range](#) (on page 14-188)

smu.source.autodelay

This attribute enables or disables the automatic delay that occurs when the source is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
state = smu.source.autodelay
smu.source.autodelay = state
state = smu.source.getattribute(function, smu.ATTR_SRC_DELAY_AUTO)
smu.source.setattribute(function, smu.ATTR_SRC_DELAY_AUTO, state)
```

state	Disable the source auto delay: smu.OFF Enable the source auto delay: smu.ON
sourceAutorange	Disable automatic source range: smu.OFF Enable automatic source range: smu.ON
function	The source function: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE

Details

When autodelay is turned on, the actual delay that is set depends on the range.

When source autodelay is on, if you set a source delay, the autodelay is turned off.

Example

smu.source.autodelay = smu.OFF	Turn off auto delay when current is being sourced.
--------------------------------	--

Also see

[smu.source.delay](#) (on page 14-180)

smu.source.configlist.catalog()

This function returns the name of one source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu.source.configlist.catalog()`

Details

You can use this command to retrieve the names of source configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no source configuration lists have been created for the instrument.

Example

<code>print(smu.source.configlist.catalog())</code>	Request the name of one source configuration list that is stored in the instrument. Send the command again until it returns <code>nil</code> to get all stored lists.
--	---

Also see

[Configuration lists](#) (on page 4-84)

[smu.source.configlist.create\(\)](#) (on page 14-173)

smu.source.configlist.create()

This function creates an empty source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list	Configuration script	

Usage

```
smu.source.configlist.create("listName")
```

<i>listName</i>	A string that represents the name of a source configuration list
-----------------	--

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list through a power cycle, create a configuration script to save instrument settings, including any defined configuration lists.

Configuration list names must be unique. For example, the name of a source configuration list cannot be the same as the name of a measure configuration list.

Example

reset()	Create a source configuration list named MyScrLst.
smu.source.configlist.create("MyScrList")	Print the name of one configuration list stored in volatile memory.
print(smu.source.configlist.catalog())	Output: MyScrList
print(smu.source.configlist.catalog())	Print the name of one configuration list.
	Output: nil
	Nil indicates that no more configuration lists are stored.
smu.source.configlist.store("MyScrList")	Store a configuration index in MyScrList.
smu.source.configlist.store("MyScrList")	Store a configuration index in MyScrList.
print(smu.source.configlist.size("MyScrList"))	Print the number of configuration indexes in MyScrList.
	Output: 2

Also see

[Configuration lists](#) (on page 4-84)

[smu.source.configlist.store\(\)](#) (on page 14-178)

smu.source.configlist.delete()

This function deletes a source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.delete("listName")
smu.source.configlist.delete("listName", index)
```

<i>listName</i>	A string that represents the name of a source configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

smu.source.configlist.delete("mySourceList")	Deletes a configuration list named mySourceList.
smu.source.configlist.delete("mySourceList", 14)	Deletes delete configuration index 14 in the source configuration list named mySourceList

Also see

[Configuration lists](#) (on page 4-84)
[smu.source.configlist.create\(\)](#) (on page 14-173)

smu.source.configlist.query()

This function returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.query("listName", index)
smu.source.configlist.query("listName", index, "fieldSeparator")
```

<i>listName</i>	A string that represents the name of a source configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list; the default is the first index in the configuration list
<i>fieldSeparator</i>	String that represents the separator for the data; use one of the following: <ul style="list-style-type: none"> ▪ Comma (default): , ▪ Semicolon: ; ▪ New line: \n

Details

This command can only return data for one configuration index. To get data for additional configuration indexes, resend the command and specify different configuration indexes.

Refer to [Settings stored in a source configuration list](#) (on page 4-99) for a complete list of source settings that the instrument stores in a source configuration list.

Example

```
print(smu.source.configlist.query("MyScrList", 2))
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Also see

[Configuration lists](#) (on page 4-84)
[smu.source.configlist.create\(\)](#) (on page 14-173)
[Settings stored in a source configuration index](#) (on page 4-99)

smu.source.configlist.recall()

This function recalls a specific configuration index in a specific source configuration list and an optional measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.configlist.recall("listName", index)
smu.source.configlist.recall("listName", index, "measureListName")
smu.source.configlist.recall("listName", index, "measureListName", measureIndex)
```

listName	A string that represents the name of a source configuration list
index	A number that defines a specific configuration index in the source configuration list
measureListName	A string that represents the name of a measure configuration list
measureIndex	A number that defines a specific configuration index in the measure configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a specific source configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified source configuration list of that index.

You can optionally specify a measure configuration list and index to recall with the source settings. If you do not specify a measure index, the measure index defaults to match the source index. Specify a source and measure list together with this command to allow the instrument to coordinate the application of the settings in the two lists appropriately. If you do not need have the application of the source and measure configuration lists coordinated, you can specify just the source configuration list with this command and use the [smu.measure.configlist.recall\(\)](#) (on page 14-132) command to recall measure settings separately in your application.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, event code 2790, "Configuration list, error, does not exist" is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. To see the settings that are recalled with an index, use the [smu.source.configlist.query\(\)](#) (on page 14-175) command.

NOTE

To recall a source configuration list separately (not with this command), recall the source configuration list before the measure configuration list. This order ensures that dependencies between source and measure settings will be properly handled.

Example

<code>smu.source.configlist.recall("MySourceList")</code>	Because an index was not specified, this command recalls configuration index 1 from a configuration list named MySourceList.
<code>smu.source.configlist.recall("MySourceList" , 5)</code>	Recalls configuration index 5 in a configuration list named MySourceList.
<code>smu.source.configlist.recall("MySourceList" , 3 , "MyMeasList")</code>	Recalls index 3 from a source configuration list named MySourceList, then recalls index 3 from a measure configuration list named MyMeasList (because the measure list index was not specified, the instrument automatically recalled the same index that was called in the source list).
<code>smu.source.configlist.recall("MySourceList" , 3 , "MyMeasList" , 5)</code>	Recalls index 3 from a source configuration list named MySourceList, then recalls index 5 from a measure configuration list named MyMeasList.

Also see[Configuration lists](#) (on page 4-84)[smu.source.configlist.create\(\)](#) (on page 14-173)**smu.source.configlist.size()**

This function returns the number of configuration indexes in a source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
indexCount = smu.source.configlist.size("listName")
```

<code>indexCount</code>	A number that represents the total quantity of indexes stored in the specified source configuration list
<code>listName</code>	A string that represents the name of a source configuration list

Details

The size of the list is equal to the number of configuration indexes in a configuration list.

Example

<code>print(smu.source.configlist.size("MyScrList"))</code>	Determine the number of configuration indexes in a source configuration list named MyScrList. Example output: 2
---	---

Also see[Configuration lists](#) (on page 4-84)[smu.source.configlist.create\(\)](#) (on page 14-173)

smu.source.configlist.store()

This function stores the active source settings into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list		

Usage

```
smu.source.configlist.store("listName")
smu.source.configlist.store("listName", index)
```

<i>listName</i>	A string that represents the name of a source configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active source settings to a configuration index in a configuration list. If the index is defined, the configuration list is stored in that index. If the index is not defined, the configuration index is appended to the end of the list. If a configuration index already exists for the specified index, the new configuration overwrites the existing configuration index.

Refer to [Settings stored in a source configuration index](#) (on page 4-99) for information about the settings this command stores.

Example

smu.source.configlist.store("MyConfigList")	Store the active settings of the instrument to the source configuration list MyConfigList. Settings are saved at the end of the list since no index parameter is specified.
smu.source.configlist.store("MyConfigList" , 5)	Store the active settings of the instrument to configuration index 5 on the source configuration list MyConfigList.

Also see

None

smu.source.configlist.storefunc()

This function allows you to store the settings for a source function into a source configuration list whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
smu.source.configlist.storefunc("ConfigListName", function)
smu.source.configlist.storefunc("ConfigListName", function, index)
```

<i>ConfigListName</i>	Name of the configuration list in which to store the function settings
<i>function</i>	The function to save into the configuration list: <ul style="list-style-type: none"> ■ Current source: smu.FUNC_DC_CURRENT ■ Voltage source: smu.FUNC_DC_VOLTAGE
<i>index</i>	The number of the configuration list index in which to store the settings

Details

The configuration list must be created before you send this function.

Example

```
smu.source.configlist.create("sourcelist")
smu.source.configlist.storefunc("sourcelist", smu.FUNC_DC_CURRENT)
Create a configuration list named sourcelist.
Store the settings for the DC current source function into the configuration list in index 1.
```

Also see

[smu.measure.configlist.create\(\)](#) (on page 14-129)

smu.source.delay

This attribute contains the source delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```
sDelay = smu.source.delay
smu.source.delay = sDelay
sDelay = smu.source.getattribute(function, smu.ATTR_SRC_DELAY)
smu.source.setattribute(function, smu.ATTR_SRC_DELAY, sDelay)
```

<i>sDelay</i>	The length of the delay (0 to 10 ks)
<i>function</i>	The source function: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE

Details

This command sets a delay for the selected source function. This delay is in addition to normal settling times.

After the programmed source is turned on, this delay allows the source level to settle before a measurement is made.

If you set a specific source delay (`smu.source.delay`), source autodelay is turned off.

When source autodelay is turned on, the manual source delay setting is overwritten with the autodelay setting.

When either a source delay or autodelay is set, the delay is applied to the first source output and then only when the magnitude of the source changes.

Example

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.delay = 3
```

Set the function to voltage. Set a 3 s delay after the source is turned on before a measurement is made.

Also see

[smu.source.autodelay](#) (on page 14-171)

[Source delay](#) (on page 4-46)

smu.source.func

This attribute contains the source function, which can be voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list	Configuration script Source configuration list	smu.FUNC_DC_VOLTAGE

Usage

```
sFunction = smu.source.func
smu.source.func = sFunction
```

<i>sFunction</i>	The source function; set to one of the following values: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE
------------------	---

Details

When you set this command, it configures the instrument as either a voltage source or a current source.

When you read this command, it returns the output setting of the source.

Example

```
smu.source.func = smu.FUNC_DC_CURRENT
```

Sets the source function of the instrument to be a current source.

Also see

[smu.source.level](#) (on page 14-183)
[smu.source.output](#) (on page 14-186)

smu.source.getattribute()

This function returns the setting for a function attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = smu.source.getattribute(function, setting)
```

<i>value</i>	The attribute value
<i>function</i>	The source function: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE
<i>setting</i>	The setting of the attribute; refer to smu.source.setattribute() (on page 14-191) for the list of attributes

Details

You can retrieve one attribute at a time.

Example

```
print(smu.source.getattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_DELAY))
```

Retrieve the source delay setting for DC current.
 Example return:
 1.05e-07

Also see

[smu.source.setattribute\(\)](#) (on page 14-191)

smu.source.highc

This attribute enables or disables high-capacitance mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFF

Usage

```
state = smu.source.highc
smu.source.highc = state
state = smu.source.getattribute(function, smu.ATTR_SRC_HIGHC)
smu.source.setattribute(function, smu.ATTR_SRC_HIGHC, state)
```

state	Turn high-capacitance mode off: smu.OFF Turn high-capacitance mode on: smu.ON
function	The source function: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE

Details

When the instrument is measuring low current and is driving a capacitive load, you may see overshoot, ringing, and instability. You can enable the high capacitance mode to minimize these problems.

Example

smu.source.highc = smu.ON	Turn the high capacitance mode on.
---------------------------	------------------------------------

Also see

[High-capacitance operation](#) (on page 5-22)

smu.source.level

This attribute immediately selects a fixed amplitude for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```
sourceLevel = smu.source.level
smu.source.level = sourceLevel
sourceLevel = smu.source.getattribute(function, smu.ATTR_SRC_LEVEL)
smu.source.setattribute(function, smu.ATTR_SRC_LEVEL, sourceLevel)
```

sourceLevel	Current: -1.05 A to 1.05 A Voltage: -210 V to 210 V
function	The source function: <ul style="list-style-type: none"> ■ Current source: smu.FUNC_DC_CURRENT ■ Voltage source: smu.FUNC_DC_VOLTAGE

Details

This command sets the output level of the voltage or current source. If the output is on, the new level is sourced immediately.

The sign of the source level dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

If a manual source range is selected, the level cannot exceed the specified range. For example, if the voltage source is on the 2 V range, you cannot set the voltage source level to 3 V. When autorange is selected, the amplitude can be set to any level supported by the instrument.

Example

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.level = 1
```

Set the instrument to source voltage and set it to source 1 V.

Also see

[smu.source.func](#) (on page 14-181)
[smu.source.output](#) (on page 14-186)
[smu.source.protect.level](#) (on page 14-187)

smu.source.offmode

This attribute defines the state of the source when the output is turned off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFFMODE_NORMAL

Usage

```
sourceOffMode = smu.source.offmode
smu.source.offmode = sourceOffMode
sourceOffMode = smu.source.getattribute(function, smu.ATTR_SRC_OFFMODE)
smu.source.setattribute(function, smu.ATTR_SRC_OFFMODE, sourceOffMode)
```

<i>sourceOffMode</i>	The output-off setting; set to one of the following values (see the Details below for specifics regarding each option): <ul style="list-style-type: none"> ■ smu.OFFMODE_NORMAL ■ smu.OFFMODE_ZERO ■ smu.OFFMODE_HIGHZ ■ smu.OFFMODE_GUARD
<i>function</i>	The source function: <ul style="list-style-type: none"> ■ Current source: smu.FUNC_DC_CURRENT ■ Voltage source: smu.FUNC_DC_VOLTAGE

Details

When the 2450 is set to the normal output-off state, the following settings are made when the source is turned off:

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

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

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

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

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

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

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

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

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

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

Note that the front-panel display does not reflect all of the changes. For example, the 4-wire display indicator continues to display when the output is off, even though the sense is changed to 2-wire.

Example

```
smu.source.offmode = smu.OFFMODE_HIGHZ
```

Sets the output-off state so that the instrument opens the output relay when the output is turned off.

Also see

[Output-off state \(on page 4-16\)](#)
[smu.source.output \(on page 14-186\)](#)

smu.source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFF

Usage

```
sourceOutput = smu.source.output
smu.source.output = sourceOutput
```

sourceOutput	Switch the source output off: smu.OFF Switch the source output on: smu.ON
--------------	--

Details

When the output is switched on, the instrument sources either voltage or current, as set by smu.source.func.

Example

smu.source.output = smu.ON	Switch the source output of the instrument to on.
----------------------------	---

Also see

[Turn the 2450 output on or off](#) (on page 3-4)
[smu.source.func](#) (on page 14-181)
[smu.source.offmode](#) (on page 14-184)

smu.source.protect.level

This attribute sets the overvoltage protection setting of the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.PROTECT_NONE

Usage

```
smu.PROTECT_x = smu.source.protect.level
smu.source.protect.level = smu.PROTECT_x
smu.PROTECT_x = smu.source.getattribute(function, smu.ATTR_SRC_PROTECT_LEVEL)
smu.source.setattribute(function, smu.ATTR_SRC_PROTECT_LEVEL, smu.PROTECT_x)
```

x	The overvoltage protection level: 2V, 5V, 10V, 20V, 40V, 60V, 80V, 100V, 120V, 140V, 160V, 180V, or NONE
sourceOffMode	The output-off setting; set to one of the following values (see the Details below for specifics regarding each option): <ul style="list-style-type: none"> ■ smu.OFFMODE_NORMAL ■ smu.OFFMODE_ZERO ■ smu.OFFMODE_HIGHZ ■ smu.OFFMODE_GUARD
function	The source function: <ul style="list-style-type: none"> ■ Current source: smu.FUNC_DC_CURRENT ■ Voltage source: smu.FUNC_DC_VOLTAGE

Details

Overtoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced.

This protection is in effect for both positive and negative output voltages.

When this attribute is used in a test sequence, it should be set before turning the source on.

WARNING

Even with the overvoltage protection set to the lowest value (2 V), never touch anything connected to the terminals of the 2450 when the instrument is on. Always assume that a hazardous voltage (greater than 30 V_{RMS}) is present when the instrument is on. To prevent damage to the device under test or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

Example

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.protect.level = smu.PROTECT_40V
```

Sets the maximum voltage limit of the instrument to 40 V.

Also see

[Overvoltage protection](#) (on page 4-34)

smu.source.protect.tripped

This attribute indicates if the overvoltage source protection feature is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = smu.source.protect.tripped
value = smu.source.getattribute(function, smu.ATTR_SRC_PROTECT_TRIPPED)
```

value	Overvoltage protection not active: smu.OFF Overvoltage protection active: smu.ON
function	The source function: <ul style="list-style-type: none">■ Current source: smu.FUNC_DC_CURRENT■ Voltage source: smu.FUNC_DC_VOLTAGE

Details

When overvoltage protection is active, the instrument restricts the maximum voltage level that the instrument can source.

Example

print(smu.source.protect.tripped)	If overvoltage protection is active, the output is: smu.ON
-----------------------------------	---

Also see

[Overvoltage protection](#) (on page 4-34)
[smu.source.protect.level](#) (on page 14-187)

smu.source.range

This attribute selects the range for the source for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	Current: 1e-8 A Voltage: 0.02 V

Usage

```
rangeValue = smu.source.range
smu.source.range = rangeValue
rangeValue = smu.source.getattribute(function, smu.ATTR_SRC_RANGE)
smu.source.setattribute(function, smu.ATTR_SRC_RANGE, rangeValue)
```

rangeValue	Set to the maximum expected voltage or current to be sourced; see Details for values; the ranges are: <ul style="list-style-type: none">■ Current: -1 A to 1 A■ Voltage: -200 V to 200 V
------------	--

<i>function</i>	The source function: <ul style="list-style-type: none">■ Current source: <code>smu.FUNC_DC_CURRENT</code>■ Voltage source: <code>smu.FUNC_DC_VOLTAGE</code>
-----------------	--

Details

This command manually selects the measurement range for the specified source.

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 μ A, 10 μ A, 100 μ A, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 20 mV, 200 mV, 2 V, 20 V, and 200 V.

When you read this value, the instrument returns the positive full-scale value that the instrument is presently using.

This command is intended to eliminate the time required by the automatic range selection.

To select the range, you can specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 50 mV, send 0.05 (or 50e-3) to select the 200 mV range.

NOTE

If automatic range selection is set to on, when you select a specific range, automatic is set to off. To set the range to automatic selection, use the source autorange command.

Example

<code>smu.source.func = smu.FUNC_DC_CURRENT</code> <code>smu.source.autorange = smu.OFF</code> <code>smu.source.range = 1</code>	Set the instrument to source current. Turn autorange off. Set the source range to 1 A.
--	--

Also see

[Ranges](#) (on page 4-38)

[smu.source.autorange](#) (on page 14-170)

smu.source.readback

This attribute determines if the instrument records the measured source value or the configured source value when making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
state = smu.source.readback
smu.source.readback = state
state = smu.source.getattribute(function, smu.ATTR_SRC_READBACK)
smu.source.setattribute(function, smu.ATTR_SRC_READBACK, state)
```

state	Disable readback: smu.OFF Enable readback: smu.ON
function	The source function: <ul style="list-style-type: none">▪ Current source: smu.FUNC_DC_CURRENT▪ Voltage source: smu.FUNC_DC_VOLTAGE

Details

When source readback is off, the instrument records and displays the source value you set. When you use the actual source value (source readback on), the instrument measures the actual source value immediately before making the device under test measurement.

Using source readback results in more accurate measurements, but also a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the device-under-test measurement. When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the device-under-test measurement.

Example

reset() testDataBuffer = buffer.make(100) smu.source.func = smu.FUNC_DC_VOLTAGE smu.measure.func = smu.FUNC_DC_CURRENT smu.source.readback = smu.ON smu.source.level = 10 smu.measure.count = 100 smu.source.output = smu.ON smu.measure.read(testDataBuffer) smu.source.output = smu.OFF printbuffer(1, 100, testDataBuffer.sourcevalues, testDataBuffer)	Reset the instrument to default settings. Make a buffer named testDataBuffer that can hold 100 readings. Set source function to voltage. Set the measurement function to current. Set read back on. Set the instrument to take 100 readings. Turn the output on. Take the measurements. Turn the output off. Get the source values and measurements from the buffer.
---	---

Also see

[smu.measure.func](#) (on page 14-143)
[smu.source.func](#) (on page 14-181)

smu.source.setattribute()

This function allows you to set up a source function whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Source configuration list	Not applicable

Usage

```
smu.source.setattribute(function, setting, value)
```

<i>function</i>	The source function; set to one of the following values: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE
<i>setting</i>	The parameter to be set; see Details
<i>value</i>	The function or setting value

Details

You can set the following parameters. The links in the list below link to the descriptions of the corresponding TSP command descriptions. The settings available for each parameter are the same as the settings for the TSP commands.

The parameters you can set are:

- [Autorange](#) (on page 14-170): smu.ATTR_SRC_RANGE_AUTO
- [Autodelay](#) (on page 14-171): smu.ATTR_SRC_DELAY_AUTO
- [Delay](#) (on page 14-180): smu.ATTR_SRC_DELAY
- [Function](#) (on page 14-181): smu.ATTR_SRC_FUNCTION
- [High capacitance](#) (on page 14-182): smu.ATTR_SRC_HIGHC
- [Level](#) (on page 14-183): smu.ATTR_SRC_LEVEL
- [Limit level](#) (on page 14-201): smu.ATTR_SRC_LIMIT_LEVEL
- [Output Off](#) (on page 14-184): smu.ATTR_SRC_OFFMODE
- [Overvoltage protection limit](#) (on page 14-187): smu.ATTR_SRC_PROTECT_LEVEL
- [Overvoltage protection limit active](#) (on page 14-188): smu.ATTR_SRC_PROTECT_TRIPPED
- [Range](#) (on page 14-188): smu.ATTR_SRC_RANGE
- [Readback](#) (on page 14-190): smu.ATTR_SRC_READBACK
- [Source limit exceeded](#) (on page 14-202): smu.ATTR_SRC_LIMIT_TRIPPED
- [User delay N](#) (on page 14-200): smu.ATTR_SRC_USER_DELAY_N, where *N* is the user delay, 1 to 5

Example

```
smu.source.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_RANGE, 0.007)
smu.source.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_DELAY, 0.35)
smu.source.setattribute(smu.FUNC_DC_CURRENT, smu.ATTR_SRC_LEVEL, 0.035)
Sets the range, delay, and level settings for DC current, whether or not DC current is the active function.
```

Also see

[smu.source.configlist.storefunc\(\)](#) (on page 14-179)
[smu.source.getattribute\(\)](#) (on page 14-181)

smu.source.sweeplinear()

This function sets up a linear sweep for a fixed number of measurement points.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplinear("configListName", start, stop, points)
smu.source.sweeplinear("configListName", start, stop, points, sDelay)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
    rangeType)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
    rangeType, failAbort)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
    rangeType, failAbort, dual)
smu.source.sweeplinear("configListName", start, stop, points, sDelay, count,
    rangeType, failAbort, dual, bufferName)
```

<i>configListName</i>	A string that contains the name of the configuration list that the instrument will create for this sweep
<i>start</i>	The voltage or current source level at which the sweep starts: <ul style="list-style-type: none"> ▪ Current: -1.05 A to 1.05 A ▪ Voltage: -210 V to 210 V
<i>stop</i>	The voltage or current at which the sweep stops: <ul style="list-style-type: none"> ▪ Current: -1.05 A to 1.05 A ▪ Voltage: -210 V to 210 V
<i>points</i>	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: $\text{Points} = [(\text{Stop} - \text{Start}) / \text{Step}] + 1$
<i>sDelay</i>	The delay between measurement points; default is <code>smu.DELAY_AUTO</code> , which enables autodelay, or a specific delay value from 50 µs to 10 ks, or 0 for no delay
<i>count</i>	The number of times to run the sweep; default is 1: <ul style="list-style-type: none"> ▪ Infinite loop: <code>smu.INFINITE</code> ▪ Finite loop: 1 to 268,435,455

<i>rangeType</i>	The source range that is used for the sweep: <ul style="list-style-type: none"> ■ Most sensitive source range for each source level in the sweep: <code>smu.RANGE_AUTO</code> ■ Best fixed range: <code>smu.RANGE_BEST</code> (default) ■ Present source range for the entire sweep: <code>smu.RANGE_FIXED</code>
<i>failAbort</i>	Complete the sweep if the source limit is exceeded: <code>smu.OFF</code> Abort the sweep if the source limit is exceeded: <code>smu.ON</code> (default)
<i>dual</i>	Determines if the sweep runs from start to stop and then from stop to start: <ul style="list-style-type: none"> ■ Sweep from start to stop only: <code>smu.OFF</code> (default) ■ Sweep from start to stop, then stop to start: <code>smu.ON</code>
<i>bufferName</i>	The name of a reading buffer; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it clears any existing trigger models, creates a source configuration list, and populates the trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the sweep delay, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```

reset()
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.source.sweeplinear("VoltLinSweep", 0, 10, 20, 1e-3, 1, smu.RANGE_FIXED)
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 100e-6
trigger.model.initiate()

Reset the instrument to its defaults.
Set the source function to voltage.
Set the source range to 20 V.
Set up a linear sweep that sweeps from 0 to 10 V in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep VoltLinSweep.
Set the measure function to current.
Set the current range to 100 μA.
Start the sweep.

```

Also see

[Sweep operation](#) (on page 4-54)
[trigger.model.initiate\(\)](#) (on page 14-241)

smu.source.sweeplinearstep()

This function sets up a linear source sweep configuration list and trigger model with a fixed number of steps.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```

smu.source.sweeplinearstep("configListName", start, stop, step)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
    rangeType)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
    rangeType, failAbort)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
    rangeType, failAbort, dual)
smu.source.sweeplinearstep("configListName", start, stop, step, sDelay, count,
    rangeType, failAbort, dual, bufferName)

```

<i>configListName</i>	A string that contains the name of the configuration list that the instrument will create for this sweep
<i>start</i>	The voltage or current source level at which the sweep starts: <ul style="list-style-type: none"> ▪ Current: -1.05 A to 1.05 A ▪ Voltage: -210 V to 210 V
<i>stop</i>	The voltage or current at which the sweep stops: <ul style="list-style-type: none"> ▪ Current: -1.05 A to 1.05 A ▪ Voltage: -210 V to 210 V
<i>step</i>	The step size at which the source level will change; must be more than 0

<i>sDelay</i>	The delay between measurement points; default is <code>smu.DELAY_AUTO</code> , which enables autodelay, a specific delay value from 50 μ s to 10 ks, or 0 for no delay
<i>count</i>	The number of times to run the sweep; default is 1: <ul style="list-style-type: none"> ■ Infinite loop: <code>smu.INFINITE</code> ■ Finite loop: 1 to 268,435,455
<i>rangeType</i>	The source range that is used for the sweep: <ul style="list-style-type: none"> ■ Most sensitive source range for each source level in the sweep: <code>smu.RANGE_AUTO</code> ■ Best fixed range: <code>smu.RANGE_BEST</code> (default) ■ Present source range for the entire sweep: <code>smu.RANGE_FIXED</code>
<i>failAbort</i>	Complete the sweep if the source limit is exceeded: <code>smu.OFF</code> Abort the sweep if the source limit is exceeded: <code>smu.ON</code> (default)
<i>dual</i>	Determines if the sweep runs from start to stop and then from stop to start: <ul style="list-style-type: none"> ■ Sweep from start to stop only: <code>smu.OFF</code> (default) ■ Sweep from start to stop, then stop to start: <code>smu.ON</code>
<i>bufferName</i>	The name of a reading buffer; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>

Details

When the sweep is started, the instrument sources a specific voltage or current voltage to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it deletes the existing trigger model and creates a trigger model with a uniform series of ascending or descending voltage or current changes, called steps. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the stop level, which is calculated from the number of steps. A measurement is performed at each source step (including the start and stop levels). At this level, the instrument performs another measurement and then stops the sweep.

