tek.com/keithley

Model 2651A High Power System SourceMeter Instrument

Reference Manual

2651A-901-01 Rev. C October 2021

2651A-901-01C

KEITHLEY
A Tektronix Company

Model 2651A

High Power System SourceMeter® Instrument

Reference Manual

These are the original instructions in English.

Document number: 2651A-901-01 Rev. C October 2021

KEITHLEY
A Tektronix Company

Safety precautions

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

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

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The  symbol indicates a connection terminal to the equipment frame.

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

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

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

Safety precaution revision as of June 2017.

Table of contents

Section 1

Introduction

In this section:

Welcome

Thank you for choosing a Keithley Instruments product. The Model 2651A High Power System SourceMeter® Instrument provides manufacturers of electronic components and semiconductor devices with an instrument that combines source and measurement capabilities in a single instrument called a source-measure unit (also called a SMU). This combination simplifies test processes by eliminating synchronization and connection issues associated with multiple instrument solutions. A Model 2651A provides a scalable, high throughput, highly cost-effective solution for precision dc, pulse, and low frequency ac source-measure testing that also maintains code compatibility with the Series 2600B instruments.

Extended warranty

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

Contact information

If you have any questions after you review the information in this documentation, please contact your local Keithley Instruments office, sales partner, or distributor. You can also call the Tektronix corporate headquarters (toll-free inside the U.S. and Canada only) at 1-800-833-9200. For worldwide contact numbers, visit tek.com/contact-us.

Customer documentation

The documentation for the Model 2651A includes a Quick Start Guide, User's Manual, and Reference Manual. The Model 2651A Quick Start Guide is provided as a hard copy with the instrument. You can also access it from tek.com/keithley as an Adobe Acrobat PDF file.

Quick Start Guide: Provides unpacking instructions, describes basic connections, and reviews basic operation information. If you are new to Keithley Instruments equipment, refer to the Quick Start Guide to take the steps needed to unpack, set up, and verify operation.

User's Manual: Provides detailed applications to help you achieve success with your Model 2651A. It provides information about the front panel to familiarize you with the instrument. It also presents an overview of each application, followed by instructions to complete the application using the front panel and TSP code.

Reference Manual: Includes advanced operation topics and maintenance information. Programmers looking for a command reference and users looking for an in-depth description of the way the instrument works (including troubleshooting and optimization) should refer to the Reference Manual.

Organization of manual sections

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

- **General operation:** Describes the components of the instrument and basic operation.
- **Functions and features:** Describes features and functions, such as relative offset, filters, reading buffers, triggering, the digital I/O port, and TSP-Link trigger lines.
- **Theory of operation:** Describes the internal circuitry and software of the Model 2651A in detail.
- **Remote commands:** Describes the basics of using remote commands to control the instrument, including descriptions of the factory scripts.
- **Instrument programming:** Describes how to control the instrument using TSP, TSB and TSB Embedded, TSP-Link system expansion, and TSP-Net.
- **Command reference:** Programming notes and an alphabetical listing of all commands available for the Model 2651A.
- **Troubleshooting guide:** Description of the error queue and basic LAN troubleshooting.
- **Frequently asked questions (FAQs):** Information that addresses commonly asked questions.
- **Next steps:** Contains sources of additional information.
- **Maintenance:** Information on instrument maintenance, including line fuse replacement and firmware upgrades.
- **Calibration:** How to verify and adjust the Model 2651A.
- **LAN concepts and settings:** How to set up the Model 2651A for use on a local area network.

- **Common commands:** Descriptions of IEEE Std. 488.2 common commands.
- **Status model:** Description of the Model 2651A status model.
- **Display character codes:** Listing of the decimal values for the display character codes and their corresponding displays.

Capabilities and features

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

- Resistance and power measurement functions
- Four-quadrant sink or source operation
- Contact check function
- High-capacitance mode for load impedance up to 50 µF (microfarads)
- Linear, logarithmic, and custom sweeping and pulsing
- Filtering to reduce reading noise
- High-speed sampling fast analog-to-digital converter (ADC); the fast ADC uses a hardware buffer that can store up to 5,000 readings in a single measure action
- A trigger model that supports extensive triggering and synchronization schemes at hardware speeds
- Internal memory that stores five user setup options
- Dedicated reading buffers that can each store and recall over 140,000 measurements; additional dynamic reading buffers can be created
- USB flash drive access for saving data buffers, test scripts, and user setups
- Digital I/O port that allows the Model 2651A to control other devices
- Compatible with Keithley IVy, a wireless I-V characterization tool
- LXI®
- Embedded TSP scripting engine that is accessible from any host interface; responds to high-speed test scripts comprised of instrument control commands
- TSP-Link® expansion bus that allows TSP-enabled instruments to trigger and communicate with each other; advanced Test Script Processor (TSP®) scripting engine features enable parallel script execution across the TSP-Link network
- Supports IEEE-488 (GPIB), RS-232, and ethernet local area network (LAN) connections

Model-specific capabilities

Additional source and measure features:

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

General information

Displaying the serial number

The instrument serial number is on a label on the rear panel of the instrument. You can also access the serial number from the front panel using the front-panel keys and menus.

To display the serial number on the front panel:

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

Section 2

General operation

In this section:

General ratings

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

Category	Specification
Supply voltage range	100 Vac to 240 Vac, 50 Hz or 60 Hz, 550 VA maximum
Input and output connections	See Rear panel (on page 2-6)
Environmental conditions	For indoor use only: **Altitude:** Maximum 2000 meters (6562 feet) above sea level **Operating:** 0 °C to 50 °C, 70% relative humidity up to 35 °C. Derate 3% relative humidity/°C, 35 °C to 50 °C **Storage:** –25 °C to 65 °C