The instrument uses the step size parameter to determine the number of source level changes. The source level changes in equal steps from the start level to the stop level. To avoid a setting conflicts error, make sure the step size is greater than the start value and less than the stop value. To calculate the number of source-measure points in a sweep, use the following formula:

$$\text{Points} = [(Stop - Start) / Step] + 1$$

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```
reset()
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 1
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
smu.source.sweeplinearstep("CurrLogSweep", -1.05, 1.05, .25, 10e-3, 1,
                           smu.RANGE_FIXED)
trigger.model.initiate()

Reset the instrument to its defaults.
Set the source function to current.
Set the source range to 1 A. Set the measure function to voltage with a range of 20 V.
Set up a linear step sweep that sweeps from -1.05 A to 1.05 A in 0.25 A increments with a source delay of
1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep
CurrLogSweep.
Start the sweep.
```

Also see

[Sweep operation](#) (on page 4-54)

smu.source.sweeplist()

This function sets up a sweep based on a configuration list, which allows you to customize the sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplist("configListName")
smu.source.sweeplist("configListName", index)
smu.source.sweeplist("configListName", index, sDelay)
smu.source.sweeplist("configListName", index, sDelay, count)
smu.source.sweeplist("configListName", index, sDelay, count, failAbort)
smu.source.sweeplist("configListName", index, sDelay, count, failAbort, bufferName)
```

<i>configListName</i>	The name of the source configuration list that the sweep uses; this must be defined before sending this command
<i>index</i>	The index in the configuration list where the sweep starts; default is 1
<i>sDelay</i>	The delay between measurement points; default is 0 for no delay or you can set a specific delay value from 50 µs to 10 ks
<i>count</i>	The number of times to run the sweep; default is 1: <ul style="list-style-type: none"> ■ Infinite loop: <code>smu.INFINITE</code> ■ Finite loop: 1 to 268,435,455
<i>failAbort</i>	Complete the sweep if the source limit is exceeded: <code>smu.OFF</code> Abort the sweep if the source limit is exceeded: <code>smu.ON</code> (default)
<i>bufferName</i>	The name of a reading buffer; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>

Details

This command allows you to set up a custom sweep, using a configuration list to specify the source levels.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

To run the sweep, initiate the trigger model.

You cannot use a writable reading buffer to collect data from a sweep.

Example

```
reset()
smu.source.configlist.create("CurrListSweep")
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 100e-3
smu.source.level = 1e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 10e-6
smu.source.configlist.store("CurrListSweep")
smu.source.level = 7e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 11e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 9e-3
smu.source.configlist.store("CurrListSweep")
smu.source.sweeplist("CurrListSweep", 1, 0.001)
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
trigger.model.initiate()
```

Reset the instrument to its defaults

Create a source configuration list called CurrListSweep.

Set the source function to current.

Set the source current range to 100 mA.

Set the source current level to 1 mA.

Save the source settings to CurrListSweep.

Set the source current level to 10 µA.

Save the source settings to CurrListSweep.

Set the source current level to 7 mA.

Save the source settings to CurrListSweep.

Set the source current level to 11 mA.

Save the source settings to CurrListSweep.

Set the source current level to 9 mA.

Save the source settings to CurrListSweep.

Set up a list sweep that uses the entries from the CurrListSweep configuration list and starts at index 1 of the list.

Set a source delay of 1 ms.

Start the sweep.

Also see

[Configuration lists](#) (on page 4-84)

[Sweep operation](#) (on page 4-54)

[trigger.model.initiate\(\)](#) (on page 14-241)

smu.source.sweeplog()

This function sets up a logarithmic sweep for a set number of measurement points.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smu.source.sweeplog("configListName", start, stop, points)
smu.source.sweeplog("configListName", start, stop, points, sDelay)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort, dual)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort, dual, bufferName)
smu.source.sweeplog("configListName", start, stop, points, sDelay, count, rangeType,
    failAbort, dual, bufferName, asymptote)
```

<i>configListName</i>	A string that contains the name of the configuration list that the instrument will create for this sweep
<i>start</i>	The voltage or current source level at which the sweep starts: <ul style="list-style-type: none"> ▪ Current: 1 pA to 1.05 A ▪ Voltage: 1 pV to 210 V
<i>stop</i>	The voltage or current at which the sweep stops: <ul style="list-style-type: none"> ▪ Current: 1 pA to 1.05 A ▪ Voltage: 1 pV to 210 V
<i>points</i>	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
<i>sDelay</i>	The delay between measurement points; default is <code>smu.DELAY_AUTO</code> , which enables autodelay, or a specific delay value from 50 µs to 10 ks, or 0 for no delay
<i>count</i>	The number of times to run the sweep; default is 1: <ul style="list-style-type: none"> ▪ Infinite loop: <code>smu.INFINITE</code> ▪ Finite loop: 1 to 268,435,455
<i>rangeType</i>	The source range that is used for the sweep: <ul style="list-style-type: none"> ▪ Most sensitive source range for each source level in the sweep: <code>smu.RANGE_AUTO</code> ▪ Best fixed range: <code>smu.RANGE_BEST</code> (default) ▪ Present source range for the entire sweep: <code>smu.RANGE_FIXED</code>
<i>failAbort</i>	Complete the sweep if the source limit is exceeded: <code>smu.OFF</code> Abort the sweep if the source limit is exceeded: <code>smu.ON</code> (default)
<i>dual</i>	Determines if the sweep runs from start to stop and then from stop to start: <ul style="list-style-type: none"> ▪ Sweep from start to stop only: <code>smu.OFF</code> (default) ▪ Sweep from start to stop, then stop to start: <code>smu.ON</code>

<i>bufferName</i>	The name of a reading buffer; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>
<i>asymptote</i>	Default is 0; see Details

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it clears the existing trigger model and creates a new trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

You cannot use a writable reading buffer to collect data from a sweep.

The asymptote changes the inflection of the sweep curve and allows it to sweep through zero. You can use the asymptote parameter to customize the inflection and offset of the source value curve. Setting this parameter to zero provides a conventional logarithmic sweep. The asymptote value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep. The asymptote value must not be equal to or between the starting and ending values. It must be outside the range defined by the starting and ending values.

Example

```

reset()
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 100e-6
smu.source.sweeplog("VoltLogSweep", 1, 10, 20, 1e-3, 1, smu.RANGE_FIXED)
trigger.model.initiate()

Reset the instrument to its defaults.
Set the source function to voltage.
Set the source range to 20 V.
Set the measure function to current.
Set the current range to 100 µA.
Set up a log sweep that sweeps from 1 to 10 V in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep VoltLogSweep.
Start the sweep.

```

Also see

[Sweep operation](#) (on page 4-54)
[trigger.model.initiate\(\)](#) (on page 14-241)

smu.source.userdelay[N]

This attribute sets a user-defined delay that you can use in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```

delayTime = smu.source.userdelay[N]
smu.source.userdelay[N] = delayTime
delayTime = smu.source.getattribute(function, smu.ATTR_SRC_USER_DELAY_N)
smu.source.setattribute(function, smu.ATTR_SRC_USER_DELAY_N, delayTime)

```

delayTime	The delay in seconds (0 to 10 ks)
N	The number that identifies this user delay (1 to 5)
function	The source function: <ul style="list-style-type: none"> ▪ Current source: smu.FUNC_DC_CURRENT ▪ Voltage source: smu.FUNC_DC_VOLTAGE

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.

The delay is specific to the selected function.

Example

```
smu.source.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_S1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay for source 1 to 5 s.
 Set trigger block 1 to turn the source output on.
 Set trigger block 2 to a dynamic delay that calls source user delay 1.
 Set trigger block 3 to make a measurement.
 Set trigger block 4 to turn the source output off.
 Set trigger block 5 to branch to block 1 ten times.
 Start the trigger model.

Also see

[trigger.model.setblock\(\) — trigger.BLOCK_DELAY_DYNAMIC](#) (on page 14-267)

smu.source.xlimit.level

This attribute selects the source limit for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	Voltage: 1.05E-4 A Current: 21 V

Usage

```
value = smu.source.xlimit.level
smu.source.xlimit.level = value
value = smu.source.getattribute(function, smu.ATTR_SRC_LIMIT_LEVEL)
smu.source.setattribute(function, smu.ATTR_SRC_LIMIT_LEVEL, value)
```

<i>value</i>	The limit: <ul style="list-style-type: none"> ■ Current source function: 1 nA to 1.05 A ■ Voltage source function: 0.02 V to 210 V
<i>x</i>	The function for which to set the limit: <ul style="list-style-type: none"> ■ Voltage: <i>v</i> ■ Current: <i>i</i>
<i>function</i>	The source function: <ul style="list-style-type: none"> ■ Current source: smu.FUNC_DC_CURRENT ■ Voltage source: smu.FUNC_DC_VOLTAGE

Details

This command sets the source limit for measurements. The values that can be set for this command are limited by the setting for the overvoltage protection limit.

The 2450 cannot source levels that exceed this limit.

If you change the measure range to a range that is not appropriate for this limit, the instrument changes the source limit to a limit that is appropriate to the range and a warning is generated. Depending on the source range, your actual maximum limit value could be lower. The instrument makes adjustments to stay in the operating boundaries.

This value can also be limited by the measurement range. If a specific measurement range is set, the limit must be 10.6% or higher of the measurement range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

Limits are absolute values.

You can use `smu.source.xlimit.tripped` to check the limit state of the source output.

Example

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 1
```

Set the source function to voltage with the current limit set to 1 A.

Also see

[smu.source.protect.level](#) (on page 14-187)
[smu.source.xlimit.tripped](#) (on page 14-202)

smu.source.xlimit.tripped

This attribute indicates if the source exceeded the limits that were set for the selected measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Function change	Not saved	Not applicable

Usage

```
state = smu.source.xlimit.tripped
state = smu.source.getattribute(function, smu.ATTR_SRC_LIMIT_TRIPPED)
```

<code>state</code>	Indicates if the limit has been tripped: <ul style="list-style-type: none"> ▪ Not tripped = 0 ▪ Tripped = 1
<code>x</code>	The function whose limit was tripped: <ul style="list-style-type: none"> ▪ v: voltage ▪ i: current
<code>function</code>	The source function: <ul style="list-style-type: none"> ▪ Current source: <code>smu.FUNC_DC_CURRENT</code> ▪ Voltage source: <code>smu.FUNC_DC_VOLTAGE</code>

Details

You can use this command to check the limit state of the source.

If the limits were exceeded, the instrument clamps the source to keep the source within the set limits.

If you check the limit for the source that is not presently selected, `nil` is returned.

Example

```
print(smu.source.vlimit.tripped)
```

Check the state of the source limit for voltage. If the limit was exceeded, the output is:
smu.ON

Also see

[smu.source.xlimit.level](#) (on page 14-201)

smu.terminals

This attribute describes which set of input and output terminals the instrument is using.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	smu.TERMINALS_FRONT

Usage

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

terminals	Use the front-panel input and output terminals: smu.TERMINALS_FRONT Use the rear-panel input and output terminals: smu.TERMINALS_REAR
-----------	--

Details

This command selects which set of input and output terminals the instrument uses. You can select front panel or rear panel terminals.

If the output is on when you change from one set of terminals to the other, the output is turned off.

This command replaces the `smu.measure.terminals` command, which is deprecated.

Example

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

Use the front-panel terminals for measurements.

Also see

None

status.clear()

This function clears event registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.clear( )
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It does not affect the Questionable Event Enable or Operation Event Enable registers.

Example

```
status.clear( )
```

Clear the bits in the registers

Also see

[*CLS \(on page 15-2\)](#)

status.condition

This attribute stores the status byte condition register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
statusByte = status.condition
```

statusByte	The status byte
------------	-----------------

Details

You can use this command to read the status byte, which is returned as a numeric value.

When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence. The returned value can indicate that one or more status events occurred. If more than one bit of the register is set, *statusByte* equals the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128). See [Understanding bit settings \(on page 16-15\)](#) for additional information about reading bit values.

NOTE

If you are using the GPIB, USB, or VXI-11 serial poll sequence of the 2450 to get the status byte (also called a serial poll byte), B6 is the Request for Service (RQS) bit. If the bit is set, it indicates that a serial poll (SRQ) has occurred. For additional detail, see [Serial polling and SRQ \(on page 16-13\)](#).

The meanings of the individual bits of this register are shown in the following table.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.MSB	An enabled measurement event
1	2	Not used	
2	4	status.EAV	An error or status message is present in the Error Queue
3	8	status.QSB	An enabled questionable event
4	16	status.MAV	A response message is present in the Output Queue
5	32	status.ESB	An enabled standard event
6	64	status.MSS	An enabled summary bit of the status byte register is set
7	128	status.OSB	An enabled operation event

Example

```
statusByte = status.condition
print(statusByte)
```

Returns `statusByte`.

Example output:

`1.29000e+02`

Converting this output (129) to its binary equivalent yields 1000 0001

Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSS) and B7 (OSB).

Also see

None

status.operation.condition

This attribute reads the Operation Event Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
operationRegister = status.operation.condition
```

`operationRegister`

The status of the operation status register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-15).

Example

```
print(status.operation.condition)
```

Returns the contents of the register.

Also see

[Operation Event Register](#) (on page 16-7)

status.operation.enable

This attribute sets or reads the contents of the Operation Event Enable Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
operationRegister = status.operation.enable
status.operation.enable = operationRegister
```

operationRegister	The status of the operation status register
-------------------	---

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

Example

```
-- decimal 20480 = binary 0101 0000 0000 0000
operationRegister = 20480
status.operation.enable = operationRegister
```

Sets the 12 and 14 bits of the operation status enable register using a decimal value.

Also see

[Operation Event Register \(on page 16-7\)](#)
[Understanding bit settings \(on page 16-15\)](#)

status.operation.event

This attribute reads the Operation Event Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
operationRegister = status.operation.event
```

operationRegister	The status of the operation status register
-------------------	---

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example

```

status.request_enable = status.OSB
status.operation.setmap(0, 4918, 4917)
status.operation.enable = 1
defbuffer1.clear()
defbuffer1.fillmode = buffer.FILL_ONCE
defbuffer1.capacity = 10
smu.measure.count = 10
smu.measure.read()
print(status.operation.event)

```

Set bits in the Status Request Enable Register to record an enabled event in the Operation Status Register.

Map event number 4918 (a default buffer is full) to set bit 0 in the Operation Event Register and event number 4917 (a default buffer is empty) to clear bit 0.

Clear defbuffer1.

Set defbuffer1 to fill once.

Resizes defbuffer1 to 10 readings.

Sets the measure count to 10 readings and makes a measurement.

Reads the operation event register.

Output:

1

Also see

[Operation Event Register](#) (on page 16-7)

status.operation.getmap()

This function requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setEvent, clearEvent = status.operation.getmap(bitNumber)
```

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

```
print(status.operation.getmap(0))
```

Query bit 0 of the Operation Event Register.

Example output:

4918 4917

Also see

[Operation Event Register](#) (on page 16-7)
[Programmable status register sets](#) (on page 16-4)
[status.operation.setmap\(\)](#) (on page 14-208)

status.operation.setmap()

This function allows you to map events to bits in the Operation Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.operation.setmap(bitNumber, setEvent)
status.operation.setmap(bitNumber, setEvent, clearEvent)
```

bitNumber	The bit number that is mapped to an event (0 to 14)
setEvent	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
clearEvent	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

See [Event numbers](#) (on page 16-9) for information about event numbers.

Example

```
status.operation.setmap(0, 2731, 2732)
```

When event 2731 (trigger model initiated) occurs, bit 0 in the condition and event registers of the Operation Event Register are set. When event 2732 (trigger model idled) occurs, bit 0 in the condition register is cleared.

Also see

[Event numbers](#) (on page 16-9)
[Operation Event Register](#) (on page 16-7)
[Programmable status register sets](#) (on page 16-4)
[status.operation.getmap\(\)](#) (on page 14-207)

status.preset()

This function resets all bits in the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.preset()
```

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the Service Request Enable Register (*SRE) to Standard Request Enable Register (*ESE).

Preset does not affect the event queue.

The Standard Event Status Register is not affected by this command.

Example

```
status.preset()
```

Resets the instrument status model.

Also see

[Status model](#) (on page 16-1)

status.questionable.condition

This attribute reads the Questionable Condition Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
questionableRegister = status.questionable.condition
```

questionableRegister	The value of the register (0 to 65535)
----------------------	--

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-15).

Example

```
print(status.questionable.condition)
```

Reads the Questionable Condition Register.

Also see

[Questionable Event Register](#) (on page 16-6)

[Understanding bit settings](#) (on page 16-15)

status.questionable.enable

This attribute sets or reads the contents of the questionable event enable register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
questionableRegister = status.questionable.enable
status.questionable.enable = questionableRegister
```

questionableRegister	The value of the register (0 to 65535)
----------------------	--

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register are set.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-15).

Example

```
status.questionable.enable = 17
print(status.questionable.enable)
```

Set bits 0 and 4 of the Questionable Event Enable Register. Returns 17, which indicates the register was set correctly.
--

Also see

[Questionable Event Register](#) (on page 16-6)

status.questionable.event

This attribute reads the Questionable Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
questionableRegister = status.questionable.event
```

questionableRegister	The value of the questionable status register (0 to 65535)
----------------------	--

Details

This query reads the contents of the questionable status event register. After sending this command and addressing the instrument to talk, a value is sent to the computer. This value indicates which bits in the appropriate register are set.

The Questionable Register can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set the Questionable Register to the sum of their decimal weights. For example, to set bits B12 and B13, set the Questionable Register to 12,288 (which is the sum of 4,096 + 8,192).

Example 1

```
-- decimal 66 = binary 0100 0010
questionableRegister = 66
status.questionable.enable = questionableRegister
Uses a decimal value to set bits B1 and B6 of the status questionable enable register.
```

Example 2

```
-- decimal 2560 = binary 00001010 0000 0000
questionableRegister = 2560
status.questionable.enable = questionableRegister
Uses a decimal value to set bits B9 and B11 of the status questionable enable register.
```

Also see

[Questionable Event Register](#) (on page 16-6)

status.questionable.getmap()

This function requests the mapped set event and mapped clear event status for a bit in the Questionable Event Registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setEvent, clearEvent = status.questionable.getmap(bitNumber)
```

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check (0 to 14)

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

```
print(status.questionable.getmap(9))
```

Returns the events that were mapped to set and clear bit 9.

Also see

[Questionable Event Register](#) (on page 16-6)

[status.questionable.setmap\(\)](#) (on page 14-212)

status.questionable.setmap()

This function maps events to bits in the questionable event registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.questionable.setmap(bitNumber, setEvent)
status.questionable.setmap(bitNumber, setEvent, clearEvent)
```

<i>bitNumber</i>	The bit number that is mapped to an event (0 to 14)
<i>setEvent</i>	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
<i>clearEvent</i>	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

See [Event numbers](#) (on page 16-9) for information about event numbers.

Example

```
status.questionable.setmap(0, 4917, 4918)
```

When event 4917 (a default buffer is 0% filled) occurs, bit 0 is set in the condition register and the event register of the Questionable Event Register. When event 4918 (a default buffer is 100% filled) occurs, bit 0 in the condition register is cleared.

Also see

[Event numbers](#) (on page 16-9)

[status.questionable.getmap\(\)](#) (on page 14-211)

status.request_enable

This attribute stores the settings of the Service Request (SRQ) Enable Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
SRQEnableRegister = status.request_enable
status.request_enable = SRQEnableRegister
```

SRQEnableRegister	The status of the service request (SRQ) enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings (0 to 255)
-------------------	--

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	status.MSB	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	status.EAV	An error or status message is present in the Error Queue.
3	8	status.QSB	An enabled event in the Questionable Status Register has occurred.
4	16	status.MAV	A response message is present in the Output Queue.
5	32	status.ESB	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	status.OSB	An enabled event in the Operation Status Register has occurred.

Example 1

```
requestSRQEnableRegister = status.MSB + status.OSB
status.request_enable = requestSRQEnableRegister
```

Uses constants to set the MSB and OSB bits of the service request (SRQ) enable register and clear all other bits.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the MSB and OSB bits and clear all other bits of the service request (SRQ) enable register.

Example 3

status.request_enable = 0

Clear the register.

Also see

[Status model](#) (on page 16-1)
[Understanding bit settings](#) (on page 16-15)

status.standard.enable

This attribute reads or sets the bits in the Status Enable register of the Standard Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
standardRegister = status.standard.enable
status.standard.enable = standardRegister
```

<code>standardRegister</code>	The value of the Status Enable register of the Standard Event Register (0 to 255)
-------------------------------	---

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or value of the bit as the `standardRegister` parameter.

You can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set `standardRegister` to the sum of their decimal weights. For example, to set bits B0 and B2, set `standardRegister` to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-110) function.
1	2	Not used	Not used.
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example 1

```
standardRegister = status.standard.OPC + status.standard.QYE
status.standard.enable = standardRegister
Uses constants to set the OPC and QYE bits of the standard event status enable register.
```

Example 2

```
-- decimal 5 = binary 0000 0101
standardRegister = 5
status.standard.enable = standardRegister
Uses a decimal value to set the OPC and QYE bits of the standard event status enable register.
```

Also see

[Standard Event Register](#) (on page 16-3)
[Understanding bit settings](#) (on page 16-15)

status.standard.event

This attribute returns the contents of the Standard Event Status Register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	status.preset()	Not applicable	0

Usage

```
standardRegister = status.standard.event
standardRegister    The status of the standard event status register
```

Details

When this command returns zero (0), no bits are set. You can send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-110) function.
1	2	Not used	Not used.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

print(status.standard.event)	May return the value 129, showing that the Standard Event Status Register contains binary 10000001
------------------------------	--

Also see

[Standard Event Register](#) (on page 16-3)
[Understanding bit settings](#) (on page 16-15)

timer.cleartime()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
timer.cleartime()
```

Example

<pre>dataqueue.clear() dataqueue.add(35) timer.cleartime() delay(0.5) dt = timer.gettime() print("Delay time was " .. dt) print(dataqueue.next())</pre>	<p>Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it. Output: Delay time was 0.500099 35</p>
---	--

Also see

[timer.gettime\(\)](#) (on page 14-217)

timer.gettime()

This function measures the elapsed time since the timer was last cleared.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
time = timer.gettime()
```

time	The elapsed time in seconds (1 µs resolution)
------	---

Example

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.