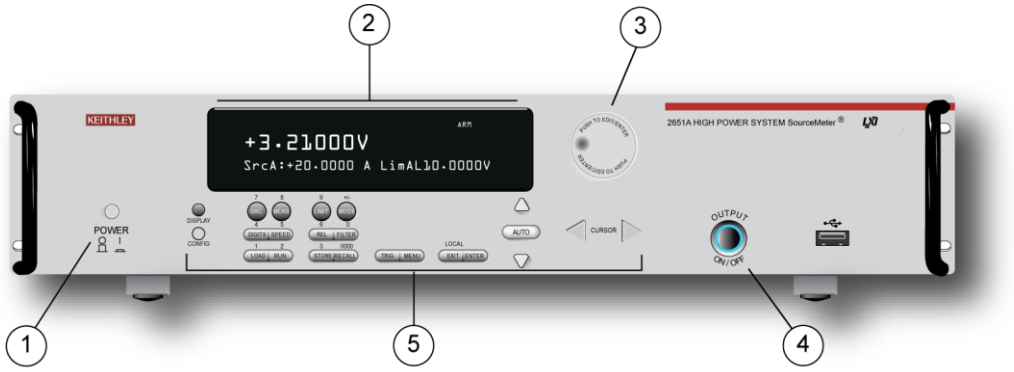
Controls, indicators, and connectors

Model 2651A controls, indicators, and the USB port are on the front panel (on page 2-2). Make connections to the Model 2651A through connectors on the rear panel (on page 2-6).

Front panel

The front panel of the Model 2651A is shown below. The descriptions of the front-panel controls, USB port, and indicators follow the figure.

Figure 1: Front panel Model 2651A



1. Power switch, display and configuration keys

Power switch. The in position turns the Model 2651A on (I); the out position turns it off (O).

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

Configures a function or operation.

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

SMU (source-measure unit) setup

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

Performance control

DIGITS	Sets the display resolution (4½, 5½, or 6½ digits).
SPEED	Sets the measurement speed (FAST, MEDium, NORMAL, HI-ACCURACY, or OTHER). Setting the speed selects either the fast or integrating A/D converter. When FAST is selected, the fast A/D converter is used; any other selection (MEDium, NORMAL, HI-ACCURACY, or OTHER) selects the integrating A/D converter. When the integrating A/D converter is selected, the measurement speed and accuracy are set by controlling the measurement aperture. Also see Speed (on page 2-78).
REL	Controls relative measurements, which allows a baseline value to be subtracted from a reading.
FILTER	Enables or disables the digital filter. You can use this filter to reduce reading noise.

Special operation

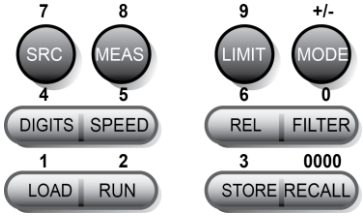
LOAD	Loads a test for execution (FACTORY, USER, or SCRIPTS).
RUN	Runs the last selected factory or user-defined test.
STORE	Accesses reading buffers and makes readings: ■ **TAKE_READINGS:** Use to make readings and store them in a reading buffer. ■ **SAVE:** Use to save a reading buffer to nonvolatile memory or to a user-installed flash drive (USB1) in CSV or XML format. Readings can include measurements, source values, and timestamp values.
RECALL	Recalls information (DATA or STATISTICS) stored in a reading buffer: ■ **DATA:** Includes stored readings, and if configured, source values and timestamp values. ■ **STATISTICS:** Includes MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, PK-PK.
TRIG	Triggers readings.
MENU	Accesses the main menu (on page 2-13). You can use the main menu to configure many functions and features.

EXIT Cancels the selection and returns to the previous menu or display. Also used as a LOCAL key to take the instrument out of remote operation.

ENTER Accepts the selection and moves to the next choice or exits the menu.

Numbers

Number keys When enabled and in EDIT mode, the number keys (0 to 9, +/-, 0000) allow direct numeric entry. Press the navigation wheel to enter EDIT mode. For more information, see Setting a value (on page 2-16).



3. Range keys

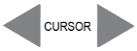
Selects the next higher source or measure range.

Enables or disables source or measure autorange.

Selects the next lower source or measure range.

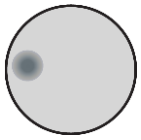
In addition to selecting range functions, the up and down range keys change the format for numbers that are not ranges, such as the limit value.

4. Cursor keys



Use the CURSOR keys to move the cursor left or right. When the cursor is on the source or compliance value digit, press the navigation wheel to enter edit mode, and turn the navigation wheel to edit the value. Press the navigation wheel again when you finish editing.
Use the CURSOR keys or the navigation wheel to move through menu items. To view a menu value, use the CURSOR keys for cursor control, and then press the navigation wheel to view the value or submenu item.

5. Navigation wheel



Turn the navigation wheel to:

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

Push the navigation wheel to:

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

6. Output control

Turns the source output on or off. The source output is on when this switch is illuminated (blue).

7. USB port

Use the USB port to connect a USB flash drive to the instrument. You can use the USB flash drive to store reading buffer data, scripts, and user setups. You can also use it to upgrade the firmware.

8. Display indicators (not shown)

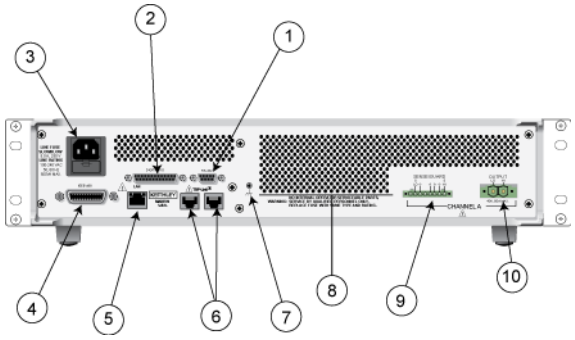
The items listed below represent the possible display indicators and their meanings.

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

Rear panel

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

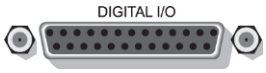
Figure 2: Rear panel Model 2651A



1. RS-232

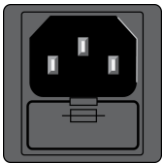
Female DB-9 connector. For RS-232 operation, use a straight-through (not null modem) DB-9 shielded cable for connection to a computer.

2. DIGITAL I/O



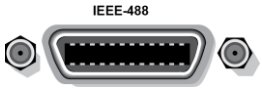
Female DB-25 connector. Includes fourteen digital input or output pins, seven GND pins, three +5 V pins, and one pin for output enable. For more information, see Digital I/O (on page 3-92).
Use a cable equipped with a male DB-25 connector (Keithley Instruments Model 7709-308).

3. Power module



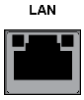
Contains the ac line receptacle and power line fuse. The instrument can operate on line voltages of 100 V to 240 Vac at line frequencies of 50 Hz or 60 Hz.

4. IEEE-488



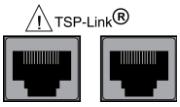
Connector for IEEE-488 (GPIB) operation. Use a shielded cable such as the Keithley Instruments Model 7007-1.

5. LAN



RJ-45 connector for a local area network (LAN). The LAN interface supports Auto-MDIX, so you can use either a CAT-5e crossover cable or a normal CAT-5e straight-through cable.

6. TSP-link

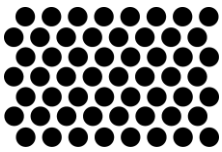


Expansion interface that allows a Model 2651A and other TSP-enabled instruments to trigger and communicate with each other. Use a category 5e or higher LAN crossover cable.

7. Chassis ground

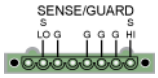
Ground screw for connections to chassis ground.

8. Cooling exhaust vent



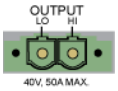
Exhaust vent for the internal cooling fan. Keep the vent free of obstructions to prevent overheating. Also see Cooling vents (on page 2-8).

9. Terminal connector



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

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



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

Cooling vents

The Model 2651A has top and side intake vents and a rear exhaust vent. The rear exhaust vent and either the top or both side intake vents must be unobstructed to properly dissipate heat.

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

Do not place a container of liquid (water or coffee, for instance) on the top cover. If it spills, the liquid may enter the case through the vents and cause severe damage.

CAUTION

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

The rear exhaust vent and either the top or both side intake vents must be unobstructed to properly dissipate heat. Even partial blockage could impair proper cooling.

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

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

If high power dissipation equipment is rack mounted next to the Model 2651A, it could cause excessive heating. To produce specified Model 2651A accuracies, maintain the specified ambient temperature around the surfaces of the Model 2651A. In rack configurations with convection cooling only, proper cooling practice places the hottest non-precision equipment (for example, the power supply) at the top of the rack away from and above precision equipment (such as the Model 2651A).

Mount precision equipment as low as possible in the rack, where temperatures are coolest. You can add space panels above and below the Model 2651A to help provide adequate airflow.

Starting up your instrument

The following topics describe how to power your instrument on and off, place the instrument in standby, configure the line frequency, and replace the line fuse.

Procedure

The Model 2651A operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz or 60 Hz. At the factory, each Model 2651A is configured to match the power line frequency appropriate for your country (either 50 Hz or 60 Hz). Make sure the operating voltage in your area is compatible.

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

CAUTION

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

To turn a Model 2651A on and off:

1. Before plugging in the power cord, make sure that the front panel POWER switch is in the off (O) position.
2. Connect the Model 2651A redundant protective earth (safety ground) on the Rear panel (on page 2-6).
3. Connect the female end of the supplied power cord to the ac receptacle on the rear panel.

WARNING

The power cord supplied with the Model 2651A contains a separate protective earth (safety ground) wire for use with grounded outlets. When proper connections are made, the instrument chassis is connected to power-line ground through the ground wire in the power cord. In addition, a redundant protective earth connection is provided through a screw on the rear panel. This terminal should be connected to a known protective earth. In the event of a failure, not using a properly grounded protective earth and grounded outlet may result in personal injury or death due to electric shock.

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

4. Connect the other end of the power cord to a grounded ac outlet.
5. To turn your instrument on, press the front panel **POWER** switch to place it in the on (I) position.
6. To turn your instrument off, press the front panel **POWER** switch to place it in the off (O) position.

Placing the Model 2651A in standby

⚠ WARNING

Placing the Model 2651A in standby does not place the instrument in a safe state (an interlock (on page 3-98) is provided for this function).

When the instrument is on, the output may be placed in an active output state (output on) or a standby mode (output off). From the front panel, pressing the **OUTPUT ON/OFF** control (see Output control (on page 2-5)) toggles the output using the present instrument configuration. You can also place the output in standby over the remote interface by sending the following command:

```
smua.source.output = 0
```

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

Warmup period

The Model 2651A must be turned on and allowed to warm up for at least two hours to achieve rated accuracies.

Line frequency configuration

The instrument automatically detects the power line frequency (either 50 Hz or 60 Hz) at each power-up. This detected line frequency is used for aperture (NPLC) calculations.

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

To configure the line frequency from the front panel:

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

To configure the line frequency from a remote interface:

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

```
localnode.linefreq = 60
```

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

```
localnode.autolinefreq = true
```

Fuse replacement

The power receptacle contains a fuse drawer (refer to Rear panel (on page 2-6)). This fuse protects the power-line input of the instrument. If the line fuse needs to be replaced, refer to Line fuse replacement (on page 11-1).

System information

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

To view the system information from the front panel:

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

To retrieve system information from a remote interface:

To retrieve the firmware revision and serial number, send the `*IDN?` query (see Identification query: *IDN? (on page 14-3) for more information).

To determine memory usage, see the meminfo() (on page 7-184) function.

To determine when calibration was last run, see smuX.cal.date (on page 7-227).

To determine when calibration is due, see smuX.cal.due (on page 7-228).