Output:

```
Delay time was 0.500099
35
```

Also see

[timer.cleartime\(\)](#) (on page 14-216)

trigger.blender[N].clear()

This function clears the blender event detector and resets the overrun indicator of blender *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].clear()
```

N	The blender number (up to two)
---	--------------------------------

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

```
trigger.blender[2].clear()
```

Clears the event detector for blender 2.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender performs OR operations or AND operations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	false (AND)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The type of operation: <ul style="list-style-type: none"> ■ true: OR operation ■ false: AND operation
N	The blender number (up to two)

Details

This command selects whether the blender waits for any one event (OR) or waits for all selected events (AND) before signaling an output event.

Example

```
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.
```

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 14-219)

trigger.blender[N].overrun

This attribute indicates whether or not an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Trigger blender N clear Trigger blender N reset	Not applicable	Not applicable

Usage

```
overrun = trigger.blender[N].overrun
```

overrun	Trigger blender overrun state (true or false)
N	The blender number (up to two)

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

```
print(trigger.blender[1].overrun)
```

If an event was ignored, the output is true.
If an event was not ignored, the output is false.

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 14-219)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].reset()
```

N	The trigger event blender (up to two)
---	---------------------------------------

Details

The `trigger.blender[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.blender[N].orenable`
- `trigger.blender[N].stimulus[M]`

It also clears `trigger.blender[N].overrun`.

Example

```
trigger.blender[1].reset()
```

Resets the trigger blender 1 settings to factory defaults.

Also see

[trigger.blender\[N\].orenable](#) (on page 14-218)
[trigger.blender\[N\].overrun](#) (on page 14-218)
[trigger.blender\[N\].stimulus\[M\]](#) (on page 14-220)

trigger.blender[N].stimulus[M]

This attribute specifies the events that trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = event
```

event	The event that triggers the blender action; see Details
N	An integer that represents the trigger event blender (up to two)
M	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event.

Use none to disable the blender input.

The *event* parameter may be any of the trigger events shown in the following table.

Trigger events	
Event description	Event constant
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
A command interface trigger: <ul style="list-style-type: none"> ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command device_trigger 	trigger.EVENT_COMMAND
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN
No trigger event	trigger.EVENT_NONE
Notify trigger block <i>N</i> (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT
Trigger timer <i>N</i> (1 to 4) expired	trigger.EVENT_TIMERN
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	trigger.EVENT_TSPLINKN

Example

```

digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

```

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 14-219)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.blender[N].wait(timeout)
```

triggered	Trigger detection indication for blender
N	The trigger blender (up to two) on which to wait
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```

digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
print(trigger.blender[1].wait(3))
Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.
Wait 3 s while checking if trigger blender 1 event has occurred.

```

Also see

[trigger.blender\[N\].clear\(\)](#) (on page 14-217)

trigger.clear()

This function clears any pending command triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

A command trigger indicates if a trigger event has been detected over a command interface since the last `trigger.wait()` command was sent. Command triggers are generated by:

- Sending *TRG over a remote interface
- GPIB GET bus commands
- USBTMC trigger messages
- VXI-11 device trigger commands

`trigger.clear()` clears the command triggers and discards the history of trigger events.

Example

<pre>*TRG print(trigger.wait(1)) trigger.clear() print(trigger.wait(1))</pre>	Generate a trigger event. Check if there are any pending trigger events. Output: true Clear any pending command triggers. Check if there are any pending trigger events. Output: false
---	---

Also see

[trigger.wait\(\)](#) (on page 14-295)

trigger.continuous

This attribute determines the trigger mode setting after bootup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Never	Nonvolatile memory	<code>trigger.CONT_AUTO</code>

Usage

```
setting = trigger.continuous
trigger.continuous = setting
```

<code>setting</code>	Do not start continuous measurements after boot: <code>trigger.CONT_OFF</code> Start continuous measurements after boot: <code>trigger.CONT_AUTO</code> Place the instrument into local control and start continuous measurements after boot: <code>trigger.CONT_RESTART</code>
----------------------	--

Details

Conditions must be valid before continuous measurements can start.

When the restart parameter is selected, the instrument is placed in local mode, aborts any running scripts, and aborts any trigger models that are running. If the command is in a script, it is the last command that runs before the script is aborted. The restart parameter is not stored in nonvolatile memory, so it does not affect start up behavior.

The off and automatic parameters are stored in nonvolatile memory, so they affect start up behavior.

Example

```
trigger.continuous = trigger.CONT_OFF
```

When the instrument starts up, the measurement method is set to idle.

Also see

None

trigger.digin[N].clear()

This function clears the trigger event on a digital input line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digin[N].clear()
```

N	Digital I/O trigger line (1 to 6)
---	-----------------------------------

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to false).

Example

```
trigger.digin[2].clear()
```

Clears the trigger event detector on I/O line 2.

Also see

- [digio.line\[N\].mode](#) (on page 14-60)
- [Digital I/O port configuration](#) (on page 8-14)
- [trigger.digin\[N\].overrun](#) (on page 14-225)
- [trigger.digin\[N\].wait\(\)](#) (on page 14-225)

trigger.digin[N].edge

This attribute sets the edge used by the trigger event detector on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.digin[N].edge
trigger.digin[N].edge = detectedEdge
```

detectedEdge	The trigger logic value: <ul style="list-style-type: none"> ▪ Detect falling-edge triggers as inputs: trigger.EDGE_FALLING ▪ Detect rising-edge triggers as inputs: trigger.EDGE_RISING ▪ Detect either falling or rising-edge triggers as inputs: trigger.EDGE_EITHER ▪ See Details for descriptions of values
N	Digital I/O trigger line (1 to 6)

Details

This command sets the logic on which the trigger event detector and the output trigger generator operate on the specified trigger line.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Trigger mode values

Value	Description
trigger.EDGE_FALLING	Detects falling-edge triggers as input when the line is configured as an input or open drain
trigger.EDGE_RISING	Detects rising-edge triggers as input when the line is configured as an open drain
trigger.EDGE_EITHER	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_IN
trigger.digin[4].edge = trigger.EDGE_RISING
```

Sets the trigger mode for digital I/O line 4 to detect a rising-edge trigger as an input.

Also see

- [digio.line\[N\].mode](#) (on page 14-60)
- [digio.line\[N\].reset\(\)](#) (on page 14-61)
- [digio.writeport\(\)](#) (on page 14-64)
- [Digital I/O port configuration](#) (on page 8-14)
- [trigger.digin\[N\].clear\(\)](#) (on page 14-223)

trigger.digin[N].overrun

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Digital I/O trigger <i>N</i> clear Digital I/O trigger <i>N</i> reset	Not applicable	Not applicable

Usage

```
overrun = trigger.digin[N].overrun
```

overrun	Trigger overrun state (true or false)
<i>N</i>	Digital I/O trigger line (1 to 6)

Details

If this is true, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = trigger.digin[1].overrun
print(overrun)
```

If there is no trigger overrun on digital input 1, the output is:
false

Also see

- [digio.line\[N\].mode](#) (on page 14-60)
- [digio.line\[N\].reset\(\)](#) (on page 14-61)
- [Digital I/O port configuration](#) (on page 8-14)
- [trigger.digin\[N\].clear\(\)](#) (on page 14-223)

trigger.digin[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.digin[N].wait(timeout)
```

triggered	Trigger detected: true No triggers detected during the timeout period: false
<i>N</i>	Digital I/O trigger line (1 to 6)
timeout	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `trigger.digin[N].wait()` or `trigger.digin[N].clear()` was called, this function returns a value immediately. After waiting for a trigger with this function, the event detector is automatically reset and is ready to detect the next trigger. This is true regardless of the number of events detected.

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_IN
triggered = trigger.digin[4].wait(3)
print(triggered)
```

Waits up to 3 s for a trigger to be detected on trigger line 4, then outputs the results.
Output if no trigger is detected:
`false`
Output if a trigger is detected:
`true`

Also see

[digio.line\[N\].mode](#) (on page 14-60)
[Digital I/O port configuration](#) (on page 8-14)
[trigger.digin\[N\].clear\(\)](#) (on page 14-223)

trigger.digout[N].assert()

This function asserts a trigger pulse on one of the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digout[N].assert()
N
```

Digital I/O trigger line (1 to 6)

Details

Initiates a trigger event and does not wait for completion. The pulse width that is set determines how long the instrument asserts the trigger.

Example

```
digio.line[2].mode = digio.MODE_TRIGGER_OUT
trigger.digout[2].pulsewidth = 20e-6
trigger.digout[2].assert()
```

Asserts a trigger on digital I/O line 2 with a pulse width of 20 μ s.

Also see

[digio.line\[N\].mode](#) (on page 14-60)
[Digital I/O port configuration](#) (on page 8-14)
[trigger.digout\[N\].pulsewidth](#) (on page 14-228)

trigger.digout[N].logic

This attribute sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger <i>N</i> reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.digout[N].logic
trigger.digout[N].logic = logicType
```

<i>logicType</i>	The output logic of the trigger generator: <ul style="list-style-type: none"> ▪ Assert a TTL-high pulse for output: trigger.LOGIC_POSITIVE ▪ Assert a TTL-low pulse for output: trigger.LOGIC_NEGATIVE
<i>N</i>	Digital I/O trigger line (1 to 6)

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

The output state of the digital I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_OUT
trigger.digout[4].logic = trigger.LOGIC_NEGATIVE
```

Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

[digio.line\[N\].mode](#) (on page 14-60)
[digio.line\[N\].reset\(\)](#) (on page 14-61)
[Digital I/O port configuration](#) (on page 8-14)

trigger.digout[N].pulsewidth

This attribute describes the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger <i>N</i> reset	Configuration script	10e-6 (10 µs)

Usage

```
width = trigger.digout[N].pulsewidth
trigger.digout[N].pulsewidth = width
```

width	The pulse width (0 to 100 ks)
<i>N</i>	Digital I/O trigger line (1 to 6)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `trigger.digout[N].release()`.

Example

```
digio.line[4].mode = digio.MODE_TRIGGER_OUT
trigger.digout[4].pulsewidth = 20e-6
```

Sets the pulse width for trigger line 4 to 20 µs.

Also see

- [digio.line\[N\].mode](#) (on page 14-60)
- [digio.line\[N\].reset\(\)](#) (on page 14-61)
- [Digital I/O port configuration](#) (on page 8-14)
- [trigger.digout\[N\].assert\(\)](#) (on page 14-226)
- [trigger.digout\[N\].release\(\)](#) (on page 14-228)

trigger.digout[N].release()

This function releases an indefinite length or latched trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digout[N].release()
```

<i>N</i>	Digital I/O trigger line (1 to 6)
----------	-----------------------------------

Details

Releases a trigger that was asserted with an indefinite pulsewidth time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Example

<code>digio.line[4].mode = digio.MODE_TRIGGER_OUT trigger.digout[4].release()</code>	Releases digital I/O trigger line 4.
--	--------------------------------------

Also see

[digio.line\[N\].mode](#) (on page 14-60)
[Digital I/O port configuration](#) (on page 8-14)
[trigger.digout\[N\].assert\(\)](#) (on page 14-226)
[trigger.digout\[N\].pulsewidth](#) (on page 14-228)

trigger.digout[N].stimulus

This attribute selects the event that causes a trigger to be asserted on the digital output line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger <i>N</i> reset	Configuration script	trigger.EVENT_NONE

Usage

<code>event = trigger.digout[N].stimulus</code>	
<code>trigger.digout[N].stimulus = event</code>	
<code>event</code>	The event to use as a stimulus; see Details
<code>N</code>	Digital I/O trigger line (1 to 6)

Details

The digital trigger pulsewidth command determines how long the trigger is asserted.

The trigger stimulus for a digital I/O line can be set to one of the trigger events that are described in the following table.

Trigger events	
Event description	Event constant
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
A command interface trigger:	trigger.EVENT_COMMAND
<ul style="list-style-type: none"> ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command device_trigger 	
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN
No trigger event	trigger.EVENT_NONE

Trigger events	
Event description	Event constant
Notify trigger block N (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT
Trigger timer N (1 to 4) expired	trigger.EVENT_TIMERN
Line edge detected on TSP-Link synchronization line N (1 to 3)	trigger.EVENT_TSPLINKN

Example

```
digio.line[2].mode = digio.MODE_TRIGGER_OUT
trigger.digout[2].stimulus = trigger.EVENT_TIMER3
Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.
```

Also see

[digio.line\[N\].mode](#) (on page 14-60)
[digio.line\[N\].reset\(\)](#) (on page 14-61)
[Digital I/O port configuration](#) (on page 8-14)
[trigger.digin\[N\].clear\(\)](#) (on page 14-223)
[trigger.digout\[N\].assert\(\)](#) (on page 14-226)

trigger.lanin[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanin[N].clear()
N           The LAN event number (1 to 8) to clear
```

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

trigger.lanin[5].clear()	Clears the event detector with LAN event trigger 5.
--------------------------	---

Also see

[trigger.lanin\[N\].overrun](#) (on page 14-231)

trigger.lanin[N].edge

This attribute sets the trigger operation and detection mode of the specified LAN event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EDGE_EITHER

Usage

```
edgeMode = trigger.lanin[N].edge
trigger.lanin[N].edge = edgeMode
```

edgeMode	The trigger mode; see the Details for more information
N	The LAN event number (1 to 8)

Details

This command controls how the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values		
Mode	Trigger packets detected as input	LAN trigger packet generated for output with a...
trigger.EDGE_EITHER	Rising or falling edge (positive or negative state)	negative state
trigger.EDGE_FALLING	Falling edge (negative state)	negative state
trigger.EDGE_RISING	Rising edge (positive state)	positive state

Example

trigger.lanin[1].edge = trigger.EDGE_FALLING	Set the edge state of LAN event 1 to falling.
--	---

Also see

- [Digital I/O](#) (on page 8-12)
- [TSP-Link system expansion interface](#) (on page 9-1)

trigger.lanin[N].overrun

This attribute contains the overrun status of the LAN event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger N clear	Not applicable	Not applicable

Usage

```
overrun = trigger.lanin[N].overrun
```

overrun	The trigger overrun state for the specified LAN packet (true or false)
N	The LAN event number (1 to 8)

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

Example

```
overrun = trigger.lanin[5].overrun
print(overrun)
```

Checks the overrun status of a trigger on LAN5 and outputs the value, such as:
false

Also see

- [trigger.lanin\[N\].clear\(\)](#) (on page 14-230)
- [trigger.lanin\[N\].wait\(\)](#) (on page 14-232)
- [trigger.lanout\[N\].assert\(\)](#) (on page 14-233)
- [trigger.lanout\[N\].stimulus](#) (on page 14-238)

trigger.lanin[N].wait()

This function waits for an input trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.lanin[N].wait(timeout)
```

triggered	Trigger detection indication (true or false)
N	The trigger packet over LAN to wait for (1 to 8)
timeout	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `trigger.lanin[N].wait()` or `trigger.lanin[N].clear()` was called, this function returns immediately.

After waiting for a LAN trigger event with this function, the event detector is automatically reset and rearmed regardless of the number of events detected.

Example

```
triggered = trigger.lanin[5].wait(3)
```

Wait for a trigger event with LAN trigger 5 with a timeout of 3 s.

Also see

- [trigger.lanin\[N\].clear\(\)](#) (on page 14-230)
- [trigger.lanin\[N\].overrun](#) (on page 14-231)
- [trigger.lanout\[N\].assert\(\)](#) (on page 14-233)
- [trigger.lanout\[N\].stimulus](#) (on page 14-238)

trigger.lanout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].assert()  
N           The LAN event number (1 to 8)
```

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudo line state to the appropriate state.

The following indexes provide the listed LXI events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- ...
- 8:LAN7

Example

```
trigger.lanout[5].assert()          Creates a trigger with LAN trigger 5.
```

Also see

- [lan.lxidomain](#) (on page 14-100)
[trigger.lanin\[N\].clear\(\)](#) (on page 14-230)
[trigger.lanin\[N\].overrun](#) (on page 14-231)
[trigger.lanin\[N\].wait\(\)](#) (on page 14-232)
[trigger.lanout\[N\].assert\(\)](#) (on page 14-233)
[trigger.lanout\[N\].ipaddress](#) (on page 14-236)
[trigger.lanout\[N\].protocol](#) (on page 14-237)
[trigger.lanout\[N\].stimulus](#) (on page 14-238)

trigger.lanout[N].connect()

This function prepares the event generator for outgoing trigger events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].connect()
N           The LAN event number (1 to 8)
```

Details

This command prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event is changed.

Example

```
trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST
trigger.lanout[1].connect()
trigger.lanout[1].assert()
```

Set the protocol for LAN trigger 1 to be multicast when sending LAN triggers. Then, after connecting the LAN trigger, send a message on LAN trigger 1 by asserting it.

Also see

[trigger.lanin\[N\].overrun](#) (on page 14-231)
[trigger.lanin\[N\].wait\(\)](#) (on page 14-232)
[trigger.lanout\[N\].assert\(\)](#) (on page 14-233)
[trigger.lanout\[N\].ipaddress](#) (on page 14-236)
[trigger.lanout\[N\].protocol](#) (on page 14-237)
[trigger.lanout\[N\].stimulus](#) (on page 14-238)

trigger.lanout[N].connected

This attribute contains the LAN event connection state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
connected = trigger.lanout[N].connected
connected           The LAN event connection state:
                    ■ true: Connected
                    ■ false: Not connected
N                   The LAN event number (1 to 8)
```

Details

This is set to `true` when the LAN trigger is connected and ready to send trigger events after a successful `trigger.lanout[N].connect()` command. If the LAN trigger is not ready to send trigger events, this value is `false`.

This attribute is also `false` when the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attribute is changed or when the remote connection closes the connection.

Example

```
trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST
print(trigger.lanout[1].connected)
Outputs true if connected, or false if not connected.
Example output:
false
```

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-234)
[trigger.lanout\[N\].ipaddress](#) (on page 14-236)
[trigger.lanout\[N\].protocol](#) (on page 14-237)

trigger.lanout[N].disconnect()

This function disconnects the LAN trigger event generator.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].disconnect()
N           The LAN event number (1 to 8)
```

Details

When this command is set for TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attributes for this event are changed.

Also see

[trigger.lanout\[N\].ipaddress](#) (on page 14-236)
[trigger.lanout\[N\].protocol](#) (on page 14-237)

trigger.lanout[N].ipaddress

This attribute specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	"0.0.0.0"

Usage

```
ipAddress = trigger.lanout[N].ipaddress
trigger.lanout[N].ipaddress = "ipAddress"
```

ipAddress	The LAN address for this attribute as a string in dotted decimal notation
N	The LAN event number (1 to 8)

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

```
trigger.lanout[3].protocol = lan.PROTOCOL_TCP
trigger.lanout[3].ipaddress = "192.0.32.10"
trigger.lanout[3].connect()
```

Set the protocol for LAN trigger 3 to be TCP when sending LAN triggers.
Use IP address "192.0.32.10" to connect the LAN trigger.

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-234)

trigger.lanout[N].logic

This attribute sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.lanout[N].logic
trigger.lanout[N].logic = logicType
```

logicType	The type of logic: <ul style="list-style-type: none"> ▪ Positive: trigger.LOGIC_POSITIVE ▪ Negative: trigger.LOGIC_NEGATIVE
N	The LAN event number (1 to 8)

Example

```
trigger.lanout[2].logic = trigger.LOGIC_POSITIVE
Set the logic for LAN trigger line 2 to positive.
```

Also see

None

trigger.lanout[N].protocol

This attribute sets the LAN protocol to use for sending trigger messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	lan.PROTOCOL_TCP

Usage

```
protocol = trigger.lanout[N].protocol
trigger.lanout[N].protocol = protocol
```

protocol	The protocol to use for messages from the trigger: <ul style="list-style-type: none"> ■ lan.PROTOCOL_TCP ■ lan.PROTOCOL_UDP ■ lan.PROTOCOL_MULTICAST
N	The LAN event number (1 to 8)

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

print(trigger.lanout[1].protocol)	Get LAN protocol that is being used for sending trigger messages for LAN event 1.
-----------------------------------	---

Also see

[trigger.lanout\[N\].connect\(\) \(on page 14-234\)](#)
[trigger.lanout\[N\].ipaddress \(on page 14-236\)](#)

trigger.lanout[N].stimulus

This attribute specifies events that cause this trigger to assert.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EVENT_NONE

Usage

```
LANevent = trigger.lanout[N].stimulus
trigger.lanout[N].stimulus = LANevent
```

<i>LANevent</i>	The LAN event that causes this trigger to assert; see Details for values
<i>N</i>	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored and the action overrun is set.

Trigger events	
Event description	Event constant
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
A command interface trigger:	trigger.EVENT_COMMAND
<ul style="list-style-type: none"> ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command device_trigger 	
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN
No trigger event	trigger.EVENT_NONE
Notify trigger block <i>N</i> (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT
Trigger timer <i>N</i> (1 to 4) expired	trigger.EVENT_TIMERN
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	trigger.EVENT_TSPLINKN

Example

```
trigger.lanout[5].stimulus = trigger.EVENT_TIMER1
```

Use the timer 1 trigger event as the source for LAN trigger 5 stimulus.

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-234)

[trigger.lanout\[N\].ipaddress](#) (on page 14-236)

trigger.model.abort()

This function stops all trigger model commands on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.abort()
```

Details

When this command is received, the instrument stops the trigger model.

Example

```
trigger.model.abort()
```

Terminates all commands related to the trigger model on the instrument.

Also see

[Effect of GPIB line events on 2450](#) (on page 2-13)

[Aborting the trigger model](#) (on page 8-52)

[Trigger model](#) (on page 8-26)

trigger.model.getblocklist()

This function returns the settings for all trigger-model blocks.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.getblocklist()
```

Details

This returns the settings for the trigger model.

Example

```
print(trigger.model.getblocklist())
Returns the settings for the trigger model. Example output is:
1) BUFFER_CLEAR           BUFFER: defbuffer1
2) MEASURE                 BUFFER: defbuffer1 COUNT: 1
3) BRANCH_COUNTER          VALUE: 5  BRANCH_BLOCK: 2
4) DELAY_CONSTANT           DELAY: 1.0000000000
5) BRANCH_COUNTER          VALUE: 3  BRANCH_BLOCK: 2
```

Also see

[trigger.model.getbranchcount\(\)](#) (on page 14-240)

trigger.model.getbranchcount()

This function returns the count value of the trigger model counter block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.getbranchcount(blockNumber)
blockNumber      The sequence of the block in the trigger model
```

Details

This command returns the counter value. When the counter is active, this returns the present count. If the trigger model has started or is running but has not yet reached the counter block, this value is 0.

Example

```
reset()
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 0.1)
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
trigger.model.initiate()
delay(1)
print(trigger.model.getbranchcount(4))
waitcomplete()
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and make five readings.
Delay 0.1 s.
Loop ten more times back to block 2.
Send the count command to check which count has been completed for block 4.
At end of execution, 10 readings are stored in defbuffer1.

Also see

[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_COUNTER](#) (on page 14-254)

trigger.model.initiate()

This function starts the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.initiate()
```

Also see

[Trigger model](#) (on page 8-26)
[trigger.model.abort\(\)](#) (on page 14-239)

trigger.model.load() — ConfigList

This function loads a trigger-model template configuration that uses source and measure configuration lists.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("ConfigList", "measureConfigList", "sourceConfigList")
trigger.model.load("ConfigList", "measureConfigList", "sourceConfigList", delay)
trigger.model.load("ConfigList", "measureConfigList", "sourceConfigList", delay,
                  bufferName)
```

<i>measureConfigList</i>	A string that contains the name of the measurement configuration list to use
<i>sourceConfigList</i>	A string that contains the name of the source configuration list to use
<i>delay</i>	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1.

Details

This trigger-model template incorporates a source configuration list and measure configuration list. You must set up the configuration lists before loading the trigger model. If the configuration lists change, you must resend this command.