Menu overview

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

Menu navigation

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

Selecting menu items

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

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

NOTE

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

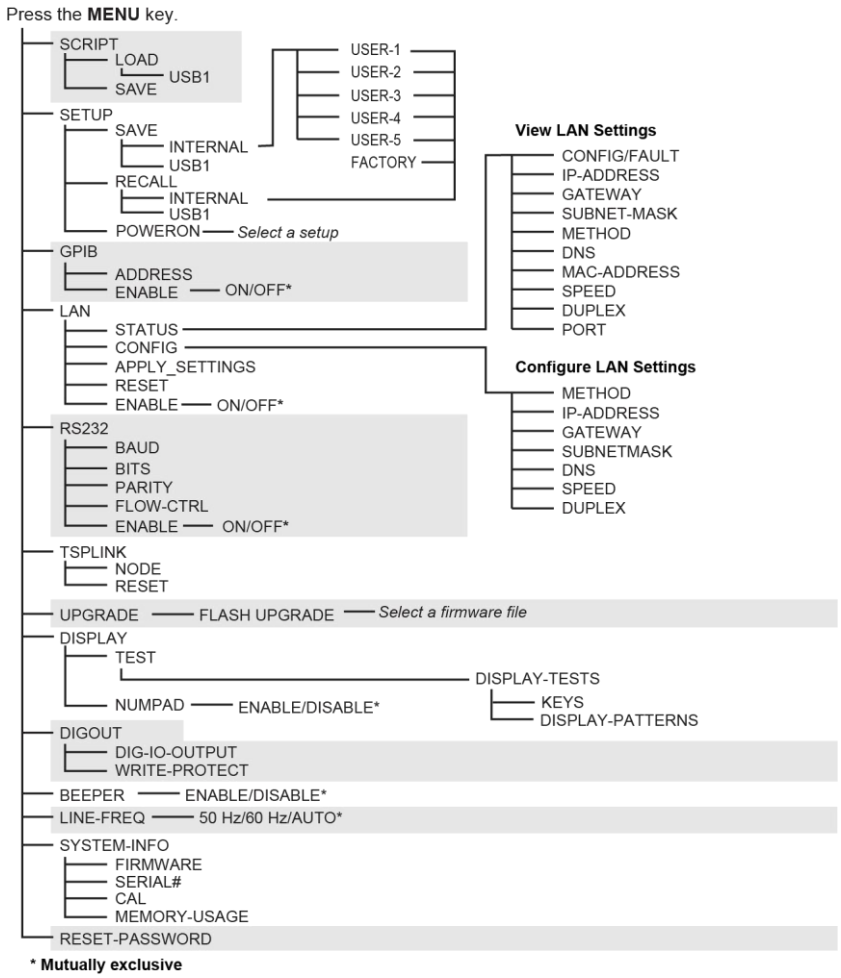
Menu trees

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

Main menu

The main menu structure is summarized in the following figure and table. For other menu items, see Configuration menu (on page 2-15).

Figure 3: Main menu tree



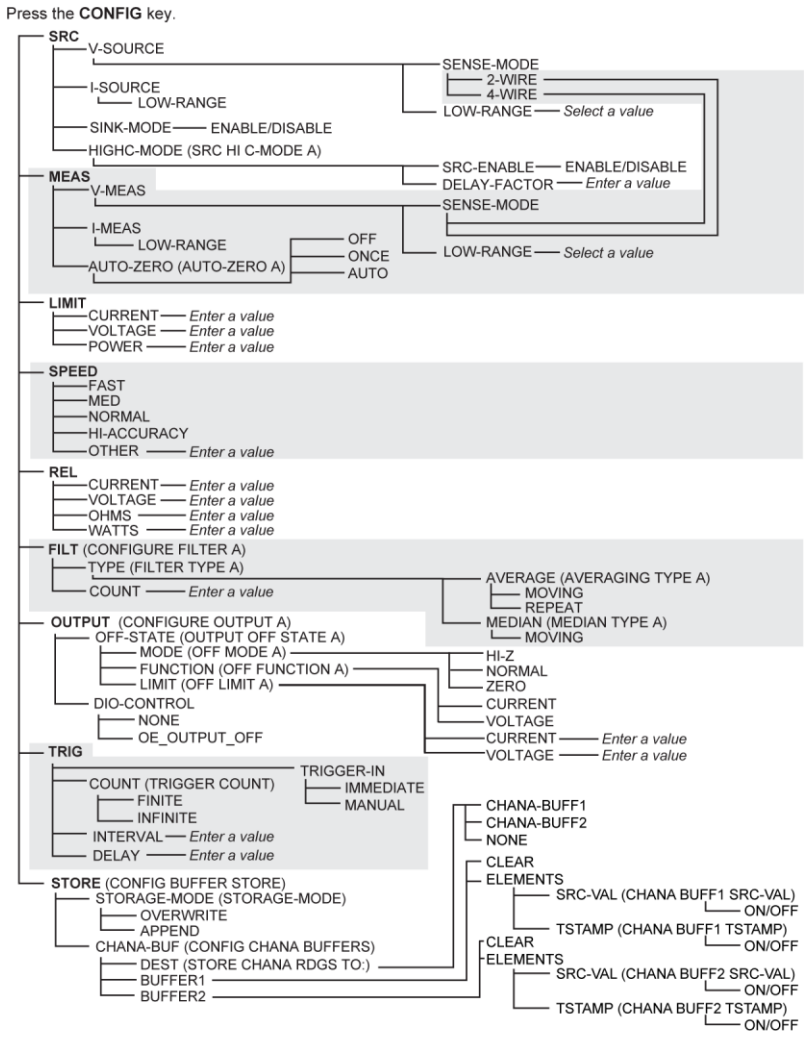
The following table contains descriptions of the main menu options and cross-references to related information. To access a menu option, press the **MENU** key, turn the navigation wheel to move the cursor to select an item, and press the navigation wheel.

Menu selection	Description	For more information, see:
SCRIPT - LOAD - SAVE	Saves and recalls users scripts Loads scripts into nonvolatile memory Saves scripts	Manage scripts (on page 6-3)
SETUP - SAVE - RECALL - POWERON	Saves and recalls user and factory setup options Saves user setup options Recalls user setup options Sets the configuration used during startup	Saved setups (on page 2-42)
GPIB - ADDRESS - ENABLE	Configures the GPIB interface options Configures the address for the GPIB interface Enables and disables the GPIB interface	Remote communications interfaces (on page 2-80), GPIB setup (on page 2-83)
LAN - STATUS - CONFIG - APPLY_SETTINGS - RESET - ENABLE	Configures the local area network (LAN) Displays LAN connection status Configures the LAN IP address and gateway Applies changes made using the CONFIG menu Restores the default settings Enables and disables the LAN interface	Remote communications interfaces (on page 2-80), LAN communications (on page 2-81), LAN concepts and settings (on page 2-81, on page 13-1)
RS232 - BAUD - BITS - PARITY - FLOW-CTRL - ENABLE	Controls the options for the RS-232 interface Sets the baud rate Configures the number of bits Sets the parity Configures the flow control Enables and disables the RS-232 interface	Remote communications interfaces (on page 2-80), RS-232 interface operation (on page 2-90)
TSPLINK - NODE - RESET	Configures the instrument in a TSP-Link® network Selects the instrument node identifier Resets the TSP-Link network	TSP-Link system expansion interface
UPGRADE	Upgrades the firmware from a USB flash drive	Upgrading the firmware (on page 11-4)
DISPLAY - TEST - NUMPAD	Accesses display functions Runs the display test Enables and disables the numeric keypad	Front panel tests (on page 11-2) See Numeric entry method in Setting a value (on page 2-16)
DIGOUT - DIG-IO-OUTPUT - WRITE-PROTECT	Controls digital outputs Selects the digital I/O values Write-protects specific digital I/O lines	Digital I/O (on page 3-92)
BEEPER - ENABLE - DISABLE	Controls the key beeps Enables the key beeps Disables the key beeps	Beeper (on page 2-18)
LINE-FREQ - 50Hz - 60Hz AUTO	Configures the line frequency Sets the line frequency to 50 Hz Sets the line frequency to 60 Hz Enables automatic line frequency detection during start up	Line frequency configuration (on page 2-10)
SYSTEM-INFO - FIRMWARE - SERIAL# - CAL - MEMORY-USAGE	Displays system information Displays the version of firmware installed Displays the serial number of the instrument Displays the last calibration date Displays the memory usage in kilobytes	System information (on page 2-11)
RESET-PASSWORD	Resets the system password	Password management (on page 6-41)

Configuration menu

The configuration menu structure is summarized in the following figure and table. For directions on navigating the menu, see Menu navigation (on page 2-11). For other menu items, see Main menu (on page 2-13).

Figure 4: CONFIG menu tree



NOTE

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

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

To access, press the CONFIG **key and then:**	**Options**	**For more information, see:**
SRC	V-source sense, low range; I-source low range; sink; and highC mode	Range (on page 2-71)
MEAS	V and I-measure sense, low range; auto-zero	Range (on page 2-71)
LIMIT	V-source and I-source compliance limits	Compliance limit (on page 2-20)
SPEED	Measurement speed (NPLC)	Speed (on page 2-78)
REL	Set relative values	Relative offset (on page 3-1)
FILTER	Control digital filter	Filters (on page 3-3)
OUTPUT ON/OFF	Set off-state, control digital I/O	Output-off states (on page 2-65)
TRIG	Set trigger in, count, interval, and delay	Triggering (on page 3-36)
STORE	Set buffer count and destination	Source-measure concepts (on page 4-2)

Setting values

Through the front panel, you can adjust a value using either the **Navigation wheel method** or **Numeric entry method** (using the keypad).

Setting a value

Navigation wheel method:

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

Numeric entry method:

1. If the keypad is disabled, press the **MENU** key, then select **DISPLAY > NUMPAD > ENABLE**.
2. Use the **CURSOR** arrow keys (or turn the navigation wheel) to move the cursor to the value that needs to be changed.
3. Press the navigation wheel or the **ENTER** key to enter edit mode. The EDIT indicator is illuminated.

4. Press any of the number keys (0-9, +/-, 0000) (see 2. SMU setup, performance control, special operation, and numbers (on page 2-3)). The cursor moves to the next digit on the right.
5. Repeat the above steps as required to set the values.
6. Press the **ENTER** key to select the value or press the **EXIT (LOCAL)** key to cancel the change.
7. To return to the main menu, press the **EXIT (LOCAL)** key.

NOTE

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

Setting source and compliance values

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

NOTE

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

To edit the source value:

1. Press the **SRC** key. The cursor flashes in the source value field.
2. Use the **CURSOR** keys (or turn the navigation wheel) to move the cursor to the digit that needs to be changed.
3. Press the navigation wheel or the **ENTER** key to edit the source value. The EDIT indicator is illuminated.
4. Change the source value (see Setting a value (on page 2-16)).

NOTE

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

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

To edit compliance limit values:

1. Press the **LIMIT** key.
2. Select the type of compliance (CURRENT, VOLTAGE, or POWER).
3. Press the navigation wheel or the **ENTER** key to enter edit mode. The EDIT indicator is illuminated.
4. Change the compliance value (see Setting a value (on page 2-16)).
5. When finished, press the **ENTER** key (the EDIT indicator is not illuminated).

NOTE

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

Beeper

The Model 2651A includes a beeper. When it is enabled, a beep indicates one of the following actions have occurred:

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

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

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

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

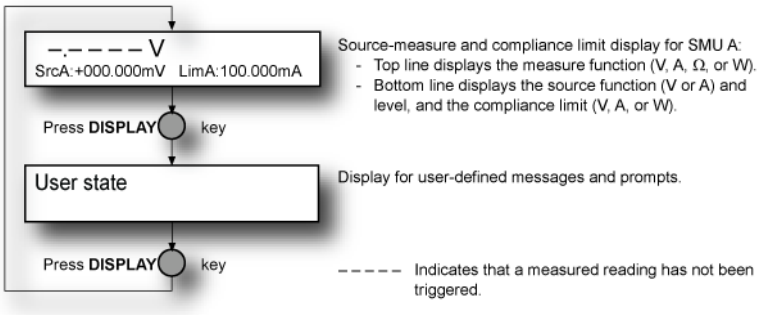
Set the `beeper.enable` attribute. For example, to enable the beeper, send:

```
beeper.enable = 1
```

Display mode

Use the **DISPLAY** key to scroll through the various display modes shown in the figure below. Refer to Display operations (on page 3-78) for more information about the display.

Figure 5: Display modes



Basic operation

WARNING

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2651A while the instrument is powered on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Operation overview

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

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

You can also configure source-measure operations using remote commands.