You can also set a delay and change the reading buffer.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Example

```

reset()
smu.source.configlist.create("SOURCE_LIST")
smu.measure.configlist.create("MEASURE_LIST")
smu.source.level = 1
smu.source.configlist.store("SOURCE_LIST")
smu.measure.range = 1e-3
smu.measure.configlist.store("MEASURE_LIST")
smu.source.level = 5
smu.source.configlist.store("SOURCE_LIST")
smu.measure.range = 10e-3
smu.measure.configlist.store("MEASURE_LIST")
smu.source.level = 10
smu.source.configlist.store("SOURCE_LIST")
smu.measure.range = 100e-3
smu.measure.configlist.store("MEASURE_LIST")
trigger.model.load("ConfigList", "MEASURE_LIST", "SOURCE_LIST")
trigger.model.initiate()

Set up a source configuration list named SOURCE_LIST and a measurement configuration list named MEASURE_LIST. Load the configuration list trigger model, using these two configuration lists. Start the trigger model.

```

Also see

None

trigger.model.load() — DurationLoop

This function loads a trigger-model template configuration that makes continuous measurements for a specified amount of time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```

trigger.model.load("DurationLoop", duration)
trigger.model.load("DurationLoop", duration, delay)
trigger.model.load("DurationLoop", duration, delay, bufferName)

```

duration	The amount of time for which to make measurements (167 ns to 100 ks)
delay	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
bufferName	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1

Details

When you load this trigger-model template, you can specify amount of time to make a measurement and the length of the delay before the measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Example

```

reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.level = 5

-- Turn on output, initiate readings, and store them in defbuffer1
trigger.model.load("DurationLoop", 10, 0.01, defbuffer1)
trigger.model.initiate()

```

Reset the instrument. Set the instrument to source voltage at 5 V. Set to measure current. Load the duration loop trigger model to take measurements for 10 s with a 10 ms delay before each measurement. Start the trigger model.

Also see

None

trigger.model.load() — Empty

This function clears the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.load("Empty")
```

Details

When you load this trigger-model template, any blocks that have been defined in the trigger model are cleared so the trigger model has no blocks defined.

Example

```

trigger.model.load("Empty")
print(trigger.model.getblocklist())

```

Clear the trigger model to have no blocks defined.
Output:
EMPTY

Also see

None

trigger.model.load() — GradeBinning

This function loads a trigger-model template configuration that sets up a grading operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern, bufferName)
```

<i>components</i>	The number of components to measure (1 to 268,435,455)
<i>startInLine</i>	The input line that starts the test; 5 for digital line 5, 6 for digital line 6; default is 5
<i>startDelay</i>	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
<i>endDelay</i>	The delay time after the measurement (167 ns to 10 ks); default is 0 for no delay
<i>limitxHigh</i>	<i>x</i> is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
<i>limitxLow</i>	<i>x</i> is 1, 2, 3, or 4; the lower limit that the measurement is compared against
<i>limit1Pattern</i>	The bit pattern that is sent when the measurement fails limit 1; range 1 to 15; default is 1
<i>limit2Pattern</i>	The bit pattern that is sent when the measurement fails limit 2; range 1 to 15; default is 2

<i>limit3Pattern</i>	The bit pattern that is sent when the measurement fails limit 3; range 1 to 15; default is 4
<i>limit4Pattern</i>	The bit pattern that is sent when the measurement fails limit 4; range 1 to 15; default is 8
<i>allPattern</i>	The bit pattern that is sent when all limits have passed; 1 to 15; default is 15
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; defaults to <code>defbuffer1</code>

Details

This trigger-model template allows you to grade components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the pass pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getBlocklist()` command to view the trigger-model blocks in a list format.

Also see

None

trigger.model.load() — LogicTrigger

This function loads a trigger-model template configuration that sets up a logic trigger through the digital I/O.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear)
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear, delay)
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear, delay,
    bufferName)
```

<i>digInLine</i>	The digital input line (1 to 6); also the event that the trigger model will wait on in block 1
<i>digOutLine</i>	The digital output line (1 to 6)
<i>count</i>	The number of measurements the instrument will make
<i>clear</i>	To clear previously detected trigger events when entering the wait block: <code>trigger.CLEAR_ENTER</code> To immediately act on any previously detected triggers and not clear them (default): <code>trigger.CLEAR_NEVER</code>
<i>delay</i>	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; defaults to <code>defbuffer1</code>

Details

This trigger model waits for a digital input event to occur, makes a measurement, and issues a notify event. The notify event asserts a digital output line.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

This command replaces the `trigger.model.load()` — `ExternalTrigger` command, which is deprecated. Note that the `clear` parameter is not available for `ExternalTrigger`.

Example

```
trigger.model.load("LogicTrigger", 1, 2, 10, 0.001, defbuffer1)
```

Set up the template to use the digital in line and wait for a pulse from digital in line 1 to trigger measurements.

Pulse digital out line 2 when the measurement is complete.

Make 10 measurements, with a delay of 1 ms before each measurement.

Store the measurements in `defbuffer1`.

Also see

[trigger.digout\[N\].logic](#) (on page 14-227)

trigger.model.load() — LoopUntilEvent

This function loads a trigger-model template configuration that makes continuous measurements until the specified event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear)
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear, delay)
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear, delay,
                  bufferName)
```

<code>triggerEvent</code>	The event that ends infinite triggering or readings set to occur before the trigger; see Details
<code>position</code>	The number of readings to make in relation to the size of the reading buffer; enter as a percentage (0% to 100%)
<code>clear</code>	To clear previously detected trigger events when entering the wait block (default): <code>trigger.CLEAR_ENTER</code> To immediately act on any previously detected triggers and not clear them: <code>trigger.CLEAR_NEVER</code>
<code>delay</code>	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
<code>bufferName</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; defaults to <code>defbuffer1</code>

Details

The event constant is the event that ends infinite triggering or ends readings set to occur before the trigger and start post-trigger readings. The trigger model makes readings until it detects the event constant. After the event, it makes a finite number of readings, based on the setting of the trigger position.

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled. For example, if this is set to 75 for a reading buffer that holds 10,000 readings, the trigger model makes 2500 readings after it detects the source event. There are 7500 pre-trigger readings and 2500 post-trigger readings.

The instrument makes two sets of readings. The first set is made until the trigger event occurs. The second set is made after the trigger event occurs, up to the number of readings calculated by the position parameter.

You cannot have the event constant set at none when you run this trigger-model template.

The following table lists the options that are available for *triggerEvent*.

Trigger events	
Event description	Event constant
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Notify trigger block <i>N</i> (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
A command interface trigger:	trigger.EVENT_COMMAND
<ul style="list-style-type: none"> ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command device_trigger 	
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	trigger.EVENT_TSPLINKN
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
Trigger timer <i>N</i> (1 to 4) expired	trigger.EVENT_TIMERN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Example

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT

-- Initiate readings
trigger.model.load("LoopUntilEvent", trigger.EVENT_DISPLAY, 50)
trigger.model.initiate()

Reset the instrument.
Set the instrument to measure current.
Load the LoopUntilEvent trigger model to make measurements until the front panel trigger key is pressed, then
continue to make measurements equal to 50% of the reading buffer size.
Start the trigger model.
```

Also see

None

trigger.model.load() — SimpleLoop

This function loads a trigger-model template configuration that makes a specific number of measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("SimpleLoop", count)
trigger.model.load("SimpleLoop", count, delay)
trigger.model.load("SimpleLoop", count, delay, bufferName)
```

count	The number of measurements the instrument will make
delay	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
bufferName	Indicates the reading buffer to use; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Example

```
reset()

-- Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1

-- Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

-- Turn on output and initiate readings
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
    print(i, defbuffer1.relativetimes[i], defbuffer1[i])
end

-- Discharge the capacitor to 0 V and turn off the output
smu.source.level = 0
delay(2)
smu.source.output = smu.OFF
```

This example uses the SimpleLoop trigger-model template to do a capacitor test. This example produces 200 readings that have output similar to the following example:

Rdg #	Time (s)	Current (A)
1	0	8.5718931952528e-11
2	0.151875	1.6215984111057e-10
3	0.303727	1.5521139928865e-10
...		
198	29.91579194	1.5521250951167e-10
199	30.067648716	1.4131290582142e-10
200	30.219497716	1.5521067764368e-10

Also see

None

trigger.model.load() — SortBinning

This function loads a trigger-model template configuration that sets up a sorting operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern, bufferName)
```

components	The number of components to measure (1 to 268,435,455)
limitxHigh	x is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
limitxLow	x is 1, 2, 3, or 4; the lower limit that the measurement is compared against
limit1Pattern	The bit pattern that is sent when the measurement passes limit 1; range 1 to 15; default is 1
limit2Pattern	The bit pattern that is sent when the measurement passes limit 2; range 1 to 15; default is 2
limit3Pattern	The bit pattern that is sent when the measurement passes limit 3; range 1 to 15; default is 4

<i>limit4Pattern</i>	The bit pattern that is sent when the measurement passes limit 4; range 1 to 15; default is 8
<i>allPattern</i>	The bit pattern that is sent when all limits have failed; 1 to 15; default is 15
<i>startInLine</i>	The input line that starts the test; 5 for digital line 5, 6 for digital line 6; default is 5
<i>startDelay</i>	The delay time before each measurement (167 ns to 10 ks); default is 0 for no delay
<i>endDelay</i>	The delay time after the measurement (167 ns to 10 ks); default is 0 for no delay
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer; defaults to <i>defbuffer1</i>

Details

This trigger-model template allows you to sort components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the all fail pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Also see

None

trigger.model.pause()

This function pauses a running trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.pause()
```

Details

This command pauses the trigger model.

To continue the trigger model, send the resume command.

Example

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
trigger.model.pause()
delay(10)
trigger.model.resume()
waitcomplete()
print(defbuffer1.n)
```

Set up a trigger model, then run it, pause for delay of 10 seconds, then resume it.

Also see

[trigger.model.initiate\(\)](#) (on page 14-241)
[trigger.model.resume\(\)](#) (on page 14-252)

trigger.model.resume()

This function continues a paused trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.resume()
```

Details

This command continues running the trigger-model operation if the trigger model was paused.

Example

```

reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
trigger.model.pause()
delay(10)
trigger.model.resume()
waitcomplete()
print(defbuffer1.n)

```

Set up a trigger model, then run it, pause for delay of 10 seconds, then resume it.

Also see

[trigger.model.initiate\(\)](#) (on page 14-241)
[trigger.model.pause\(\)](#) (on page 14-251)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS

This function defines a trigger-model block that always goes to a specific block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ALWAYS, branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number to execute when the trigger model reaches the Branch Always block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by *branchToBlock*.

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ALWAYS, 20)
```

When the trigger model reaches block 6, always branch to block 20.

Also see

None

trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER

This function defines a trigger-model block that branches to a specified block a specified number of times.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_COUNTER, targetCount,
                      branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
targetCount	The number of times to repeat
branchToBlock	The block number of the trigger-model block to execute when the counter is less than the <i>targetCount</i> value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value. At that point, the trigger model continues to the next building block in the sequence.

The counter is reset to 0 when the trigger model starts. It is incremented each time trigger model execution reaches the counter block.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting. Use `trigger.model.getbranchcount()` to query the counter.

Example

```
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
print(trigger.model.getbranchcount(4))
```

When the trigger model reaches this block, the trigger model returns to block 2. This repeats 10 times.

An example of the return if the trigger model has reached this block 5 times is:

5

Also see

[trigger.model.getbranchcount\(\)](#) (on page 14-240)

[trigger.model.setblock\(\) — trigger.BLOCK_RESET_BRANCH_COUNT](#) (on page 14-274)

trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA

This function defines a trigger-model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
                      branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
                      branchToBlock, measureBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>targetDifference</i>	The value against which the block compares the difference between the measurements
<i>branchToBlock</i>	The block number of the trigger-model block to execute when the difference between the measurements is less than or equal to the <i>targetDifference</i>
<i>measureBlock</i>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_DELTA, 0.35, 8, 3)
```

Configure trigger block 5 to branch to block 8 when the measurement difference from block 3 is less than 0.35.

Also see

[Delta block](#) (on page 8-41)

trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT

This function defines a trigger-model block that goes to a specified block if a measurement meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
                      limitA, limitB, branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
                      limitA, limitB, branchToBlock, measureBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>limitType</i>	The type of limit, which can be one of the following types: <ul style="list-style-type: none"> ■ trigger.LIMIT ABOVE ■ trigger.LIMIT BELOW ■ trigger.LIMIT INSIDE ■ trigger.LIMIT OUTSIDE
<i>limitA</i>	The lower limit that the measurement is tested against; if <i>limitType</i> is set to: <ul style="list-style-type: none"> ■ trigger.LIMIT ABOVE: This value is ignored ■ trigger.LIMIT BELOW: The measurement must be below this value ■ trigger.LIMIT INSIDE: The low limit that the measurement is compared against ■ trigger.LIMIT OUTSIDE: The low limit that the measurement is compared against
<i>limitB</i>	The upper limit that the measurement is tested against; if <i>limitType</i> is set to: <ul style="list-style-type: none"> ■ trigger.LIMIT ABOVE: The measurement must be above this value ■ trigger.LIMIT BELOW: This value is ignored ■ trigger.LIMIT INSIDE: The high limit that the measurement is compared against ■ trigger.LIMIT OUTSIDE: The high limit that the measurement is compared against
<i>branchToBlock</i>	The block number of the trigger-model block to execute when the measurement meets the defined criteria
<i>measureBlock</i>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected
- Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected
- Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
- Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value

The measurement block must be a measure/digitize block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measure/digitize block is used.

If the limit A is more than the limit B, the values are automatically swapped so that the lesser value is used as the lower limit.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_ABOVE,
    0.1, 1, 2)
```

Sets trigger block 5 to be a constant limit that branches to block 2 when the measurement is above the value set for limit B (which is set to 1). Note that limit A must be set, but is ignored.

Also see

[Constant Limit block](#) (on page 8-39)

trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC

This function defines a trigger-model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    limitNumber, branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    limitNumber, branchToBlock, measureBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>limitType</i>	The type of limit, which can be one of the following types: <ul style="list-style-type: none"> ▪ trigger.LIMIT_ABOVE ▪ trigger.LIMIT_BELOW ▪ trigger.LIMIT_INSIDE ▪ trigger.LIMIT_OUTSIDE

<i>limitNumber</i>	The limit number (1 or 2)
<i>branchToBlock</i>	The block number of the trigger-model block to execute when the measurement meets the criteria set in the configuration list
<i>measureBlock</i>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-dynamic-limits block defines a trigger-model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

Example

```
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC,
                      trigger.LIMIT_OUTSIDE, 2, 10, 5)
```

Configure block 7 to check if limit 2 is outside its limit values, based on the measurements made in block 5. If values are outside the measurements, branch to block 10. If the values are not outside the measurements, trigger model execution continues to block 8.

Also see

- [Dynamic Limit block](#) (on page 8-40)
- [smu.measure.limit\[Y\].low.value](#) (on page 14-151)
- [smu.measure.limit\[Y\].high.value](#) (on page 14-150)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT

This function branches to a specified block when a specified trigger event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ON_EVENT, event,
                      branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
event	The event that must occur before the trigger model branches the specified block
branchToBlock	The block number of the trigger-model block to execute when the specified event occurs

Details

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The following table shows the constants for the events.

Trigger events	
Event description	Event constant
Trigger event blender N (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
A command interface trigger: <ul style="list-style-type: none"> ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command device_trigger 	trigger.EVENT_COMMAND
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line N (1 to 6)	trigger.EVENT_DIGION
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Appropriate LXI trigger packet is received on LAN trigger object N (1 to 8)	trigger.EVENT_LANN
No trigger event	trigger.EVENT_NONE
Notify trigger block N (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT

Trigger events	
Event description	Event constant
Trigger timer N (1 to 4) expired	trigger.EVENT_TIMERN
Line edge detected on TSP-Link synchronization line N (1 to 3)	trigger.EVENT_TSPLINKN

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ON_EVENT, trigger.EVENT_DISPLAY, 2)
When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger
model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the
next block in the trigger model).
```

Also see

[On event block](#) (on page 8-37)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE

This function causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE, branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger-model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time trigger model execution encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when trigger model execution reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model execution encounters this block.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE, 4)
```

When the trigger model reaches block 2, the trigger model goes to block 4 instead of going in the default sequence of block 3.

Also see

[Once block](#) (on page 8-42)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED

This function defines a trigger-model block that causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE_EXCLUDED,
                      branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger-model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE_EXCLUDED, 4)
```

When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

[Once excluded block](#) (on page 8-42)

trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR

This function defines a trigger-model block that clears the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR, bufferName)
```

blockNumber	The sequence of the block in the trigger model
bufferName	The name of the buffer, which must be an existing buffer; if no buffer is defined, defbuffer1 is used

Details

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

Example

```
trigger.model.setblock(3, trigger.BLOCK_BUFFER_CLEAR, capTest2)
```

Assign trigger block 3 to buffer clear; when the trigger model reaches block 3, it clears the reading buffer named `capTest2`.

Also see

[buffer.make\(\)](#) (on page 14-14)

[Buffer clear block](#) (on page 8-28)

trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT

This function recalls the settings at the next index of a source or measure configuration list, or both a source and measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_NEXT, "configurationList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_NEXT, "configurationList",
                      "optionalConfigList")
```

<code>blockNumber</code>	The sequence of the block in the trigger model
<code>configurationList</code>	A string that defines the source or measure configuration list to recall
<code>optionalConfigList</code>	The name of the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list

Details

If two configuration lists are specified with this command, they must not be of the same type. For example, if the first configuration list is a measure configuration list, the second configuration list must be a source configuration list. The order of the configuration lists does not matter with this command, as long as they are of the opposite type.

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger-model block. If the configuration list has recalled an index with the recall block, the next index in the list is recalled instead of the first. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the next block is reached after that recall, index 2 is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Next block.

Example 1

```
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "measTrigList")  
Configure trigger block 5 to load the next index in the configuration list named measTrigList.
```

Example 2

```
trigger.model.load("Empty")  
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "measTrigList")  
trigger.model.setblock(2, trigger.BLOCK_BUFFER_CLEAR)  
trigger.model.setblock(3, trigger.BLOCK_CONFIG_NEXT, "measTrigList")  
print(trigger.model.getblocklist())  
  
Clear the trigger model.  
Recall index 1 of a configuration list named measTrigList.  
Clear reading buffer named defbuffer1.  
Recall the second index of a configuration list named measTrigList.  
Output:  
1) CONFIG_RECALL           CONFIG_LIST: measTrigList   INDEX: 1  
2) BUFFER_CLEAR             BUFFER: defbuffer1  
3) CONFIG_NEXT              CONFIG_LIST: measTrigList
```

Example 3

```
trigger.model.setblock(7, trigger.BLOCK_CONFIG_NEXT, "measTrigList",  
                     "sourTrigList")  
Configure trigger block 7 to load the next index in both the configuration list named measTrigList and the  
configuration list named sourTrigList.
```

Also see

[Configuration lists](#) (on page 4-84)

trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV

This function defines a trigger-model block that recalls the settings stored at the previous index in a source or measure configuration list, or both a source and measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_PREV, "configurationList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_PREV, "configurationList",
                      "optionalConfigList")
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>configurationList</i>	A string that defines the source or measure configuration list to recall
<i>optionalConfigList</i>	A string that defines the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list

Details

If two configuration lists are specified with this command, they must not be of the same type. For example, if the first configuration list is a measure configuration list, the second configuration list must be a source configuration list. The order of the configuration lists does not matter with this command, as long as they are of the opposite type.

The Config List Prev block defines a trigger-model block that recalls the settings stored at the previous index in a source or measure configuration list if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are each recalled from the previous index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled if the list has not already had an index recalled by the recall block command in an earlier trigger-model block. If the configuration list has recalled an index with the recall block, the previous index in the list is called instead of the last. For example, the recall block recalls index 1 by default, so if the trigger model uses a recall block before this one, the first time the previous block is reached after that recall, the last index is recalled. However, if the recall block recalled index 3, the previous block would recall index 2. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration lists must be defined before you can use this block.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Prev block.

Example 1

```
trigger.model.setblock(8, trigger.BLOCK_CONFIG_PREV, "measTrigList")
Configure trigger block 8 to load the previous index in the configuration list named measTrigList.
```

Example 2

```
trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 3)
trigger.model.setblock(2, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(3, trigger.BLOCK_CONFIG_PREV, "measTrigList")
print(trigger.model.getblocklist())
Clear the trigger model.
Recall index 3 of a configuration list named measTrigList.
Clear reading buffer named defbuffer1.
Then, recall the second index of a configuration list named measTrigList.
Output:
1) CONFIG_RECALL           CONFIG_LIST: measTrigList   INDEX: 3
2) BUFFER_CLEAR             BUFFER: defbuffer1
3) CONFIG_PREV              CONFIG_LIST: measTrigList
```

Example 3

```
trigger.model.setblock(7, trigger.BLOCK_CONFIG_PREV, "measTrigList",
                      "sourTrigList")
Configure trigger block 7 to load the previous index in both the configuration list named measTrigList and the configuration list named sourTrigList.
```

Also see

[Configuration lists](#) (on page 4-84)

trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL

This function recalls the system settings that are stored in a source or measure configuration list, or both a source and measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
                      "configurationList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
                      "configurationList", index)
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
                      "configurationList", index, "optionalConfigList")
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
                      "configurationList", index, "optionalConfigList", optionalIndex)
```

blockNumber	The sequence of the block in the trigger model
configurationList	A string that defines the source or measure configuration list to recall
index	The index in the configuration list to recall; default is 1

<i>optionalConfigList</i>	A string that defines the second configuration list to recall the index from; must be the opposite type of list than the first; for example, if the first configuration list is a measure list, the second configuration list must be a source list
<i>optionalIndex</i>	The index to recall from the second configuration list; defaults to 1

Details

If two configuration lists are specified with this command, they must not be of the same type. For example, if the first configuration list is a measure configuration list, the second configuration list must be a source configuration list. The order of the configuration lists does not matter with this command, as long as they are of the opposite type.

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled if a single configuration list is specified. If both measure and source configuration lists are specified, measure and source settings are recalled from the next index in each list when this block is reached. The index numbers recalled may not match; it depends on the number of indexes in each list and what index number each list is on.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration lists must be defined before you can use this block. If one of the indices for the configuration list changes, verify that the trigger model count is still accurate.

If you need to swap the source and measure configuration lists, you need to delete this block and create a new Config List Recall block.

Example 1

```
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 5)
Configure trigger block 3 to load index 5 from the configuration list named measTrigList.
```

Example 2

```
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 5,
    "sourTrigList")
print(trigger.model.getblocklist())
Configure trigger block 3 to load index 5 from the configuration list named measTrigList and load index 1
from the configuration list name sourTrigList.
Query the configuration of the block.
Output:
5) CONFIG_RECALL    CONFIG_LIST: measTrigList and sourTrigList  INDEX: 5 and 1
```

Also see

[Configuration lists](#) (on page 4-84)

trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT

This function adds a constant delay to the execution of a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DELAY_CONSTANT, time)
```

blockNumber	The sequence of the block in the trigger model
time	The amount of time to delay in seconds (167 ns to 10 ks, or 0 for no delay)

Details

When trigger model execution reaches a delay block, it stops normal measurement and trigger-model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

Example

```
trigger.model.setblock(7, trigger.BLOCK_DELAY_CONSTANT, 30e-3)
```

Configure trigger block 7 to delay the trigger model before the next block until a delay of 30 ms elapses.

Also see

None

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC

This function adds a user delay to the execution of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DELAY_DYNAMIC,
                      trigger.USER_DELAY_Mn)
```

blockNumber	The sequence of the block in the trigger model
userDelay	<p>The number of the user delay:</p> <ul style="list-style-type: none"> ■ trigger.USER_DELAY_Mn, where n is the number of the user delay (1 to 5) set by smu.measure.userdelay[N] ■ trigger.USER_DELAY_Sn, where n is the number of the user delay (1 to 5) set by smu.source.userdelay[N]

Details

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger-model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Each source function can also have up to five unique user delay times (S1 to S5). The delay time is set by the user-delay command, which is only available over a remote interface.

Though the trigger model can be used with any function, the user delay is set per function. Make sure you are setting the delay for the function you intend to use with the trigger model.

Example

```
smu.measure.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay for measure 1 to 5 s.
 Set trigger block 1 to turn the source output on.
 Set trigger block 2 to a dynamic delay that calls measure user delay 1.
 Set trigger block 3 to make a measurement.
 Set trigger block 4 to turn the source output off.
 Set trigger block 5 to branch to block 1 ten times.
 Start the trigger model.

Also see

[smu.measure.userdelay\[N\]](#) (on page 14-169)
[smu.source.userdelay\[N\]](#) (on page 14-200)

trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO

This function defines a trigger-model block that sets the lines on the digital I/O port high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DIGITAL_IO, bitPattern, bitMask)
```

blockNumber	The sequence of the block in the trigger model
bitPattern	Sets the value that specifies the output line bit pattern (0 to 63)
bitMask	Specifies the bit mask; if omitted, all lines are driven (0 to 63)

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern. The pattern can be specified as a six-bit binary, hexadecimal, or integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63, 0x3F, 0b111111) or omit this parameter. If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

```
for x = 3, 6 do digio.line[x].mode = digio.MODE_DIGITAL_OUT end
trigger.model.setblock(4, trigger.BLOCK_DIGITAL_IO, 20, 60)
```

The for loop configures digital I/O lines 3 through 6 as digital outputs. Trigger block 4 is then configured with a bit pattern of 20 (digital I/O lines 3 and 5 high). The optional bit mask is specified as 60 (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

[digio.line\[N\].mode](#) (on page 14-60)

trigger.model.setblock() — trigger.BLOCK_LOG_EVENT

This function allows you to log an event in the event log when the trigger model is running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_LOG_EVENT, eventNumber, "message")
```

blockNumber	The sequence of the block in the trigger model
eventNumber	<p>The event number:</p> <ul style="list-style-type: none"> ■ trigger.LOG_INFON ■ trigger.LOG_WARNN ■ trigger.LOG_ERRORN <p>Where <i>n</i> is 1 to 4; you can define up to four of each type You can also set trigger.LOG_WARN_ABORT, which aborts the trigger model immediately and posts a warning event log message</p>
message	A string up to 31 characters

Details

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger-model blocks.

Example

```
trigger.model.setblock(9, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Trigger model complete.")
```

Set trigger-model block 9 to log an event when the trigger model completes. In the event log, the message is:
TM #1 block #9 logged: Trigger model complete.

Also see

None

trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE

This function defines a trigger block that makes or digitizes a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE, bufferName)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE, bufferName,
count)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>bufferName</i>	The name of the buffer, which must be an existing buffer; if no buffer is defined, defbuffer1 is used
<i>count</i>	<p>The number of measure or digitize readings to make before moving to the next block in the trigger model; set to:</p> <ul style="list-style-type: none"> ■ A specific value ■ Infinite (run continuously until stopped): <code>trigger.COUNT_INFINITE</code> ■ Stop infinite to stop the block: <code>trigger.COUNT_STOP</code> ■ Use most recent count value: <code>trigger.COUNT_AUTO</code>

Details

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

Digitized measurements are not a feature on the 2450. However, you can use this command to communicate with other Keithley instruments that do offer the digitized measurements feature and to share code with other Keithley instruments.

Firmware versions of the 2450 before version 1.7.0 had a separate measure block. If you have code that is using that block, it works in this version of the 2450 firmware.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1. For example, `trigger.model.setblock(1, trigger.BLOCK_MEASURE)` changes to `trigger.model.setblock(1, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1, 1)`.

Example 1

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.

Set the function to measure DC voltage.

Set block 1 to clear defbuffer1.

Set block 2 to set a delay of 0.

Set block 3 to make measurements infinitely.

Set block 4 to wait until the front-panel TRIGGER key is pressed.

Set block 5 to stop making measurements.

Set block 6 to send a notification.

Start the trigger model.

You must press the front-panel TRIGGER key to stop measurements.

Output the number of readings.

Example 2

```

reset()
smu.measure.configlist.create("countactive")
smu.measure.count = 2
smu.measure.configlist.store("countactive") -- index1
smu.measure.count = 10
smu.measure.configlist.store("countactive") -- index2
smu.measure.count = 3
smu.measure.configlist.store("countactive") -- index3

trigger.model.setblock(1, trigger.BLOCK_CONFIG_NEXT, "countactive")
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_AUTO)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 1)
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 3, 1)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)

```

Reset the instrument.

Set up a configuration list named `countactive`.

Set the measure count to 2 (replace `smu.measure.count` with `smu.digitize.count` if using a digitize function.)

Store the count in index 1.

Set the measure count to 10.

Store the count in index 2.

Set the measure count to 3.

Store the count in index 3.

Set up trigger-model block 1 to call the next index from the `countactive` configuration list.

Set block 2 to measure or digitize and store the readings in `defbuffer1`, using the most recent count value.

Set block 3 to add a delay of 1 s.

Set block 4 to iterate through the trigger model 3 times, returning to block 1.

Start the trigger model.

Output the number of readings. There should be 15 readings.

Also see

[buffer.make\(\)](#) (on page 14-14)

[Measure/Digitize block](#) (on page 8-28)

trigger.model.setblock() — trigger.BLOCK_NOP

This function creates a placeholder that performs no action in the trigger model; available only using remote commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOP)
```

<code>blockNumber</code>	The sequence of the block in the trigger model
--------------------------	--

Details

If you remove a trigger-model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

```
trigger.model.setblock(4, trigger.BLOCK_NOP)
```

Set block number 4 to be a no operation block.

Also see

None

trigger.model.setblock() — trigger.BLOCK_NOTIFY

This function defines a trigger-model block that generates a trigger event and immediately continues to the next block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFYN)
```

blockNumber	The sequence of the block in the trigger model
N	The identification number of the notification; 1 to 8

Details

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

Define trigger-model block 5 to be the notify 2 event. Assign the notify 2 event to be the stimulus for digital output line 3.

Also see

[Notify block](#) (on page 8-32)

trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT

This function creates a block in the trigger model that resets a branch counter to 0.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_RESET_BRANCH_COUNT, counter)
```

blockNumber	The sequence of the block in the trigger model
counter	The block number of the counter that is to be reset

Details

When the trigger model reaches the Counter Reset block, it resets the count of the specified Branch on Counter block to zero.

Example

```
trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_BRANCH_COUNTER, 5, 2)
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 3, 2)
trigger.model.setblock(6, trigger.BLOCK_RESET_BRANCH_COUNT, 3)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and take 5 readings.
Delay a second.
Loop three more times back to block 2.
Reset block 3 to 0.
Start the trigger model and wait for measurements to complete.
Print the number of readings in the buffer.
Output:
15

Also see

[trigger.model.getbranchcount\(\)](#) (on page 14-240)

[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_COUNTER](#) (on page 14-254)

trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT

This function defines a trigger block that turns the output source on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_SOURCE_OUTPUT, state)
```

blockNumber	The sequence of the block in the trigger model
state	Turn the source off: smu.OFF Turn the source on: smu.ON

Details

The source output block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as SOURCE_OUTPUT.

Example

```
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
```

Set trigger model to turn the source on when it reaches block 2.

Also see

[Wait block](#) (on page 8-29)

trigger.model.setblock() — trigger.BLOCK_WAIT

This function defines a trigger-model block that waits for an event before allowing the trigger model to continue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear, logic, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear, logic, event,
                      event)
```

blockNumber	The sequence of the block in the trigger model
-------------	--

<i>event</i>	The event that must occur before the trigger block allows trigger execution to continue (see Details)
<i>clear</i>	To clear previously detected trigger events when entering the wait block: <code>trigger.CLEAR_ENTER</code> To immediately act on any previously detected triggers and not clear them (default): <code>trigger.CLEAR_NEVER</code>
<i>logic</i>	If each event must occur before the trigger model continues: <code>trigger.WAIT_AND</code> If at least one of the events must occur before the trigger model continues: <code>trigger.WAIT_OR</code>

Details

You can use the wait block to synchronize measurements with other instruments and devices.

You can set the instrument to wait for the following events:

- Front-panel TRIGGER key press
- Notify (only available when using remote commands)
- Command interface trigger
- Digital input/output signals, such as DIGIO and TSP-Link
- LAN
- Blender
- Timer
- Source limit condition

The event can occur before trigger model execution reaches the wait block. If the event occurs after trigger model execution starts but before the trigger model execution reaches the wait block, the trigger model records the event. By default, when trigger model execution reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when trigger model execution is at the start block and when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action; you cannot combine AND and OR logic in a single block.

You cannot leave the first event as no trigger. If the first event is not defined, the trigger model errors when you attempt to initiate it.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the Wait block. Set the Notify block to use the same timer as the Wait block.

The following table shows the constants for the events.

Trigger events	
Event description	Event constant
Trigger event blender N (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
A command interface trigger:	
<ul style="list-style-type: none"> ■ Any remote interface: *TRG ■ GPIB only: GET bus command ■ USB only: A USBTMC TRIGGER message ■ VXI-11: VXI-11 command device_trigger 	trigger.EVENT_COMMAND
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line N (1 to 6)	trigger.EVENT_DIGION
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Appropriate LXI trigger packet is received on LAN trigger object N (1 to 8)	trigger.EVENT_LANN
No trigger event	trigger.EVENT_NONE
Notify trigger block N (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT
Trigger timer N (1 to 4) expired	trigger.EVENT_TIMERN
Line edge detected on TSP-Link synchronization line N (1 to 3)	trigger.EVENT_TSPLINKN

Example

```
trigger.model.setblock(9, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
Set trigger-model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing.
```

Also see

[Wait block](#) (on page 8-29)

trigger.model.state()

This function returns the present state of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status, status, n = trigger.model.state()
```

status	The status of the trigger model: <ul style="list-style-type: none"> ■ trigger.STATE_ABORTED ■ trigger.STATE_ABORTING ■ trigger.STATE_BUILDING ■ trigger.STATE_EMPTY ■ trigger.STATE_FAILED ■ trigger.STATE_IDLE ■ trigger.STATE_RUNNING ■ trigger.STATE_WAITING
n	The last trigger-model block that was executed

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model last executed.

The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped

Example

```
print(trigger.model.state())
```

An example output if the trigger model is waiting and is at block 9 would be:
trigger.STATE_WAITING trigger.STATE_EMPTY 9

Also see

None

trigger.timer[N].clear()

This function clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].clear()
N           Trigger timer number (1 to 4)
```

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

trigger.timer[1].clear()	Clears trigger timer 1.
--------------------------	-------------------------

Also see

[trigger.timer\[N\].count](#) (on page 14-279)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event or is enabled as a timer or alarm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
count           Number of times to repeat the trigger (0 to 1,048,575)
N               Trigger timer number (1 to 4)
```

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

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

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example 1

```
print(trigger.timer[1].count)
Read trigger count for timer number 1.
```

Example 2

```
reset()
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY8
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON

trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY8)
trigger.model.setblock(3, trigger.BLOCK_WAIT, trigger.EVENT_TIMER4)
trigger.model.setblock(4, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 20, 3)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.

Reset trigger timer 4.

Set trigger timer 4 to have a 0.5 s delay.

Set the stimulus for trigger timer 4 to be the notify 8 event.

Set the trigger timer 4 stimulus to off.

Set the timer event to occur when the timer delay elapses.

Set the trigger timer 4 count to 20.

Enable trigger timer 4.

Clear the trigger model.

Set trigger-model block 1 to clear the buffer.

Set trigger-model block 2 to generate the notify 8 event.

Set trigger-model block 3 to wait for the trigger timer 4 to occur.

Set trigger-model block 4 to make a measurement and store it in default buffer 1.

Set trigger-model block 5 to repeat the trigger model 20 times, starting at block 3.

Start the trigger model.

Wait until all commands are complete.

Print the number of entries in default buffer 1.

Output:

20

Also see

[trigger.timer\[N\].clear\(\)](#) (on page 14-279)

[trigger.timer\[N\].delay](#) (on page 14-281)

[trigger.timer\[N\].reset\(\)](#) (on page 14-283)

trigger.timer[N].delay

This attribute sets and reads the timer delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	10e-6 (10 µs)

Usage

<code>interval = trigger.timer[N].delay</code>	
<code>trigger.timer[N].delay = <i>interval</i></code>	Delay interval in seconds (8 µs to 100 ks)
<i>N</i>	Trigger timer number (1 to 4)

Details

Once the timer is enabled, each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

```
trigger.timer[N].delaylist = {interval}
```

This creates a delay list of one value.

Reading this attribute returns the delay interval that will be used the next time the timer is triggered.

If you use the trigger timer with a trigger model, make sure the trigger timer delay is set so that the readings are paced correctly.

Example

<code>trigger.timer[1].delay = 50e-6</code>	Set the trigger timer 1 to delay for 50 µs.
---	---

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-283)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	10e-6 (10 µs)

Usage

<code>intervals = trigger.timer[N].delaylist</code>	
<code>trigger.timer[N].delaylist = <i>intervals</i></code>	Table of delay intervals in seconds
<i>N</i>	Trigger timer number (1 to 4)

Details

Each time the timer is triggered after it is enabled, it uses the next delay period from the array. The default value is an array with one value of 10 μ s.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be $\geq 50 \mu$ s.

Example

```
trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}
DelayList = trigger.timer[3].delaylist
for x = 1, table.getn(DelayList) do
    print(DelayList[x])
end
```

Set a delay list on trigger timer 3 with three delays (50 μ s, 100 μ s, and 150 μ s).

Read the delay list on trigger timer 3.

Output:

```
5e-05
0.0001
0.00015
```

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-283)

trigger.timer[N].enable

This attribute enables the trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].enable
trigger.timer[N].enable = state
```

state	Disable the trigger timer: trigger.OFF Enable the trigger timer: trigger.ON
N	Trigger timer number (1 to 4)

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

Example

<code>trigger.timer[3].enable = trigger.ON</code>	Enable the trigger timer for timer 3.
---	---------------------------------------

Also see

None

trigger.timer[N].reset()

This function resets trigger timer settings to their default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

<code>trigger.timer[N].reset()</code>	
<code>N</code>	Trigger timer number (1 to 4)

Details

The `trigger.timer[N].reset()` function resets the following attributes to their default values:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].enable`
- `trigger.timer[N].start.fractionalseconds`
- `trigger.timer[N].start.generate`
- `trigger.timer[N].start.seconds`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

<code>trigger.timer[1].reset()</code>	Resets the attributes associated with timer 1 to their default values.
---------------------------------------	--

Also see

- [trigger.timer\[N\].count](#) (on page 14-279)
- [trigger.timer\[N\].delay](#) (on page 14-281)
- [trigger.timer\[N\].delaylist](#) (on page 14-281)
- [trigger.timer\[N\].enable](#) (on page 14-282)
- [trigger.timer\[N\].start.generate](#) (on page 14-284)
- [trigger.timer\[N\].start.overrun](#) (on page 14-285)
- [trigger.timer\[N\].start.seconds](#) (on page 14-286)
- [trigger.timer\[N\].start.stimulus](#) (on page 14-286)

trigger.timer[N].start.fractionalseconds

This attribute configures the fractional seconds of an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	0

Usage

```
time = trigger.timer[N].start.fractionalseconds
trigger.timer[N].start.fractionalseconds = time
```

time	The time in fractional seconds (0 to <1 s)
<i>N</i>	Trigger timer number (1 to 4)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

```
trigger.timer[1].start.fractionalseconds = 0.4 Set the trigger timer to start in 0.4 s.
```

Also see

[trigger.timer\[N\].start.generate](#) (on page 14-284)

trigger.timer[N].start.generate

This attribute specifies when timer events are generated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].start.generate
trigger.timer[N].start.generate = state
```

state	Generate a timer event when the timer delay elapses: trigger.OFF Generate a timer event when the timer starts and when the delay elapses: trigger.ON
<i>N</i>	Trigger timer number (1 to 4)

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event trigger.EVENT_TIMER*N*.

Example

```
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY8
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON
```

Reset trigger timer 4.
Set trigger timer 4 to have a 0.5 s delay.
Set the stimulus for trigger timer 4 to be the notify 8 event.
Set the timer event to occur when the timer delay elapses.
Set the trigger timer 4 count to 20.
Enable trigger timer 4.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-283)

trigger.timer[N].start.overrun

This attribute indicates if an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Trigger timer <i>N</i> reset	Not applicable	Not applicable

Usage

```
state = trigger.timer[N].start.overrun
```

state	The trigger overrun state (<i>true</i> or <i>false</i>)
<i>N</i>	Trigger timer number (1 to 4)

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

```
print(trigger.timer[1].start.overrun)
```

If an event was ignored, the output is *true*.
If the event was not ignored, the output is *false*.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-283)

trigger.timer[N].start.seconds

This attribute configures the seconds of an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	0 (0 s)

Usage

```
time = trigger.timer[N].start.seconds
trigger.timer[N].start.seconds = time
```

<i>time</i>	The time: 0 s to 2,147,483,647 s
<i>N</i>	Trigger timer number (1 to 4)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

```
trigger.timer[1].start.seconds = localnode.gettime() + 30
trigger.timer[1].enable = trigger.ON
```

Set the trigger timer to start 30 s from the time when the timer is enabled.

Also see

None

trigger.timer[N].start.stimulus

This attribute describes the event that starts the trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.timer[N].start.stimulus
trigger.timer[N].start.stimulus = event
```

<i>event</i>	The event that starts the trigger timer; see Details
<i>N</i>	Trigger timer number (1 to 4)

Details

Set the stimulus to any trigger event to start the timer when that event occurs.

Set the stimulus to none to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

Trigger events	
Event description	Event constant
Trigger event blender N (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
A command interface trigger:	trigger.EVENT_COMMAND
<ul style="list-style-type: none"> ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command device_trigger 	
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line N (1 to 6)	trigger.EVENT_DIGION
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Appropriate LXI trigger packet is received on LAN trigger object N (1 to 8)	trigger.EVENT_LANN
No trigger event	trigger.EVENT_NONE
Notify trigger block N (1 to 8) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
Source limit condition occurs	trigger.EVENT_SOURCE_LIMIT
Trigger timer N (1 to 4) expired	trigger.EVENT_TIMERN
Line edge detected on TSP-Link synchronization line N (1 to 3)	trigger.EVENT_TSPLINKN

Example

```

digio.line[3].mode = digio.MODE_TRIGGER_IN
trigger.timer[1].delay = 3e-3
trigger.timer[1].start.stimulus = trigger.EVENT_DIGIO3
Set digital I/O line 3 to be a trigger input.
Set timer 1 to delay for 3 ms.
Set timer 1 to start the timer when an event is detected on digital I/O line 3.

```

Also see

None

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.timer[N].wait(timeout)
```

triggered	Trigger detection indication
N	Trigger timer number (1 to 4)
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

triggered = trigger.timer[3].wait(10) print(triggered)	Waits up to 10 s for a trigger on timer 3. If <code>false</code> is returned, no trigger was detected during the 10 s timeout. If <code>true</code> is returned, a trigger was detected.
---	--

Also see

[trigger.timer\[N\].clear\(\)](#) (on page 14-279)

trigger.tsplinkin[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

trigger.tsplinkin[N].clear()	
N	The trigger line (1 to 3) to clear

Details

The trigger event detector enters the detected state when an event is detected. When this command is sent, the instrument:

- Clears the trigger event detector
- Discards the history of the trigger line
- Clears the `trigger.tsplinkin[N].overrun` attribute

Example

```
tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[2].clear()
Clears the trigger event on TSP-Link line 2.
```

Also see

[trigger.tsplinkin\[N\].overrun](#) (on page 14-290)
[tsplink.line\[N\].mode](#) (on page 14-297)

trigger.tsplinkin[N].edge

This attribute indicates which trigger edge controls the trigger event detector for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.tsplinkin[N].edge
trigger.tsplinkin[N].edge = detectedEdge
```

<i>detectedEdge</i>	The trigger mode: <ul style="list-style-type: none">▪ Detect falling-edge triggers as inputs: trigger.EDGE_FALLING▪ Detect rising-edge triggers as inputs: trigger.EDGE_RISING▪ Detect either falling or rising-edge triggers as inputs: trigger.EDGE_EITHER
<i>N</i>	The trigger line (1 to 3)

Details

When the edge is detected, the instrument asserts a TTL-low pulse for the output.

The output state of the I/O line is controlled by the trigger logic. The user-specified output state of the line is ignored.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[3].edge = trigger.EDGE_RISING
Sets synchronization line 3 to detect rising edge triggers as input.
```

Also see

[digio.writeport\(\)](#) (on page 14-64)
[tsplink.line\[N\].mode](#) (on page 14-297)
[tsplink.line\[N\].reset\(\)](#) (on page 14-298)

trigger.tsplinkin[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup TSP-Link line <i>N</i> clear	Not applicable	Not applicable

Usage

```
overrun = trigger.tsplinkin[N].overrun
```

overrun	Trigger overrun state
<i>N</i>	The trigger line (1 to 3)

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself.

It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun.

Example

```
print(trigger.tsplinkin[1].overrun)
```

If an event on line 1 was ignored, displays true; if no additional event occurred, displays false.

Also see

None

trigger.tsplinkin[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.tsplinkin[N].wait(timeout)
```

triggered	Trigger detection indication; set to one of the following values: <ul style="list-style-type: none"> ■ true: A trigger is detected during the timeout period ■ false: A trigger is not detected during the timeout period
<i>N</i>	The trigger line (1 to 3)
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events are detected since the last time this command or `trigger.tsplinkin[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
triggered = trigger.tsplinkin[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on TSP-Link line 3.
If `false` is returned, no trigger was detected during the 10-s timeout.
If `true` is returned, a trigger was detected.

Also see

[trigger.tsplinkin\[N\].clear\(\)](#) (on page 14-288)
[tsplink.line\[N\].mode](#) (on page 14-297)

trigger.tsplinkout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkout[N].assert()
N           The trigger line (1 to 3)
```

Details

Initiates a trigger event and does not wait for completion. The set pulse width determines how long the trigger is asserted.

Example

```
tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[2].assert()
Asserts trigger on trigger line 2.
```

Also see

[tsplink.line\[N\].mode](#) (on page 14-297)

trigger.tsplinkout[N].logic

This attribute defines the trigger output with output logic for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.tsplinkout[N].logic
trigger.tsplinkout[N].logic = logicType
```

<i>logicType</i>	The output logic of the trigger generator: <ul style="list-style-type: none"> ▪ Assert a TTL-high pulse for output: trigger.LOGIC_POSITIVE ▪ Assert a TTL-low pulse for output: trigger.LOGIC_NEGATIVE
<i>N</i>	The trigger line (1 to 3)

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

The output state of the digital I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].logic = trigger.LOGIC_POSITIVE
Sets the trigger logic for synchronization line 3 to output a positive pulse.
```

Also see

[trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-291)
[tsplink.line\[N\].mode](#) (on page 14-297)

trigger.tsplinkout[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	10e-6 (10 µs)

Usage

```
width = trigger.tsplinkout[N].pulsewidth
trigger.tsplinkout[N].pulsewidth = width
```

<i>width</i>	The pulse width (0.0 to 100 ks)
<i>N</i>	The trigger line (1 to 3)

Details

Setting the pulse width to 0 asserts the trigger indefinitely.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN  
trigger.tsplinkout[3].pulsewidth = 20e-6
```

Sets pulse width for trigger line 3 to 20 µs.

Also see

- [trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-291)
- [trigger.tsplinkout\[N\].release\(\)](#) (on page 14-293)
- [tsplink.line\[N\].mode](#) (on page 14-297)

trigger.tsplinkout[N].release()

This function releases a latched trigger on the given TSP-Link trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkout[N].release()  
N The trigger line (1 to 3)
```

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN  
trigger.tsplinkout[3].release()  
Releases trigger line 3.
```

Also see

- [trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-291)
- [tsplink.line\[N\].mode](#) (on page 14-297)

trigger.tsplinkout[N].stimulus

This attribute specifies the event that causes the synchronization line to assert a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.tsplinkout[N].stimulus
trigger.tsplinkout[N].stimulus = event
```

event	The event identifier for the triggering event (see Details)
<i>N</i>	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to `trigger.EVENT_NONE`.

Do not use this attribute when triggering under script control. Use `trigger.tsplinkout[N].assert()` instead.

The *event* parameters that you can use are described in the table below.

Trigger events	
Event description	Event constant
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
A command interface trigger: <ul style="list-style-type: none"> ■ Any remote interface: *TRG ■ GPIB only: GET bus command ■ USB only: A USBTMC TRIGGER message ■ VXI-11: VXI-11 command device_trigger 	<code>trigger.EVENT_COMMAND</code>
Digital input line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
No trigger event	<code>trigger.EVENT_NONE</code>
Notify trigger block <i>N</i> (1 to 8) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
Source limit condition occurs	<code>trigger.EVENT_SOURCE_LIMIT</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>

Example

```
print(trigger.tsplinkout[3].stimulus)
```

Outputs the event that will start action on TSP-Link trigger line 3.

Also see

[trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-291)

[tsplink.line\[N\].reset\(\)](#) (on page 14-298)

trigger.wait()

This function waits for a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
triggered = trigger.wait(timeout)
```

triggered	A trigger was detected during the timeout period: true No triggers were detected during the timeout period: false
timeout	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

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

If one or more of these trigger events were previously detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.wait(10)
print(triggered)
```

Waits up to 10 s for a trigger.
If *false* is returned, no trigger was detected during the 10 s timeout.
If *true* is returned, a trigger was detected.

Also see

[trigger.clear\(\)](#) (on page 14-222)

tsplink.group

This attribute contains the group number of a TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

groupNumber	The group number of the TSP-Link node (0 to 64)
-------------	---

Details

To remove the node from all groups, set the attribute value to 0.

When the node is turned off, the group number for that node changes to 0.

The master node can be assigned to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

Example

tsplink.group = 3	Assign the instrument to TSP-Link group number 3.
-------------------	---

Also see

[Using groups to manage nodes on a TSP-Link system](#) (on page 9-7)

tsplink.initialize()

This function initializes all instruments and enclosures in the TSP-Link system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.initialize()
tsplink.initialize()
tsplink.initialize(expectedNodes)
```

nodesFound	The number of nodes found on the system, including the node on which the command is running
expectedNodes	The number of nodes expected on the system (1 to 32)

Details

This function regenerates the system configuration information regarding the nodes connected to the TSP-Link system. You must initialize the system after making configuration changes.