NOTE

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

Voltage and current

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

Model 2651A source-measure capabilities		
Range	**Source**	**Measure**
100 mV	± 101 mV	± 102 mV
1 V	± 1.01 V	± 1.02 V
10 V	± 10.1 V	± 10.2 V
20 V	± 20.2 V	± 20.4 V
40 V	± 40.4 V	± 40.8 V
1 µA	± 1.01 µA	± 1.02 µA
10 µA	± 10.1 µA	± 10.2 µA
100 µA	± 101 µA	± 102 µA
1 mA	± 1.01 mA	± 1.02 mA
10 mA	± 10.1 mA	± 10.2 mA
100 mA	± 101 mA	± 102 mA
1 A	± 1.01 A	± 1.02 A
5 A	± 5.05 A	± 5.1 A
10 A	± 10.1 A	± 10.2 A
20 A	± 20.2 A	± 20.4 A
50 A	± 50.5 A	± 51 A
Maximum power = 202 W		

Limits

When sourcing voltage, you can set the instrument to limit current or power. Conversely, when sourcing current, you can set the instrument to limit voltage or power. In steady-state conditions, the instrument output does not exceed the limit. The maximum limit is the same as the maximum values listed in the following table.

The limit circuit limits in either polarity regardless of the polarity of the source or limit value. The accuracy of the limit opposite in polarity from the source is diminished unless the instrument is in sink mode (on page 2-23). The maximum limits are based on source range. For more information, see Limit principles (on page 4-2).

The limit operation of the instrument changes dependent on the source mode (current or voltage), load, and the configured limits (current, voltage, and power). It is important to distinguish both the current and voltage limits from the power limit. As the names imply, the current limit restricts the current for sourced voltage, and the voltage limit restricts the voltage for a sourced current. The power limit, however, restricts power by lowering the present limit in effect (voltage or current) as needed to restrict the SMU from exceeding the specified power limit. For additional details on using limits, including load considerations when specifying both a current (or a voltage) limit and a power limit, see Operating boundaries (on page 4-5).

NOTE

The only exception to the limit not being exceeded is the voltage limit when operating as a current source. To avoid excessive and potentially destructive currents from flowing, the voltage limit sources or sinks up to 160 mA for current source ranges on or below 100 mA. For the ranges 1 A and 5 A, the maximum current is 5.4 A. For ranges 10 A and above, the maximum current allowed is the current source setting.

Maximum limits for Model 2651A

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

Setting the limit

Front-panel limit

Set the limit from the front panel as follows:

1. Press the **LIMIT** key to directly edit the limit value. Pressing the LIMIT key while in limit edit mode will toggle the display between the complementary function limit and the power limit display.
2. Press the navigation wheel and set the limit to the new value.
3. Press the **ENTER** key or the navigation wheel to complete editing.
4. Press the **EXIT (LOCAL)** key to return to the normal display.

Setting the limit from a remote interface

The table below summarizes basic commands to program a limit. For a more complete description of these commands, refer to the TSP command reference (on page 7-1).

Limit commands

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

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

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

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

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

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

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

Sink operation

CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the Model 2651A to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other Model 2651A instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

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

NOTE

The accuracy of the limit opposite in polarity from the source is diminished unless the instrument is in sink mode.

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

CAUTION

When using the I-Source as a sink, always set the voltage compliance limit to levels that are higher than the external voltage level. Failure to do so could result in excessive current flow into the Model 2651A (<102 mA) and incorrect measurements.

The sink operating limits are shown in Continuous power operating boundaries (on page 4-6).

Sink mode

When operating as a sink, limit inaccuracies are introduced. Enabling sink mode reduces the source limit inaccuracy seen when operating in quadrants II and IV. Quadrants I and III show this source limit inaccuracy.

Setting the sink mode using the front panel

To enable or disable the sink mode from the front panel:

1. Press the **CONFIG** key and then the **SRC** key.
2. Select **V-SOURCE**.
3. Select **SINK-MODE**.
4. Select **ENABLE** or **DISABLE**.
5. Press the **ENTER** key. Sink mode is enabled or disabled, as applicable.
6. Press the **EXIT (LOCAL)** key twice to return to the main display.

Setting the sink mode from a remote interface

To enable sink mode from the remote interface, send:

```
smua.source.sink = smua.ENABLE
```

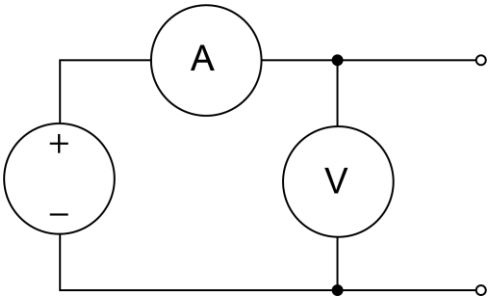
To disable sink mode, send:

```
smua.source.sink = smua.DISABLE
```

Fundamental circuit configurations

The fundamental source-measure configurations for the Model 2651A are shown in the following figures. When sourcing voltage, you can measure current or voltage, as shown in the following figure.

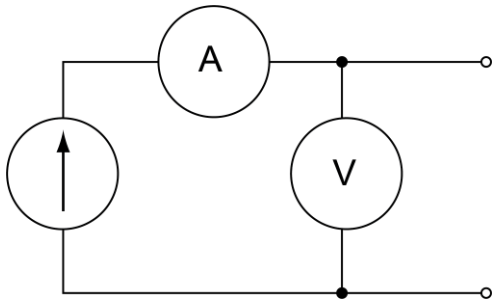
Figure 6: Fundamental source-measure configurations: Source V



A	Current meter
+ **–**	Voltage source
V	Voltage meter

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

Figure 7: Fundamental source-measure configuration: Source I



A	Current meter
↑	Current source
V	Voltage meter

See Basic circuit configurations (on page 4-20) for detailed information.

Operation considerations for the ADC

The following paragraphs discuss autozero and NPLC caching with the analog-to-digital converter (ADC) (on page 4-1). Autozero and NPLC caching only apply to the integrating ADC. They are not used with the fast (high-speed) ADC.