You need to initialize the system after you:

- Turn off power or reboot any instrument in the system
- Change node numbers on any instrument in the system
- Rearrange or disconnect the TSP-Link cable connections between instruments

If the only node on the TSP-Link network is the one running the command and *expectedNodes* is not provided, this function generates an error event. If you set *expectedNodes* to 1, the node is initialized.

If you include *expectedNodes*, if *nodesFound* is less than *expectedNodes*, an error event is generated.

NOTE

If any TSP-Link cabled node is powered off, initialize will fail.

Example

```
nodesFound = tsplink.initialize(2)
print("Nodes found = " .. nodesFound)

Perform a TSP-Link initialization and indicate how many nodes are found.
Example output if two nodes are found:
Nodes found = 2

Example output if fewer nodes are found and if localnode.showevents = 7:
1219, TSP-Link found fewer nodes than expected
Nodes found = 1
```

Also see

[Initializing the TSP-Link system](#) (on page 9-4)
[localnode.showevents](#) (on page 14-107)
[tsplink.node](#) (on page 14-300)
[tsplink.state](#) (on page 14-301)

tsplink.line[N].mode

This attribute defines the trigger operation of a TSP-Link line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	tsplink.MODE_DIGITAL_OPEN_DRAIN

Usage

```
mode = tsplink.line[N].mode
tsplink.line[N].mode = mode
```

mode	The trigger mode; see Details
<i>N</i>	The trigger line (1 to 3)

Details

This command defines whether or not the line is used as a digital or trigger control line and if it is an input or output.

The line mode can be set to the following options:

- TSP-Link digital open drain line: `tsplink.MODE_DIGITAL_OPEN_DRAIN`
- TSP-Link trigger open drain line: `tsplink.MODE_TRIGGER_OPEN_DRAIN`
- TSP-Link trigger synchronous master: `tsplink.MODE_SYNCHRONOUS_MASTER`
- TSP-Link trigger synchronous acceptor: `tsplink.MODE_SYNCHRONOUS_ACCEPTOR`

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
```

Sets the trigger mode for synchronization line 3 as a trigger open drain line.

Also see

- [trigger.tsplinkin\[N\].edge](#) (on page 14-289)
[trigger.tsplinkout\[N\].logic](#) (on page 14-292)

tsplink.line[N].reset()

This function resets some of the TSP-Link trigger attributes to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.line[N].reset()
```

<code>N</code>	The trigger line (1 to 3)
----------------	---------------------------

Details

The `tsplink.line[N].reset()` function resets the following attributes to their default values:

- `tsplink.line[N].mode`
- `trigger.tsplinkin[N].edge`
- `trigger.tsplinkout[N].logic`
- `trigger.tsplinkout[N].pulsewidth`
- `trigger.tsplinkout[N].stimulus`

This also clears `trigger.tsplinkin[N].overrun`.

Example

```
tsplink.line[3].reset()
```

Resets TSP-Link trigger line 3 attributes to default values.

Also see

[trigger.tsplinkin\[N\].edge](#) (on page 14-289)
[trigger.tsplinkin\[N\].overrun](#) (on page 14-290)
[trigger.tsplinkout\[N\].logic](#) (on page 14-292)
[trigger.tsplinkout\[N\].pulsewidth](#) (on page 14-292)
[trigger.tsplinkout\[N\].stimulus](#) (on page 14-294)
[tsplink.line\[N\].mode](#) (on page 14-297)

tsplink.line[N].state

This attribute reads or writes the digital state of a TSP-Link synchronization line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	tsplink.STATE_HIGH

Usage

```
lineState = tsplink.line[N].state
tsplink.line[N].state = lineState
```

lineState	The state of the synchronization line: <ul style="list-style-type: none"> ▪ Low: tsplink.STATE_LOW ▪ High: tsplink.STATE_HIGH
N	The trigger line (1 to 3)

Details

Use `tsplink.writeport()` to write to all TSP-Link synchronization lines.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

```
lineState = tsplink.line[3].state
print(lineState)
```

Assume line 3 is set high, and then the state is read.
Output:
`tsplink.STATE_HIGH`

Also see

[tsplink.line\[N\].mode](#) (on page 14-297)
[tsplink.writeport\(\)](#) (on page 14-302)

tsplink.master

This attribute reads the node number assigned to the master node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
masterNodeNumber = tsplink.master
```

masterNodeNumber	The node number of the master node (1 to 63)
------------------	--

Details

This attribute returns the node number of the master in a set of instruments connected using TSP-Link.

Example

LinkMaster = tsplink.master	Store the TSP-Link master node number in a variable called LinkMaster.
-----------------------------	--

Also see

[tsplink.initialize\(\)](#) (on page 14-296)

tsplink.node

This attribute defines the node number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	2

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

nodeNumber	The node number of the instrument or enclosure (1 to 63)
------------	--

Details

This command sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.reset()` from an earlier TSP-Link instrument or `tsplink.initialize()` is executed on any node in the system.

Each node connected to the TSP-Link system must be assigned a different node number.

Example

tsplink.node = 3	Sets the TSP-Link node for this instrument to number 3.
------------------	---

Also see

[tsplink.initialize\(\)](#) (on page 14-296)
[tsplink.state](#) (on page 14-301)

tsplink.readport()

This function reads the TSP-Link trigger lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readport()
data           Numeric value that indicates which lines are set
```

Details

The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and the value of bit 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. This indicates that line 2 is high (1), and that the other two lines are low (0).

Example

<pre>data = tsplink.readport() print(data)</pre>	Reads state of all three TSP-Link lines. Assuming line 2 is set high, the output is: 2.000000e+00 (binary 010) The format of the output may vary depending on the ASCII precision setting.
--	--

Also see

[Triggering using TSP-Link trigger lines](#) (on page 9-6)
[tsplink.line\[N\].state](#) (on page 14-299)
[tsplink.writeport\(\)](#) (on page 14-302)

tsplink.state

This attribute describes the TSP-Link online state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
state = tsplink.state
state           TSP-Link state (online or offline)
```

Details

When the instrument power is first turned on, the state is `offline`. After `tsplink.initialize()` or `tsplink.reset()` is successful, the state is `online`.

Example

<pre>state = tsplink.state print(state)</pre>	Read the state of the TSP-Link system. If it is online, the output is: online
---	--

Also see

[tsplink.initialize\(\)](#) (on page 14-296)
[tsplink.node](#) (on page 14-300)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writeport(data)
data           Value to write to the port (0 to 7)
```

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other two lines are set low (0).

The `reset()` function does not affect the present states of the trigger lines.

Example

<code>tsplink.writeport(3)</code>	Sets the synchronization lines 1 and 2 high (binary 011).
-----------------------------------	---

Also see

[tsplink.line\[N\].state](#) (on page 14-299)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.clear(connectionID)
connectionID  The connection ID returned from tspnet.connect()
```

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

<code>tspnet.write(testdevice, "print([[hello]])") print(tspnet.readavailable(testdevice))</code>	Write data to a device, then print how much is available. Output: <code>6.00000e+00</code>
<code>tspnet.clear(testdevice) print(tspnet.readavailable(testdevice))</code>	Clear data and print how much data is available again. Output: <code>0.00000e+00</code>

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.readavailable\(\)](#) (on page 14-308)
[tspnet.write\(\)](#) (on page 14-313)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
connectionID = tspnet.connect("ipAddress")
connectionID = tspnet.connect("ipAddress", portNumber, "initString")
```

<code>connectionID</code>	The connection ID to be used as a handle in all other <code>tspnet</code> function calls
<code>ipAddress</code>	IP address to which to connect in a string
<code>portNumber</code>	Port number (default 5025)
<code>initString</code>	Initialization string to send to <code>ipAddress</code>

Details

This command connects a device to another device through the LAN interface. If the `portNumber` is 23, the interface uses the Telnet protocol and sets appropriate termination characters to communicate with the device.

If a `portNumber` and `initString` are provided, it is assumed that the remote device is not TSP-enabled. The 2450 does not perform any extra processing, prompt handling, error handling, or sending of commands. In addition, the `tspnet.tsp.*` commands cannot be used on devices that are not TSP-enabled.

If neither a `portNumber` nor an `initString` is provided, the remote device is assumed to be a Keithley Instruments TSP-enabled device. Depending on the state of the `tspnet.tsp.abortonconnect` attribute, the 2450 sends an abort command to the remote device on connection.

The 2450 also enables TSP prompts on the remote device and event management. The 2450 places remote errors and events from the TSP-enabled device in its own event queue and prefixes these events with `Remote Error`, followed by an event description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show events by changing `localnode.showerrors` or `localnode.showevents` on the remote TSP-enabled device. If you do this, subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

```
instrumentID = tspnet.connect("192.0.2.1")
if instrumentID then
    -- Use instrumentID as needed here
    tspnet.disconnect(instrumentID)
end
```

Connect to a TSP-enabled device.

Example 2

```
instrumentID = tspnet.connect("192.0.2.1", 1394, "*rst\r\n")
if instrumentID then
    -- Use instrumentID as needed here
    tspnet.disconnect(instrumentID)
end
```

Connect to a device that is not TSP-enabled.

Also see

[localnode.prompts](#) (on page 14-104)
[localnode.showevents](#) (on page 14-107)
[tspnet.tsp.abortonconnect](#) (on page 14-311)
[tspnet.disconnect\(\)](#) (on page 14-304)

tspnet.disconnect()

This function disconnects a specified TSP-Net session.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.disconnect(connectionID)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
---------------------	---

Details

This function disconnects the two devices by closing the connection. The `connectionID` is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

```
testID = tspnet.connect("192.0.2.0")
-- Use the connection
tspnet.disconnect(testID)
```

Create a TSP-Net session.

Close the session.

Also see

[tspnet.connect\(\)](#) (on page 14-303)

tspnet.execute()

This function sends a command string to the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.execute("connectionID", "commandString")
value1 = tspnet.execute("connectionID", "commandString", formatString)
value1, value2 = tspnet.execute("connectionID", "commandString", formatString)
value1, ..., valueN = tspnet.execute("connectionID", "commandString", formatString)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>commandString</i>	The command to send to the remote device
<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valueN</i>	The <i>N</i> th value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
<i>formatString</i>	Format string for the output

Details

This command sends a command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (`tspnet.termination()`). You can also specify a format string, which causes the command to wait for a response from the remote instrument. The 2450 decodes the response message according to the format specified in the format string and returns the message as return values from the function (see `tspnet.read()` for format specifiers).

When this command is sent to a TSP-enabled instrument, the 2450 suspends operation until a timeout error is generated or until the instrument responds. The TSP prompt from the remote instrument is read and discarded. The 2450 places any remotely generated errors and events into its event queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

<code>tspnet.execute(instrumentID, "runScript()")</code>
Command the remote device to run a script named <code>runScript</code> .

Example 2

```
tspnet.timeout = 5
id_instr = tspnet.connect("192.0.2.23", 23, "*rst\r\n")
tspnet.termination(id_instr, tspnet.TERM_CRLF)
tspnet.execute(id_instr, "*idn?")
print("tspnet.execute returns:", tspnet.read(id_instr))
Print the *idn? string from the remote device.
```

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.read\(\)](#) (on page 14-307)
[tspnet.termination\(\)](#) (on page 14-309)
[tspnet.write\(\)](#) (on page 14-313)

tspnet.idn()

This function retrieves the response of the remote device to *IDN?.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
idnString = tspnet.idn(connectionID)
```

<i>idnString</i>	The returned *IDN? string
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

Details

This function retrieves the response of the remote device to *IDN?.

Example

```
deviceID = tspnet.connect("192.0.2.1")
print(tspnet.idn(deviceID))
tspnet.disconnect(deviceID)
```

Assume the instrument is at IP address 192.0.2.1.
The output that is produced when you connect to the instrument and read the identification string may appear as:
KEITHLEY INSTRUMENTS,MODEL
2450,00000170,1.1.0s

Also see

[tspnet.connect\(\)](#) (on page 14-303)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valueN = tspnet.read(connectionID, formatString)
```

value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The nth value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
connectionID	The connection ID returned from <code>tspnet.connect()</code>
formatString	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following specifiers:

%[width]s	Read data until the specified length
%[max width]t	Read data until the specified length or until punctuation is found, whichever comes first
%[max width]n	Read data until a newline or carriage return
%d	Read a number (delimited by punctuation)

A maximum of 10 format specifiers can be used for a maximum of 10 return values.

If `formatString` is not provided, the command returns a string that contains the data until a new line is reached. If no data is available, the 2450 pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When the 2450 reads from a TSP-enabled remote instrument, the 2450 removes Test Script Processor (TSP®) prompts and places any errors or events it receives from the remote instrument into its own event queue. The 2450 prefacing events and errors from the remote device with `Remote Error`, followed by the event number and description.

Example

```
tspnet.write(deviceID, "*idn?\r\n")
print("write/read returns:", tspnet.read(deviceID))

Send the "*idn?\r\n" message to the instrument connected as deviceID.
Display the response that is read from deviceID (based on the *idn? message).
```

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.readavailable\(\)](#) (on page 14-308)
[tspnet.timeout](#) (on page 14-310)
[tspnet.write\(\)](#) (on page 14-313)

tspnet.readavailable()

This function checks to see if data is available from the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
bytesAvailable = tspnet.readavailable(connectionID)
```

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from <code>tspnet.connect()</code>

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")
repeat bytes = tspnet.readavailable(ID) until bytes > 0
print(tspnet.read(ID))
tspnet.disconnect(ID)
Send commands that will create data.
Wait for data to be available.
```

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.read\(\)](#) (on page 14-307)

tspnet.reset()

This function disconnects all TSP-Net sessions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.reset()
```

Details

This command disconnects all remote instruments connected through TSP-Net. For TSP-enabled devices, this causes any commands or scripts running remotely to be terminated.

Also see

None

tspnet.termination()

This function sets the device line termination sequence.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

type	The termination type: <ul style="list-style-type: none"> ■ tspnet.TERM_LF ■ tspnet.TERM_CR ■ tspnet.TERM_CRLF ■ tspnet.TERM_LFCR
connectionID	The connection ID returned from <code>tspnet.connect()</code>
termSequence	The termination sequence: <ul style="list-style-type: none"> ■ tspnet.TERM_LF ■ tspnet.TERM_CR ■ tspnet.TERM_CRLF ■ tspnet.TERM_LFCR

Details

This function sets and gets the termination character sequence that is used to indicate the end of a line for a TSP-Net connection.

Using the `termSequence` parameter sets the termination sequence. The present termination sequence is always returned.

For the `termSequence` parameter, use the same values listed in the table above for type. There are four possible combinations, all of which are made up of line feeds (LF or 0x10) and carriage returns (CR or 0x13). For TSP-enabled devices, the default is `tspnet.TERM_LF`. For devices that are not TSP-enabled, the default is `tspnet.TERM_CRLF`.

Example

<pre>deviceID = tspnet.connect("192.0.2.1") if deviceID then tspnet.termination(deviceID, tspnet.TERM_LF) end</pre>	Sets termination type for IP address 192.0.2.1 to TERM_LF.
---	---

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.disconnect\(\)](#) (on page 14-304)

tspnet.timeout

This attribute sets the timeout value for the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	20.0 (20 s)

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

value	The timeout duration in seconds (1 ms to 30.0 s)
-------	--

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands will wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution, but is only accurate to the nearest 10 ms.

Example

```
tspnet.timeout = 2.0
```

Sets the timeout duration to 2 s.

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.execute\(\)](#) (on page 14-305)
[tspnet.read\(\)](#) (on page 14-307)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.abort(connectionID)
```

connectionID	Integer value used as a handle for other <code>tspnet</code> commands
--------------	---

Details

This function is appropriate only for TSP-enabled instruments.

Sends an abort command to the remote instrument.

Example

```
tspnet.tsp.abort(testConnection)
```

Stops remote instrument execution on testConnection.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	<ul style="list-style-type: none"> ■ Enable: 1 ■ Disable: 0
-------	---

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument will not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

```
tspnet.tsp.abortonconnect = 0
```

Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

[tspnet.connect\(\)](#) (on page 14-303)

tspnet.tsp.rbtabcopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
table = tspnet.tsp.rbtabcopy(connectionID, "name")
table = tspnet.tsp.rbtabcopy(connectionID, "name", startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other <code>tspnet</code> commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The `startIndex` and `endIndex` parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

```
times =
    tspnet.tsp.rbtabcopy(testTspdevice,
    "testRemotebuffername.timestamps", 1, 3)
print(times)
```

Copy the specified timestamps table for items 1 through 3, then display the table.
Example output:
01/01/2015
10:10:10.0000013,01/01/2015
10:10:10.0000233,01/01/2015
10:10:10.0000576

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.runscript(connectionID, "name", "script")
```

connectionID	Integer value used as an identifier for other <code>tspnet</code> commands
--------------	--

<i>name</i>	The name that is assigned to the script
<i>script</i>	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions later.

Example

```
tspnet.tsp.runscript(myConnection, "myTest",
"print([[start]]) for d = 1, 10 do print([[work]]) end print([[end]]))")
Load and run a script entitled myTest on the TSP-enabled instrument connected with myConnection.
```

Also see

[tspnet.execute\(\)](#) (on page 14-305)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.write(connectionID, "inputString")
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>inputString</i>	The string to be written

Details

The `tspnet.write()` function sends *inputString* to the remote instrument. It does not wait for command completion on the remote instrument.

The 2450 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

<code>tspnet.write(myID, "runscript()\r\n")</code>	Commands the remote instrument to execute a command or script named <code>runscript()</code> on a remote device identified in the system as <code>myID</code> .
--	---

Also see

[tspnet.connect\(\)](#) (on page 14-303)
[tspnet.read\(\)](#) (on page 14-307)

upgrade.previous()

This function returns to a previous version of the 2450 firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`upgrade.previous()`

Details

This function allows you to revert to an earlier version of the firmware.

When you send this function, the instrument searches the USB flash drive in the front-panel USB port for an upgrade file. If the file is found, the instrument performs the upgrade. An error is returned if an upgrade file is not found.

NOTE

Use this command with caution. Make sure your instrument can support the earlier version and that there are no compatibility issues. Check with Keithley Instruments before using this command if you have questions.

Also see

[Upgrading the firmware](#) (on page 10-3)
[upgrade.unit\(\)](#) (on page 14-314)

upgrade.unit()

This function upgrades the 2450 firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`upgrade.unit()`

Details

When `upgrade.unit()` is used, the firmware is only loaded if the version of the firmware is newer than the existing version. If the version is older or at the same revision level, it is not upgraded.

When you send this function, the instrument searches the USB flash drive in the front-panel USB port for an upgrade file. If the file is found, the instrument verifies that the file is a newer version. If the version is older or at the same revision level, it is not upgraded, although it does request a reboot. If it is a newer version, the instrument performs the upgrade. An error event message is returned if no upgrade file is found.

Also see

- [upgrade.previous\(\)](#) (on page 14-314)
- [Upgrading the firmware](#) (on page 10-3)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.add( "name" , "value" )
```

<code>name</code>	The name of the string; the key of the key-value pair
<code>value</code>	The string to associate with <code>name</code> ; the value of the key-value pair

Details

This function associates the string `value` with the string `name` and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the `value` associated with the specified `name`.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

Example

```
userstring.add( "assetnumber" , "236" )
userstring.add( "product" , "Widgets" )
userstring.add( "contact" , "John Doe" )
for name in userstring.catalog() do
    print(name .. " = " ..
          userstring.get(name))
end
```

Stores user-defined strings in nonvolatile memory and recalls them from the instrument using a for loop.
 Example output:
 assetnumber = 236
 contact = John Doe
 product = Widgets

Also see

- [userstring.catalog\(\)](#) (on page 14-316)
- [userstring.delete\(\)](#) (on page 14-316)
- [userstring.get\(\)](#) (on page 14-317)

userstring.catalog()

This function creates an iterator for the user-defined string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do body end
```

name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The catalog provides access for user-defined string pairs, allowing you to manipulate all the key-value pairs in nonvolatile memory. The entries are enumerated in no particular order.

Example 1

```
for name in userstring.catalog()
    userstring.delete(name)
end
```

Deletes all user-defined strings in nonvolatile memory.

Example 2

```
userstring.add( "assetnumber", "236" )
userstring.add( "product", "Widgets" )
userstring.add( "contact", "John Doe" )
for name in userstring.catalog() do
    print(name .. " = " ..
          userstring.get(name))
end
```

Prints all userstring key-value pairs.
Output:
product = Widgets
assetnumber = 236
contact = John Doe
Notice the key-value pairs are not listed in the order they were added.

Also see

[userstring.add\(\)](#) (on page 14-315)
[userstring.delete\(\)](#) (on page 14-316)
[userstring.get\(\)](#) (on page 14-317)

userstring.delete()

This function deletes a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.delete( "name" )
```

name	The name (key) of the key-value pair of the user-defined string to delete
------	---

Details

This function deletes the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.delete( "assetnumber" )
userstring.delete( "product" )
userstring.delete( "contact" )
```

Deletes the user-defined strings associated with the assetnumber, product, and contact names.

Also see

[userstring.add\(\)](#) (on page 14-315)
[userstring.catalog\(\)](#) (on page 14-316)
[userstring.get\(\)](#) (on page 14-317)

userstring.get()

This function retrieves a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value = userstring.get( "name" )

value          The value of the user-defined string key-value pair
name           The name (key) of the user-defined string
```

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.add( "assetnumber" , "236" )
value = userstring.get( "assetnumber" )
print(value)

Create the user-defined string assetnumber, set to a value of 236.
Read the value associated with the user-defined string named assetnumber.
Store it in a variable called value, then print the variable value.
Output:
236
```

Also see

[userstring.add\(\)](#) (on page 14-315)
[userstring.catalog\(\)](#) (on page 14-316)
[userstring.delete\(\)](#) (on page 14-316)

waitcomplete()

This function waits for all previously started overlapped commands to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
waitcomplete()
waitcomplete(group)
```

group	Specifies which TSP-Link group on which to wait
-------	---

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `waitcomplete()` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. This command is not needed for sequential commands.

A group number may only be specified when this node is the master node.

If no `group` is specified, the local group is used.

If zero (0) is specified for the `group`, this function waits for all nodes in the system.

NOTE

Any nodes that are not assigned to a group (group number is 0) are part of the master node's group.

Example 1

waitcomplete()	Waits for all nodes in the local group.
----------------	---

Example 2

waitcomplete(G)	Waits for all nodes in group G.
-----------------	---------------------------------

Example 3

waitcomplete(0)	Waits for all nodes on the TSP-Link network.
-----------------	--

Also see

None

Section 15

Common commands

In this section:

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

Introduction

This section describes the general remote interface commands and common commands. Note that although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the 2450 does not strictly conform to that standard.

The general remote interface commands are commands that have the same general meaning, regardless of the instrument you use them with (for example, DCL always clears the GPIB interface and returns it to a known state).

The common commands perform operations such as reset, wait-to-continue, and status.

Common commands always begin with an asterisk (*) and may include one or more parameters. The command keyword is separated from the first parameter by a blank space.

If you are using a SCPI remote interface, the commands can be combined. Use a semicolon (;) to separate multiple commands, as shown below:

```
*RST; *CLS; *ESE 32; *OPC?
```

Although the commands in this section are shown in uppercase, they are not case sensitive (you can use either uppercase or lowercase).

If you are using the TSP remote interface, each command must be sent in a separate message.