Autozero

The ADC of the Model 2651A uses a ratiometric analog to digital (A/D) conversion technique. To ensure reading accuracy, the instrument must periodically obtain fresh measurements of its internal ground and voltage reference. Separate reference and zero measurements are used for each aperture.

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

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

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

Autozero settings

Autozero setting	Description
OFF	Turns automatic reference measurements off.
ONCE	After immediately making one reference and one zero measurement, turns automatic reference measurements off.
AUTO	Automatically makes new acquisitions when the Model 2651A determines reference and zero values are out-of-date.

Setting autozero from the front panel

To change autozero from the front panel:

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

Setting autozero from a remote interface

To set autozero from a remote interface:

Use the autozero command with the appropriate option shown in the following table to set autozero through a remote interface (see smuX.measure.autozero (on page 7-245)). For example, send the following command to activate automatic reference measurements:

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

Autozero command and options

Command	Description
`smua.measure.autozero = smua.AUTOZERO_OFF`	Disable autozero (old NPLC cache values are used when autozero is disabled (see NPLC caching (on page 2-27)).
`smua.measure.autozero = smua.AUTOZERO_ONCE`	After immediately taking one reference and one zero measurement, turns automatic reference measurements off.
`smua.measure.autozero = smua.AUTOZERO_AUTO`	Automatically takes new acquisitions when the Model 2651A determines reference and zero values are out-of-date.

NPLC caching

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

Basic source-measure procedure

Front-panel source-measure procedure

Use the following procedure to perform the basic source-measure operations of the Model 2651A SMU using the front panel. The following procedure assumes that the Model 2651A is already connected to the device under test (DUT), as explained in DUT test connections (on page 2-45).

⚠ WARNING

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2651A while the instrument is powered on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Step 1: Select and set the source level

To select the source and edit the source value:

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

Step 2: Set the compliance limit

Perform the following steps to edit the compliance limit value:

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

Step 3: Select the measurement function and range

Select measurement function and range as follows:

1. Select the measurement function by pressing the **MEAS** key.
2. Set the measurement range with the **RANGE** keys, or enable **AUTO** range. When setting the range, consider the following points:
   - When measuring the source (such as when sourcing V and measuring V), you cannot select the measurement range using the RANGE keys. The selected source range determines the measurement range.
   - When not measuring the source (such as when sourcing V but measuring I), you can select the measurement range selection manually or the instrument can set it automatically. When using manual ranging, use the lowest possible range for best accuracy. When autorange is enabled, the Model 2651A automatically goes to the most sensitive range to make the measurement.

Step 4: Turn the output on

Turn on the output by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns on.

Step 5: Observe readings on the display.

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

Step 6: Turn the output off

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

Remote source-measure commands

Basic source-measurement procedures can also be performed through a remote interface. To do this, send the appropriate commands. The following table summarizes basic source-measure commands. See Introduction to TSP operation (on page 5-1) for more information on using these commands.

Basic source-measure commands

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

Requesting readings

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

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

Source-measure programming example

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

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

```
-- Restore Model 2651A defaults.
smua.reset()
-- Select voltage source function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set source range to autorange.
smua.source.autorangev = smua.AUTORANGE_ON
-- Set voltage source to 5 V.
smua.source.levelv = 5
-- Set current limit to 10 mA.
smua.source.limiti = 10e-3
-- Set current range to 10 mA.
smua.measure.rangei = 10e-3
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Print and place the current reading in the reading buffer.
print(smua.measure.i(smua.nvbuffer1))
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
```

Triggering in local mode

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

NOTE

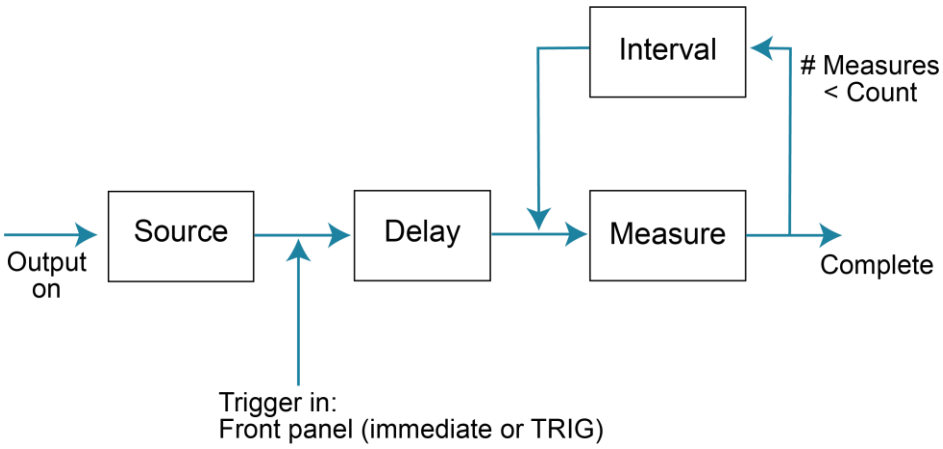
Press the **MENU** key, and then select **SETUP > RECALL > INTERNAL > FACTORY** to reset the factory default conditions.

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

- When the output is turned on, the programmed source value is immediately applied to the device under test (DUT).
- Through front panel only: If the immediate trigger source is selected, a measurement is triggered immediately. However, if the manual trigger source is selected, the front-panel TRIG key must be pressed.
- The instrument waits for the programmed delay period (if any).

- The instrument makes one measurement.
- If the number of measurements is less than the programmed trigger count, it cycles to make another measurement (the measurement cycle is repeated indefinitely if the infinite trigger count is selected).
- For multiple measurements, the instrument waits for the programmed trigger interval (if any) before making the next measurement.

Figure 8: Local triggering



Configuring trigger attributes in local mode

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

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

Front-panel triggering example

This example configures the trigger parameters to meet the following requirements:

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

To configure the trigger parameters:

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

Configuring for measure-only tests using the MODE key

In addition to using the Model 2651A for conventional source-measure operations, you can also use it like a meter to measure current, voltage, resistance, or power.