NOTE

If you are using the TSP remote interface, note that the common commands and general bus commands cannot be used in scripts.

*CLS

This command clears the event registers and queues.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*CLS

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It also clears the event log. It does not affect the Questionable Event Enable or Operation Event Enable registers.

This is the equivalent of sending the SCPI commands :STATUs:CLEar and :SYStem:CLEar or the TSP commands `status.clear()` and `eventlog.clear()`.

To reset all the bits of the Standard Event Enable Register, send the command:

*ESE 0

Also see

- [*ESE \(on page 15-2\)](#)
- [:STATus:PRESet \(on page 12-101\)](#)
- [status.preset\(\) \(on page 14-209\)](#)

*ESE

This command sets and queries bits in the Status Enable register of the Standard Event Register.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	See Details

Usage

*ESE <n>
*ESE?

<n> The value of the Status Enable register of the Standard Event Register (0 to 255)

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or the value of the bit as the <n> parameter.

If you are using TSP, you can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B2, set *standardRegister* to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

If you are using SCPI, you can only set the bit as a numeric value. When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-110) function.
1	2	Not used	Not used.
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

*ESE 129

*ESE 129 sets the Status Enable register of the Standard Event Register to binary 10000001, which enables the PON and OPC bits.

Also see

[*CLS](#) (on page 15-2)

[Standard Event Register](#) (on page 16-3)

[Status model](#) (on page 16-1)

*ESR?

This command reads and clears the contents of the Standard Event Status Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

*ESR?

Details

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-110) function.
1	2	Not used	Not used.
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

*ESR?

Example output:

128

Shows that the Standard Event Status Register contains binary 10000000, which indicates that the instrument was rebooted since the last time this register was read.

Also see

[Status model](#) (on page 16-1)

*IDN?

This command retrieves the identification string of the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

*IDN?

Details

The identification string includes the manufacturer, model number, serial number, and firmware revision of the instrument. The string is formatted as follows:

```
KEITHLEY INSTRUMENTS,MODEL nnnn,xxxxxxxx,yyyyyy
```

Where:

- nnnn is the model number
- xxxxxxxx is the serial number
- yyyy is the firmware revision level

Example

*IDN?	Output: KEITHLEY INSTRUMENTS,MODEL 2450,01234567,1.0.0i
-------	--

Also see

[System information](#) (on page 2-39)

*LANG

This command determines which command set is used by the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	SCPI

Usage

```
*LANG <commandSet>
*LANG?
```

<commandSet>	The command set to be used: <ul style="list-style-type: none"> ■ TSP ■ SCPI ■ SCPI2400
--------------	---

Details

The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.
- **SCPI 2400:** An instrument-specific language that allows you to run code developed for earlier Series 2400 instruments.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

Example

*LANG TSP *LANG?	Set the command set to TSP. Verify setting by sending the command set query. Output: TSP The TSP command set is in use.
---------------------	---

Also see

[Status model](#) (on page 16-1)

*OPC

This command sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

*OPC
*OPC?

Details

When *OPC is sent, the OPC bit (bit 0) in the Status Event Status Register is set after all pending command operations have been executed. After all programmed operations are complete, the instrument returns to idle, at which time all pending commands (including *OPC and *OPC?) are executed. After the last pending command is executed, the OPC bit is set or an ASCII "1" is placed in the Output Queue.

When the trigger model is executing, most sent commands are not executed. If a command cannot be processed, an error event message is generated in the event log.

Also see

[:INITiate\[:IMMediate\]](#) (on page 12-146)

[opc\(\)](#) (on page 14-110)

*RST

This command resets the instrument settings to their default values and clears the reading buffers.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*RST

Details

Returns the instrument to default settings, cancels all pending commands, and cancels the response to any previously received *OPC and *OPC? commands.

Also see

[reset\(\)](#) (on page 14-116)

*SRE

This command sets or clears the bits of the Service Request Enable Register.

Type	Affected by	Where saved	Default value
Command and query	:STATus:PRESet status.preset()	Not applicable	0

Usage

*SRE <n>
*SRE?

<n>

Clear the Status Request Enable Register: 0
Set the instrument for an SRQ interrupt: 32

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	status.MSB	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	status.EAV	An error or status message is present in the Error Queue.
3	8	status.QSB	An enabled event in the Questionable Status Register has occurred.
4	16	status.MAV	A response message is present in the Output Queue.
5	32	status.ESB	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	status.OSB	An enabled event in the Operation Status Register has occurred.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

*SRE 0	Clear the bits of the Status Request Enable Register.
--------	---

Also see

[Understanding bit settings \(on page 16-15\)](#)

*STB?

This command gets the status byte of the instrument without clearing the request service bit.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

*STB?

Details

This command is similar to a serial poll, but it is processed like any other instrument command.

The *STB? command returns the same result as a serial poll, but the master summary bit (MSB) is not cleared if a serial poll has occurred. The MSB is not cleared until all other bits feeding into the MSB are cleared.

Example

*STB?	Queries the status byte.
-------	--------------------------

Also see

None

*TRG

This command generates a trigger event from a remote command interface.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*TRG

Details

Use the *TRG command to generate a trigger event.

If you are using the SCPI command set, this command generates the COMMAND event. If you are using the TSP command set, this command generates the trigger.EVENT_COMMAND event. You can use this constant as the stimulus of any trigger object, which causes that trigger object to respond to the trigger events generated by *TRG. See [Using trigger events to start actions in the trigger model](#) (on page 8-54).

Also see

[:INITiate\[:IMMEDIATE\]](#) (on page 12-146)

*TST?

This command is accepted and returns 0. A self-test is not actually performed.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	0

Usage

*TST?

Also see

None

*WAI

This command postpones the execution of subsequent commands until all previous overlapped commands are finished.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*WAI

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The *WAI command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. The *WAI command is not needed for sequential commands. Typically, this command is sent after the initiate trigger model command.

Also see

- [:INITiate\[:IMMediate\]](#) (on page 12-146)
[waitcomplete\(\)](#) (on page 14-318)

Section 16

Status model

In this section:

Overview	16-1
Serial polling and SRQ	16-13
Programming enable registers	16-14
Reading the registers	16-14
Understanding bit settings.....	16-15
Clearing registers	16-16
Status model programming examples.....	16-17

Overview

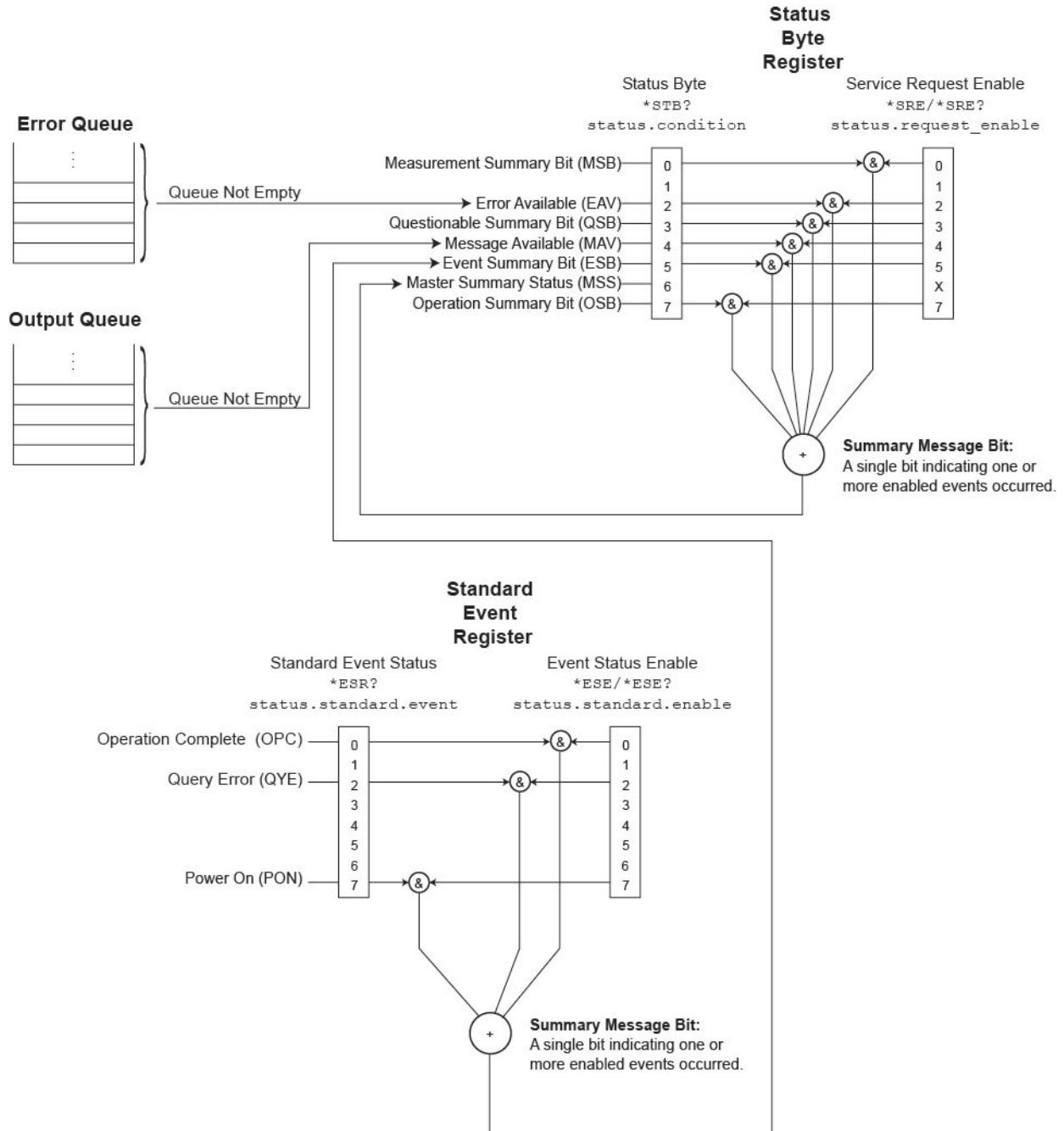
The status model consists of status register sets and queues. You can monitor the status model to view instrument events and configure the status model to control the events.

As you work with the status model, be aware that the result applies to the Status Byte Register. All the status register sets and queues flow into the Status Byte Register. Your test program can read this register to determine if a service request (SRQ) has occurred, and if so, which event caused it.

The Status Byte Register, register sets, and queues include:

- Standard Event Register
- Questionable Event Register
- Operation Event Register
- Output Queue
- Error Queue

The relationship between the Status Byte Register, Standard Event Register, event queue, and output queue is shown in the [Non-programmable status registers diagram](#) (on page 16-2). The relationship between the Status Byte Register, Questionable Event Register, and the Operation Event Register is shown in the [Programmable status registers diagram](#) (on page 16-5).

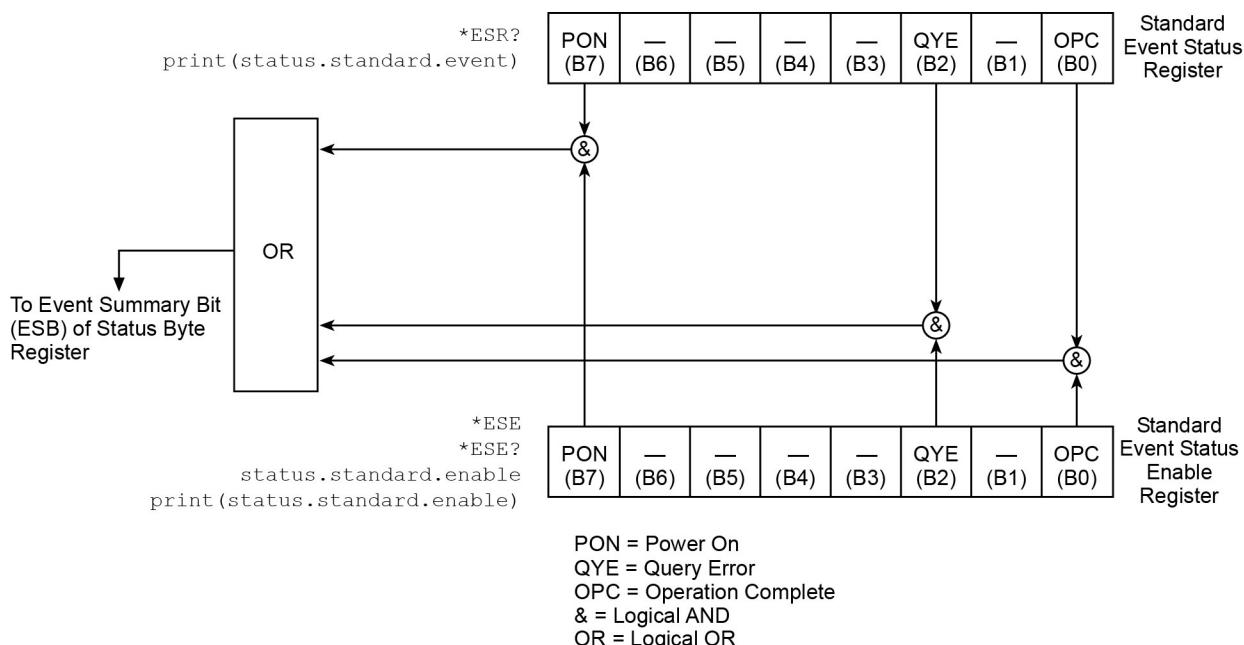
Figure 155: Non-programmable status registers diagram

Standard Event Register

The Standard Event Register set includes two 8-bit registers:

- **Standard Event Status Register:** Reports when a predefined event has occurred. The register latches the event and the corresponding bit remains set until it is cleared by a read.
- **Standard Event Status Enable Register:** You can enable or disable bits in this register. This allows the predefined event (from the Standard Event Status Register) to set the ESB of the Status Byte Register.

Figure 156: 2450 Standard Event Register



Bit	When set, indicates the following has occurred:
0	Operation complete: All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-6) command or TSP opc() (on page 14-110) function.
1	Not used.
2	Query error: Attempt to read data from an empty Output Queue.
3	Not used.
4	Not used.
5	Not used.
6	Not used.
7	Power-on: The instrument has been turned off and turned back on since the last time this register was read.

You can use the following commands to read and set bits in the Standard Event Register.

Description	SCPI command	TSP command
Read the Standard Event Status Register	*ESR? (on page 15-4)	status.standard.event (on page 14-215)
Set or read the OR bits in the Standard Event Status Enable Register	*ESE (on page 15-2) ESE?	status.standard.enable (on page 14-214)

Programmable status register sets

You can program the registers in the Questionable Event Register and Operation Event Register sets.

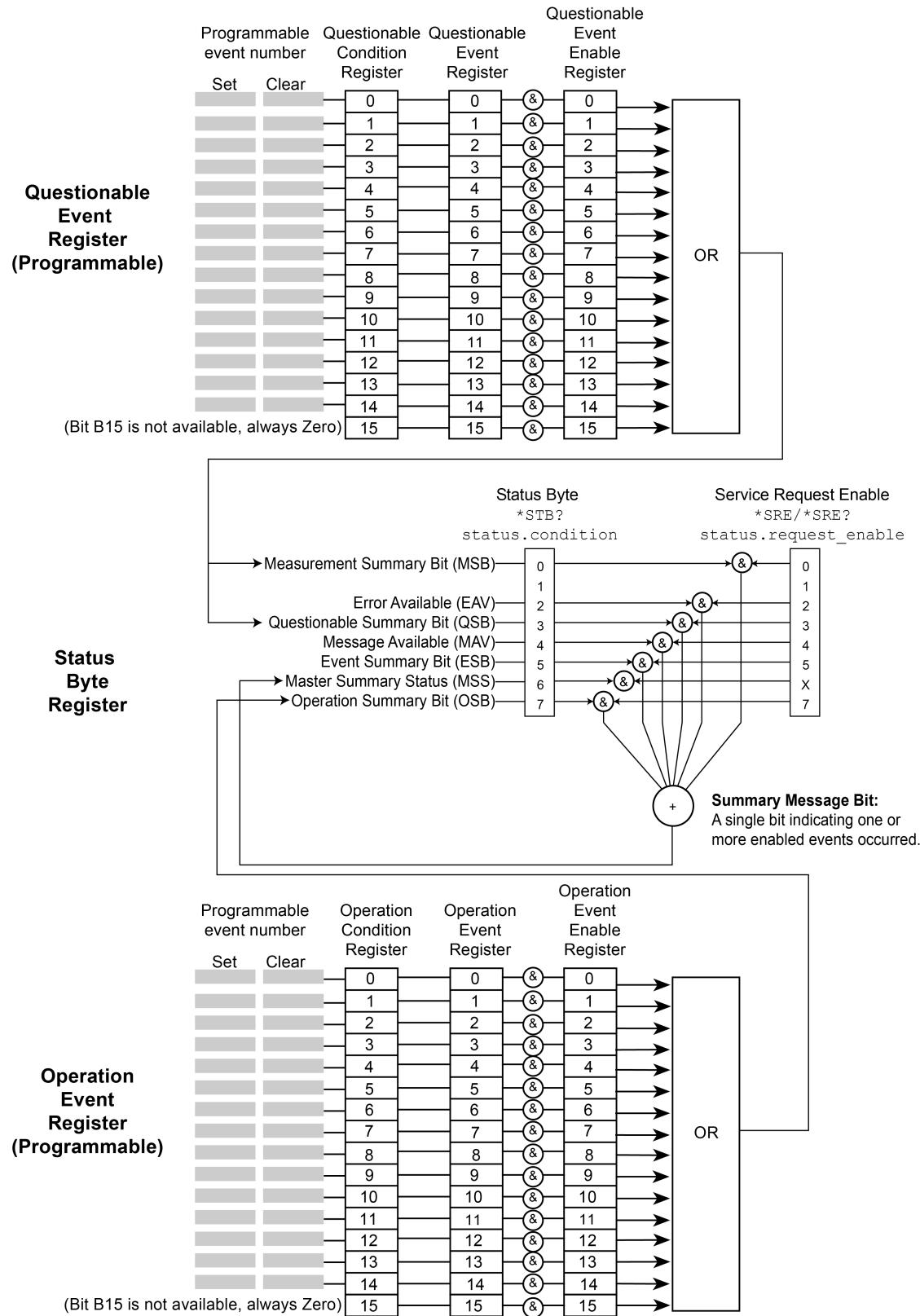
These event registers contain bits that identify the state of an instrument condition or event. They also contain bits that determine if those events are sent to the Status Byte Register. You can enable the events, which causes the associated bit to be set in the Status Byte Register.

The Questionable and Operation Event Registers are identical except that they set different bits in the Status Byte Register. The Questionable Event Registers set the MSB and QSM bits. The Operation Event Registers set the OSB bit.

Each 16-bit register set includes the following registers:

- **Condition:** A read-only register that is constantly updated to reflect the present operating conditions of the instrument. You can determine which events set or clear the bits.
- **Event:** A read-only register that sets a bit to 1 when an applicable event occurs. The bit remains at 1 until the register is reset. This register is reset when power is cycled, when a *CLS command is sent, or when the register is read. You can determine which events set the bits.
- **Event enable:** A read-write register that determines which events set the summary bit in the Status Byte Register. For example, if a bit is a 1 in the event register and the corresponding bit is a 1 in the Event Enable Register, bits in the Status Byte Register are set. If the event enable bit is set in the Questionable Event Registers, the event sets the MSB and QSM bits in the Status Byte Register. If the event enable bit is set in the Operation Event Registers, the event sets the OSB bit in the Status Byte Register.

When the instrument is powered on, all bits in the Questionable Event and Operation Event Registers are set to 0.

Figure 157: Programmable status registers diagram

Questionable Event Register

You can program the bits in the Questionable Event Register to be cleared or set when an event occurs.

When an enabled Questionable Event Register bit is set (because the enabled event occurs), the corresponding bit B0 (MSB) and Bit B3 (QSB) of the Status Byte Register is set. The corresponding Questionable Event Register Condition Register reflects the present status of the instrument, so it is set while the event occurs.

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see [Understanding bit settings](#) (on page 16-15).

You can use the following commands to read and set bits in the Questionable Event Register.

Description	SCPI command	TSP command
Read the Questionable Condition Register	:STATUs:QUEStionable:CONDition? (on page 12-102)	status.questionable.condition (on page 14-209)
Set or read the contents of the Questionable Event Enable Register	:STATUs:QUEStionable:ENABLE (on page 12-102)	status.questionable.enable (on page 14-210)
Read the Questionable Event Register	:STATUs:QUEStionable[:EVENT]? (on page 12-104)	status.questionable.event (on page 14-210)
Request the mapped set event and mapped clear event status for a bit in the Questionable Event Register	:STATUs:QUEStionable:MAP (on page 12-103)	status.questionable.getmap() (on page 14-211)
Map event to a bit in the Questionable Event Register	:STATUs:QUEStionable:MAP (on page 12-103)	status.questionable.setmap() (on page 14-212)

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register. See [Event numbers](#) (on page 16-9) for information about event numbers. You can use a copy of the following table to record settings for your instrument.

Bit	Decimal value	Set Event	Clear Event	When set, indicates the following has occurred:
0	1			
1	2			
2	4			
3	8			
4	16			
5	32			
6	64			
7	128			
8	256			
9	512			
10	1024			
11	2048			
12	4096			
13	8192			
14	16 384			

Operation Event Register

You can program the bits in the Operation Condition and Operation Event Status Registers to be cleared or set when an event occurs.

When an enabled Operation Event Register bit is set (because the enabled event occurs), the corresponding bit B7 (OSB) of the Status Byte Register is set. The corresponding Operation Event Register Condition Register reflects the present status of the instrument, so it will be set while the event occurs.

You can use the following commands to read and set bits in the Operation Event Register.

Description	SCPI command	TSP command
Read the Operation Condition Register	:STATus:OPERation:CONDition? (on page 12-99)	status.operation.condition (on page 14-205)
Set or read the contents of the Operation Event Enable Register	:STATus:OPERation:ENABLE (on page 12-99)	status.operation.enable (on page 14-206)
Read the Operation Event Register	:STATus:OPERation[:EVENT]? (on page 12-100)	status.operation.event (on page 14-206)
Request the mapped set event and mapped clear event status for a bit in the Operation Event Registers	:STATus:OPERation:MAP (on page 12-100)	status.operation.getmap() (on page 14-207)
Map events to bit in the Operation Event Register	:STATus:OPERation:MAP (on page 12-100)	status.operation.setmap() (on page 14-208)

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register. See [Event numbers](#) (on page 16-9) for information about event numbers. You can use a copy of the following table to record settings for your instrument.

Bit	Decimal value	Set Event	Clear Event	When set, indicates the following has occurred:
0	1			
1	2			
2	4			
3	8			
4	16			
5	32			
6	64			
7	128			
8	256			
9	512			
10	1024			
11	2048			
12	4096			
13	8192			
14	16 384			

Mapping events to bits

To program the Questionable and Operation Event Registers, you map events to specific bits in the register. This causes a bit in the condition and event registers to be set (or cleared) when the specified event occurs. You can map events to bits B0 through B14 (bit B15 is always set to zero).

When you have a mapped-set event, the bits in the corresponding condition register and event register are set when the mapped-set event is detected. The bits remain at 1 until the event register is read or the status model is reset.

When you have a mapped-clear event, the bit in the condition register is cleared to 0 when the event is detected.

You can map any event to any bit in these registers. An event is the number that accompanies an error, warning, or informational message that is reported in the event log. For example, for the event code "Error -221, Settings Conflict," the event is -221. Note that some informational messages do not have a related event number, so they cannot be mapped to a register.

You do not need to map clear events to generate SRQs. However, if you want to read the condition register to report status, you must map both a set event and a clear event. If no clear event is mapped, the bits are cleared only when the instrument power is turned off and turned on.

You can use the following SCPI commands to read and map events to bits in the programmable registers:

- [.STATus:OPERation:MAP](#) (on page 12-100)

This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.

- [.STATus:QUESTIONable:MAP](#) (on page 12-103)

This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.

You can use the following TSP commands to read and map events to bits in the programmable registers:

- [status.operation.getmap\(\)](#) (on page 14-207)

This command reads the mapped set and clear status for the specified operation event bit.

- [status.operation.setmap\(\)](#) (on page 14-208)

This command maps the set and clear events to a specified operation event register bit.

- [status.questionable.getmap\(\)](#) (on page 14-211)

This command reads the mapped set and clear status for the specified questionable event bit.

- [status.questionable.setmap\(\)](#) (on page 14-212)

This command maps the set and clear events to a specified questionable event register bit.

You can map any event that appears with a number in the event queue to any available bit in a programmable register. The programmable registers and their relationships to the Status Byte Register are shown in the [Programmable status registers diagram](#) (on page 16-5). The following example event queue log entries contain actual events that can be mapped to a status model bit.

```
2731 Trigger Model Initiated "Trigger model #1 has been initiated"
2732 Trigger Model Idle "Trigger model #1 has been idled"
4917 Reading buffer cleared "Reading buffer <buffer name> is 0% filled"
4918 Reading buffer full "Reading buffer <buffer name> is 100% filled"
5080 SMU Source Limit Tripped "Source limiting is active on output"
5081 SMU Source Limit Cleared "Source limiting is no longer necessary and output is normal"
```

See [Using the event log](#) (on page 3-52) for additional information on finding events.

Event numbers

You can decide what events to use to set and clear the bits in the Operation Event Register and Questionable Event Register. The following table lists the event numbers and descriptions.

NOTE

This table lists frequently used event numbers. This table does not list all event numbers.

Event number	Event description for user
2728	Trigger model was running, but then aborted by user action.
2730	The trigger model was running, but has failed and returned to the idle possibly due to an error in settings.
2731	Trigger model was idle, but is now running.
2732	Trigger model was running, but completed successfully, and is now in idle.
2733	The trigger model has ended in specified block number due to the specified condition.
2734	A trigger model block has logged a user-defined informational event message.
2735	A trigger model block has logged a user-defined informational event message.
2736	A trigger model block has logged a user-defined informational event message.
2737	A trigger model block has logged a user-defined informational event message.
2738	A trigger model block has logged a user-defined warning event message.
2739	A trigger model block has logged a user-defined warning event message.
2740	A trigger model block has logged a user-defined warning event message.
2741	A trigger model block has logged a user-defined warning event message.
2742	A trigger model block has logged a user-defined error event message.
2743	A trigger model block has logged a user-defined error event message.
2744	A trigger model block has logged a user-defined error event message.
2745	A trigger model block has logged a user-defined error event message.
2771	The instrument is nearing an internal temperature where corrective action will be necessary. The next limit will automatically turn off source outputs or limit other functionality.
2774	The instrument is experiencing an internal temperature malfunction since it is unable to supply readings and requires immediate service.
2775	An internal malfunction had cleared itself and the instrument is operating normally.

Event number	Event description for user
2776	The instrument is operating at normal temperature levels and fully functional.
2777	The instrument temperature has reached a level in which source output and other functionality will not be available.
2778	The instrument temperature has fallen to a level in which source output and other functionality is now available.
4917	The specified reading buffer is cleared (empty).
4918	The specified reading buffer is full.
5080	The source output has gone over the source limit level and is being limited.
5081	The source output has returned below the source limit level and is no longer being limited.

Status Byte Register

The Status Byte Register monitors the registers and queues in the status model and generates service requests (SRQs).

When bits are set in the status model registers and queues, they generate summary messages that set or clear bits of the Status Byte Register. You can enable these bits to generate an SRQ.

Service requests (SRQs) instruct the controller that the instrument needs attention or that some event has occurred. When the controller receives an SRQ, the controller can interrupt existing tasks to perform tasks that address the request for service.

For example, you might program your instrument to send an SRQ when a specific instrument error event occurs. To do this, you set the Status Request Enable bit 2 (EAV). In this example, the following actions occur:

- The error event occurs.
- The error event is logged in the Error Queue.
- The Error Queue sets the EAV bit of the Status Byte Register.
- The EAV bits are summed.
- The RQS bit of the Status Byte Register is set.
- On a GPIB system, the SRQ line is asserted. On a VXI-11 or USB system, an SRQ event is generated.

For an example of this, see the example code provided in [SRQ on error](#) (on page 16-21).

The summary messages from the status registers and queues set or clear the appropriate bits (B0, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are solely dependent on the summary messages (0 or 1). For example, if the Standard Event Register is read, its register will clear. As a result, its summary message resets to 0, which in turn resets the ESB bit in the Status Byte Register.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status (MSS) bit.

When using the GPIB, USB, or VXI-11 serial poll sequence of the 2450 to get the status byte (serial poll byte), bit B6 is the RQS bit. See [Serial polling and SRQ](#) (on page 16-13) for details on using the serial poll sequence.

When using the *STB? common command or `status.condition` command to read the status byte, bit B6 is the MSS bit.

To reset the bits of the Service Request Enable Register to 0, use 0 as the parameter value for the command (for example, *SRE 0 or `status.request_enable = 0`).

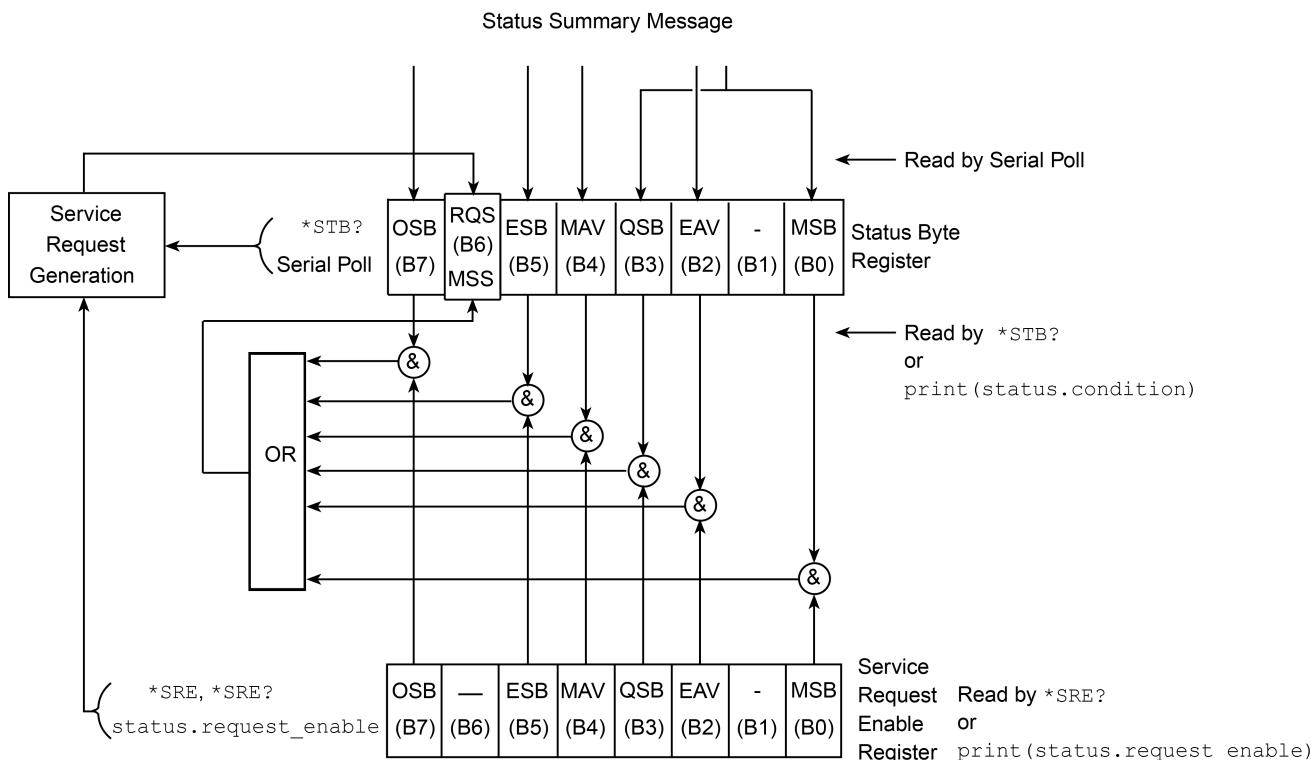
You can read and set which bits to AND in the Status Byte Register using the following commands.

Description	SCPI command	TSP command
Read the Status Byte Register	*STB? (on page 15-8)	status.condition (on page 14-204)
Read the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-213)
Enable bits in the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-213)

Status Byte Register diagram

The Status Byte Register consists of two 8-bit registers that control service requests, the Status Byte Register and the Service Request Enable Register. These registers are shown in the following figure.

Figure 158: 2450 Status Byte Register



The bits in the Status Byte Register are described in the following table.

Bit	Decimal value	Bit name	When set, indicates the following has occurred:
0	1	Measurement summary Bit (MSB)	An enabled questionable event
1	2	Not used	Not applicable
2	4	Error available (EAV)	An error is present in the error queue (warning and information messages do not affect this bit)
3	8	Questionable summary bit (QSB)	An enabled questionable event
4	16	Message available (MAV)	A response message is present in the output queue
5	32	Event summary bit (ESB)	An enabled standard event
6	64	Request for service (RQS)/Master summary status (MSS)	An enabled summary bit of the Status Byte Register is set; depending on how it is used, this is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit
7	128	Operation summary bit (OSB)	An enabled operation event

Service Request Enable Register

This register is programmed by the user and is used to enable or disable the setting of bit B6 (RQS/MSS) by the Status Summary Message bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in the [Status Byte Register](#) (on page 16-10) topic, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic “1” output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

You can set or clear the individual bits of the Service Request Enable Register by using the *SRE common command or `status.request_enable`. To read the Service Request Enable Register, use the *SRE? query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a *SRE 0 or `status.request_enable = 0` is sent). You can program and read the SRQ Enable Register using the following commands.

Description	SCPI command	TSP command
Read the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-213)
Enable bits in the Status Request Enable Register	*SRE (on page 15-7)	status.request_enable (on page 14-213)

Queues

The instrument includes an Output Queue and an Error Queue. The Output Queue holds messages from readings and responses. The Error Queue holds error event messages from the event log. Both are first-in, first-out (FIFO) registers.

Output Queue

The output queue holds response messages to SCPI query and TSP `print()` commands.

When data is placed in the Output Queue, the Message Available (MAV) bit in the Status Byte Register is set. The bit is cleared when the Output Queue is empty.

To clear data from the Output Queue, read the messages. To read a message from the Output Queue, address the instrument to talk after the appropriate query is sent.

Error Queue

The Error Queue holds any error events that are posted in the event log. When an error event occurs, it is posted to the Error Queue, which sets the Error Available (EAV) bit in the Status Byte Register.

The instrument clears error event messages from the event log when it retrieves the event log. When the error event messages are cleared from the event log, the EAV bit in the Status Byte Register is cleared.

You can clear the Error Queue by sending the common command `*CLS` or the TSP command `status.clear()`. Note that `status.clear()` also clears all event registers.

For information regarding the event log, see [Using the event log](#) (on page 3-52).

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ occurred and what caused it. If an SRQ occurred, the program can, for example, branch to an appropriate subroutine that will service the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

The serial poll does not clear the low-level registers that caused the SRQ to occur. You must clear the low-level registers explicitly. Refer to [Clearing registers](#) (on page 16-16).

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit remains set until all enabled Status Byte Register summary bits are reset.

For information on serial polling on a GPIB system, see [SPE, SPD](#) (on page 2-15).

Programming enable registers

You can program the bits in the enable registers of the Status Model registers.

When you program an enable register bit to 0, no action occurs if the bits in the corresponding registers are set (1).

When you program an enable register bit to 1, if the bits in the corresponding registers are set (1), the AND condition occurs and a bit in the Status Byte Register is set to (1).

You must program all bits in an enable register at the same time. This means you need to determine what each bit value in the register will be, then add them together to determine the value of all the bits in the register. See [Understanding bit settings](#) (on page 16-15) for more information on determining the value of the bits in the registers.

For example, you might want to enable the Standard Event Register to set the ESB bit in the Status Byte Register whenever an operation complete occurs or whenever an operation did not execute properly because of an internal condition. To do this, you need to set bits 0 and 3 of the Standard Event Register to 1. These bits have decimal values of 1 and 8, so to set both bits to 1, you set the register to 9.

Using SCPI:

Send the command:

```
*ese 9
```

Using TSP

Send the command:

```
status.standard.enable = 9
```

Reading the registers

You can read any register in the status model. The response is a decimal value that indicates which bits in the register are set. See [Understanding bit settings](#) (on page 16-15) for information on how to convert the decimal value to bits.

Using SCPI commands:

If you are using SCPI, you use the query commands in the STATus subsystem and common commands to read registers.

Using TSP commands:

If you are using TSP, you print the TSP command to read the register. You can use either `print()`, which returns the decimal value, or `print(tostring())`, which returns the string equivalent of the decimal value.

You can also send the common commands to read the register.

For example, you can send any one of the following commands to read the Status Enable Register of the Standard Event Register:

```
print(status.standard.enable)
*ese?
print(tostring(status.standard.enable))
```

Understanding bit settings

When you write to or read a status register, you can use binary, decimal, or hexadecimal values to represent the binary values of the bit states. When the value is converted to its binary equivalent, you can determine which bits are set on or clear. Zero (0) indicates that all bits are clear.

In the 2450, the least significant bit is always bit B0. The most significant bit differs for each register, but in most cases is either bit B7 or bit B15.

Bit position	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	1000 0000	0100 0000	0010 0000	0001 0000	1000	0100	0010	0001
Decimal value	128	64	32	16	8	4	2	1
Weight	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Bit position	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	1000 0000 0000 0000	0100 0000 0000 0000	0010 0000 0000 0000	0001 0000 0000 0000	1000 0000 0000 0000	0100 0000 0000 0000	0010 0000 0000 0000	0001 0000 0000 0000
Decimal value	32768	16384	8192	4096	2048	1024	512	256
Weight	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

For example, if a value of 129 is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set and all other bits are cleared. If you read a value of 12288 for the condition register, the binary equivalent is 0011 0000 0000 0000. This value indicates that bits B12 and B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

When bit B12 (4096) and bit B13 (8192) are set (1), the decimal equivalent is $4096 + 8192 = 12,288$.

Clearing registers

Registers in the status model can be cleared using commands or by instrument actions. When a register is cleared, the bits in the register are set to 0.

The event log and all registers are cleared when instrument power is cycled.

When you read a bit from the Operation Event, Questionable Event, or Standard Event Status Register, the entire 16-bit or 8-bit register value is returned. The event register is cleared or set to 0.

Using SCPI commands:

To clear the event registers of the Questionable Event Status Register and Operation Event Status Register sets, Standard Event Register, and Status Byte Register, send:

```
*CLS
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register and Operation Event Enable Register sets.

To clear the Questionable Event Status Register, the Operation Event Status Register sets, Standard Event Register, and Status Byte Register, send:

```
STATus : CLEar
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register or Operation Event Enable Register sets.

Using TSP commands:

To clear the event registers of the Questionable Event Status Register and Operation Event Status Register sets, Standard Event Register, and Status Byte Register send:

```
*CLS
```

To clear the Questionable Event Status Register, the Operation Event Status Register sets, Standard Event Register, and Status Byte Register send:

```
status.clear()
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register or Operation Event Enable Register sets.

Status model programming examples

The following examples illustrate how to generate an SRQ using the status model.

SRQ when the SMU reaches its source limit

This example demonstrates how to generate a service request (SRQ) when the source-measure unit (SMU) detects it has reached its source limit. After configuring the status model, in this example the source will be configured to output current and measure voltage. If the output terminals of the SMU are left open when running this example, the SMU will reach its source limit and generate an SRQ.

Using SCPI commands:

```
*RST
STAT:CLE
STAT:OPER:MAP 0, 5080, 5081
STAT:OPER:ENAB 1
*SRE 128
SOUR:FUNC CURR
SOUR:CURR:RANG 1e-3
SOUR:CURR 1e-3
SOUR:CURR:VLIM 1
SENS:FUNC "VOLT"
OUTP ON
READ?
OUTP OFF
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()
-- Map the event numbers
-- Map bit 0 of operational status register to set on reaching the
-- source limit (5080) and clear on dropping below the source
-- limit (5081)
status.operation.setmap(0, 5080, 5081)
-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1
-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB
-- This code will make SMU reach V limit if output terminals are open
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 1e-3
smu.source.level = 1e-3
smu.source.vlimit.level = 1
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.source.output = smu.ON
print(smu.measure.read())
smu.source.output = smu.OFF
```

SRQ when trigger model is finished

This example shows you how to generate a service request (SRQ) when the trigger model is completed and the 2450 has returned to the Idle state. After configuring the status model, this code configures and runs the trigger model. When the trigger model completes, the instrument generates an SRQ and the data is returned.

Using SCPI commands:

```
*RST
TRAC:CLE
STAT:CLE
STAT:OPER:MAP 0, 2732, 2731
STAT:OPER:ENAB 1
*SRE 128
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
TRIG:BLOC:BUFF:CLE 1
TRIG:BLOC:SOUR:STAT 2, ON
TRIG:BLOC:DEL:CONS 3, 100e-3
TRIG:BLOC:MDIG 4, "defbuffer1"
TRIG:BLOC:BRAN:COUN 5, 9, 3
TRIG:BLOC:SOUR:STAT 6, OFF
INIT
*WAI
TRAC:DATA? 1, 9, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the reading buffer
defbuffer1.clear()
-- Clear the status byte
status.clear()
-- Map bit 0 of operational status register to set on trigger
-- model exit (2732) and clear on trigger model enter (2731).
status.operation.setmap(0, 2732, 2731)
-- Enable bit 0 to flow through to the status byte
status.operation.enable = 1
-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB
-- Configure a simple measurement loop using the trigger model
smu.source.level = 1
smu.source.ilimit.level = 10e-3
-- Configure a simple trigger model to take 10 measurements
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 100e-3)
trigger.model.setblock(4, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 9, 3)
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Start the trigger model
trigger.model.initiate()
-- Instrument will generate an SRQ when done
waitcomplete()
printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ on trigger model notify event

This example shows you how to use the Log Event block of the trigger model to generate a service request (SRQ). This example configures the trigger model to perform a sweep and to repeat that sweep multiple times. The Log Event block is used to generate events to indicate when a sweep starts and when it ends. An SRQ is generated each time the sweep ends. This example is most useful when you are gathering several sweeps of data on a device and you want to notify the controlling computer when each sweep has completed so it can retrieve the data from the sweep without interrupting the test.

Using SCPI commands:

```
*RST
SOUR:CONF:LIST:CRE "sourceList"
SOUR:VOLT:RANG 10
SOUR:VOLT:ILIM 10e-3
SENS:NPLC 1
SENS:CURR:RANG 10e-3
SENS:AZER:ONCE
SOUR:VOLT 1
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 2
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 3
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 4
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 5
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 6
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 7
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 8
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 9
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 10
SOUR:CONF:LIST:STORE "sourceList"
TRAC:CLE
TRIG:BLOC:CONF:RECALL 1, "sourceList"
TRIG:BLOC:SOUR:STAT 2, ON
TRIG:BLOC:LOG:EVEN 3, INFO1, "Sweep started."
TRIG:BLOC:BRAN:ONCE 4, 6
TRIG:BLOC:CONF:NEXT 5, "sourceList"
TRIG:BLOC:DEL:CONS 6, 1e-3
TRIG:BLOC:MDIG 7, "defbuffer1"
TRIG:BLOC:BRAN:COUN 8, 11, 5
TRIG:BLOC:LOG:EVEN 9, INFO2, "Sweep complete."
TRIG:BLOC:BRAN:COUN 10, 5, 3
TRIG:BLOC:SOUR:STAT 11, OFF
STAT:CLE
STAT:OPER:MAP 0, 2735, 2734
```

```
STAT:OPER:ENAB 1
*SRE 128
INIT
*WAI
TRAC:DATA? 1, 55, "defbuffer1", READ
```

Using TSP commands:

```
reset()
smu.source.configlist.create("sourceList")
smu.source.range = 10
smu.source.ilimit.level = 10e-3
smu.measure.nplc = 1
smu.measure.range = 10e-3
smu.measure.autozero.once()
for i = 0, 10 do
    smu.source.level = i
    smu.source.configlist.store("sourceList")
end
defbuffer1.clear()

-- Configure the trigger model
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "sourceList")
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(3, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "Sweep
    started.")
trigger.model.setblock(4, trigger.BLOCK_BRANCH_ONCE, 6)
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "sourceList")
trigger.model.setblock(6, trigger.BLOCK_DELAY_CONSTANT, 1e-3)
trigger.model.setblock(7, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(8, trigger.BLOCK_BRANCH_COUNTER, 11, 5)
trigger.model.setblock(9, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Sweep
    complete.")
trigger.model.setblock(10, trigger.BLOCK_BRANCH_COUNTER, 5, 3)
trigger.model.setblock(11, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Configure the Status Model
=====
-- Clear the status byte
status.clear()

-- Map the notify messages to operational register bit 0.
-- The Sweep Complete message will set the bit while the Sweep
-- Started message will clear the bit.
status.operation.setmap(0, trigger.LOG_INFO2, trigger.LOG_INFO1)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master Summary Bit/SRQ
status.request_enable = status.OSB

-- Start the trigger model.
trigger.model.initiate()

waitcomplete()
print(string.format("Number of Readings in Buffer: %d", defbuffer1.n))
printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ on error

This example shows you how to generate a service request (SRQ) when an instrument error event occurs.

Using SCPI commands:

```
*RST  
SYST:CLE  
STAT:CLE  
*SRE 4  
MAKEERROR
```

Using TSP commands:

```
reset()  
-- Clear Error Queue so EAV bit can go low.  
eventlog.clear()  
  
-- Clear the status byte.  
status.clear()  
  
-- Enable SRQ on error available.  
status.request_enable = status.EAV  
  
-- Send a line of code that will generate an error event.  
beeper = 1
```

SRQ when reading buffer becomes full

This example shows you how to generate a service request (SRQ) when the 2450 reading buffer is full. You can use this to notify the controlling computer that it needs to read back the data and empty the buffer. After configuring the status model, this code configures the default reading buffer 1 to a size of 100, and then configures the 2450 to fill the buffer. After the buffer is full, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST  
STAT:CLE  
STAT:OPER:MAP 0, 4918, 4917  
STAT:OPER:ENAB 1  
*SRE 128  
TRAC:CLE  
TRAC:POIN 100, "defbuffer1"  
SOUR:VOLT:RANG 1  
SOUR:VOLT 1  
SOUR:VOLT:ILIM 10e-3  
COUNT 100  
OUTP ON  
READ? "defbuffer1"  
OUTP OFF  
TRAC:DATA? 1, 100, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()

-- Map bit 0 of operational status register to set on default buffer
-- full (4918) and clear on buffer empty (4917).
status.operation.setmap(0, 4918, 4917)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB

-- Clear the buffer and make it smaller
defbuffer1.clear()
defbuffer1.capacity = 100
smu.source.range = 1
smu.source.level = 1
smu.source.ilimit.level = 10e-3

-- Set the measure count to fill the buffer
smu.measure.count = 100
smu.measure.range = 10e-3
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
smu.source.output = smu.OFF

printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ when a measurement completes

This example shows you how to generate a service request (SRQ) when a measurement completes. This is most useful when you have a measurement that will take a long time to complete and you wish to free up the controlling computer to do other things while it is waiting. This can happen if you have configured the measurement with a long delay value, a large aperture, or have enabled filtering.

This code configures the source-measure unit (SMU) for a long reading, and then configures the trigger model to generate notify events before and after the measurement. The status model is then configured to generate an SRQ based on the "Measurement Done" notify event. The output is turned on and the trigger model is used to take the measurement. When the measurement completes, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST
TRAC:CLE
SOUR:VOLT:RANG 1
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
SENS:CURR:RANG 10e-3
SENS:NPLC 10
TRIG:BLOC:LOG:EVEN 1, INFO1, "Measurement Started."
TRIG:BLOC:MDIG 2, "defbuffer1"
TRIG:BLOC:LOG:EVEN 3, INFO2, "Measurement Done."
STAT:CLE
STAT:OPER:MAP 0, 2735, 2734
STAT:OPER:ENAB 1
*SRE 128
OUTP ON
INIT
*WAI
OUTP OFF
TRAC:DATA? 1, 1, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the reading buffer
defbuffer1.clear()

-- Configure the source and take a measurement
smu.source.range = 1
smu.source.level = 1
smu.source.ilimit.level = 10e-3

smu.measure.range = 10e-3

-- Setup the NPLC for a really long measurement
smu.measure.nplc = 10

-- Use the trigger model to create events before and after
-- the measurement to generate SRQ when measurement is done.
trigger.model.setblock(1, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "Measurement
    Started.")
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(3, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Measurement
    Done.")

-- Clear the status byte
status.clear()

-- Map bit 0 of the Operation Event Register to set on the Measurement
-- Done log notification (trigger.LOG_INFO2) and clear on the
-- Measurement Started log notification (trigger.LOG_INFO1).
status.operation.setmap(0, trigger.LOG_INFO2, trigger.LOG_INFO1)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1
```

```
-- Enable the Operational Summary Bit to set the Master Summary
-- Bit/RQS
status.request_enable = status.OSB

smu.source.output = smu.ON
trigger.model.initiate()
waitcomplete()
smu.source.output = smu.OFF

printbuffer(1, defbuffer1.n, defbuffer1)
```