To configure the Model 2651A as a voltage meter, current meter, ohmmeter, or wattmeter:

1. Press the **MODE** key.
2. Turn the navigation wheel to select the type of meter from the menu (**I-METER**, **V-METER**, **OHM-METER**, or **WATT-METER**).
3. Press the **ENTER** key to complete the configuration of the Model 2651A as the selected meter.

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

- Voltmeter and ammeter measurements (on page 2-33)
- Ohms measurements (on page 2-34)
- Power measurements (on page 2-37)

Voltmeter and ammeter measurements

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

To use the Model 2651A to measure voltage or current:

1. Select the source-measure functions:
   - **Voltmeter:** Press the **SRC** key to select the current source and press the **MEAS** key to select the voltage measurement function.
   - **Ammeter:** Press the **SRC** key to select the voltage source and press the **MEAS** key to select the current measurement function.
2. Set source and compliance levels. To edit the source level, use the procedure provided in Step 1: Select and set the source level (on page 2-27); to edit the compliance level, use the procedure provided in Step 2: Set the compliance limit (on page 2-28):
   - Select the lowest source range and set the source level to zero.
   - Set the compliance level to a value that is higher than the expected measurement.

CAUTION

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

3. Use the **RANGE** keys to select a fixed measurement range that accommodates the expected reading. Use the lowest possible range for best accuracy. You can also select autorange, which automatically sets the Model 2651A to the most sensitive range.
4. Connect the voltage or current to be measured. Make sure to use 2-wire connections from the Model 2651A to the device under test (DUT) (see DUT test connections (on page 2-45)).
5. Press the **OUTPUT ON/OFF** control to turn the output on.
6. View the displayed reading (press the **TRIG** key if necessary).
7. When finished, press the **OUTPUT ON/OFF** control to turn the output off.

Ohms measurements

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

$R = V/I$

Where:

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

Ohms ranging

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

Basic ohms measurement procedure

When you use the MODE key to select ohms measurement, the Model 2651A is automatically configured as a current source with a level of 1 mA. If you wish to change the source function, source value, or compliance value (in other words, if you wish to customize the MODE key's standard ohm-meter's configuration), then perform the following steps to perform ohms measurements. The following procedure assumes that the Model 2651A is already connected to the device under test (see DUT test connections (on page 2-45)).

To take an ohms measurement:

1. Press the **SRC** key to select the source function.
2. Set the output source (current or voltage, dependent on which function is selected) to a value based on the expected resistance. See Step 1: Select and set the source level (on page 2-27) earlier in this section.
3. Press the **LIMIT** key to edit the voltage or current limit. When programming a voltage limit, set the voltage limit above the maximum expected voltage across the resistor under test. When programming a current limit, set the current limit at or above the maximum expected current through the resistor under test. See Step 2: Set the compliance limit earlier in this section.
4. Press the **MEAS** key to display voltage or current.
5. Make sure that AUTO measurement range is on (press the **AUTO** key if needed).
6. Press the **MEAS** key as many times as needed to display ohms.
7. Press the **OUTPUT ON/OFF** control to turn the output on.
8. View the displayed reading (press the **TRIG** key if necessary). When finished, press the **OUTPUT ON/OFF** control again to turn the output off.

Remote ohms command

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

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

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

See Remote source-measure commands (on page 2-29) for more commands to set up source and measure functions, and Introduction to TSP operation (on page 5-1) for more details.

Ohms programming example

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

- Source function: current, 10 mA range, 10 mA output
- Voltage measure range: auto
- Voltage compliance: 10 V
- Sense mode: 4-wire

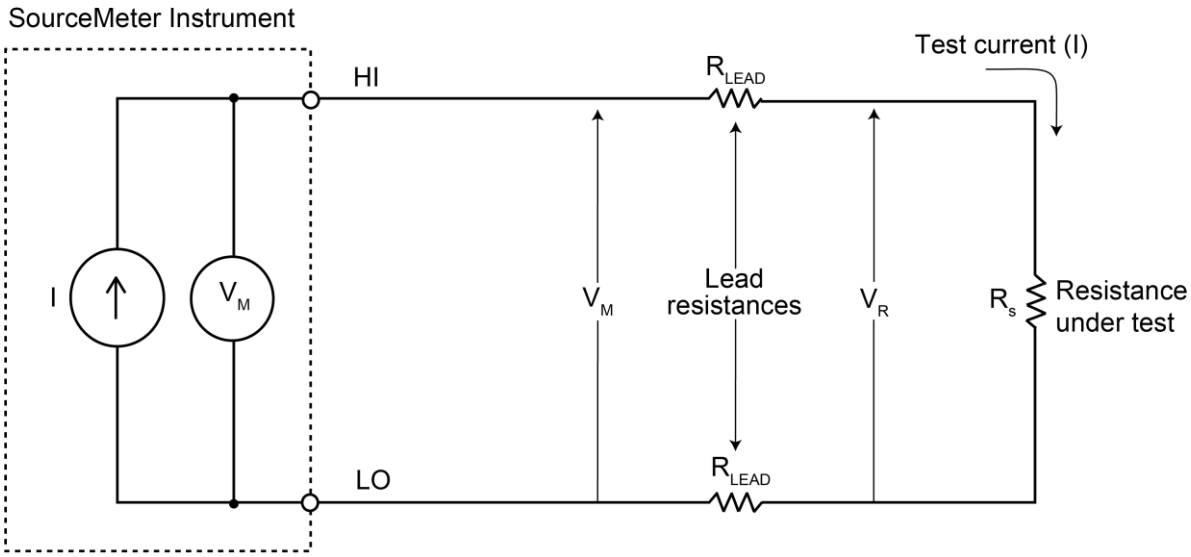
```
-- Restore Model 2651A defaults.
smua.reset()
-- Select current source function.
smua.source.func = smua.OUTPUT_DCAMPS
-- Set source range to 10 mA.
smua.source.rangei = 10e-3
-- Set current source to 10 mA.
smua.source.leveli = 10e-3
-- Set voltage limit to 10 V.
smua.source.limitv = 10
-- Enable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

Ohms sensing

Ohms measurements can be made using either 2-wire or 4-wire sensing. See DUT test connections (on page 2-45) for information on connections and sensing methods.

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

Figure 9: Two-wire resistance sensing



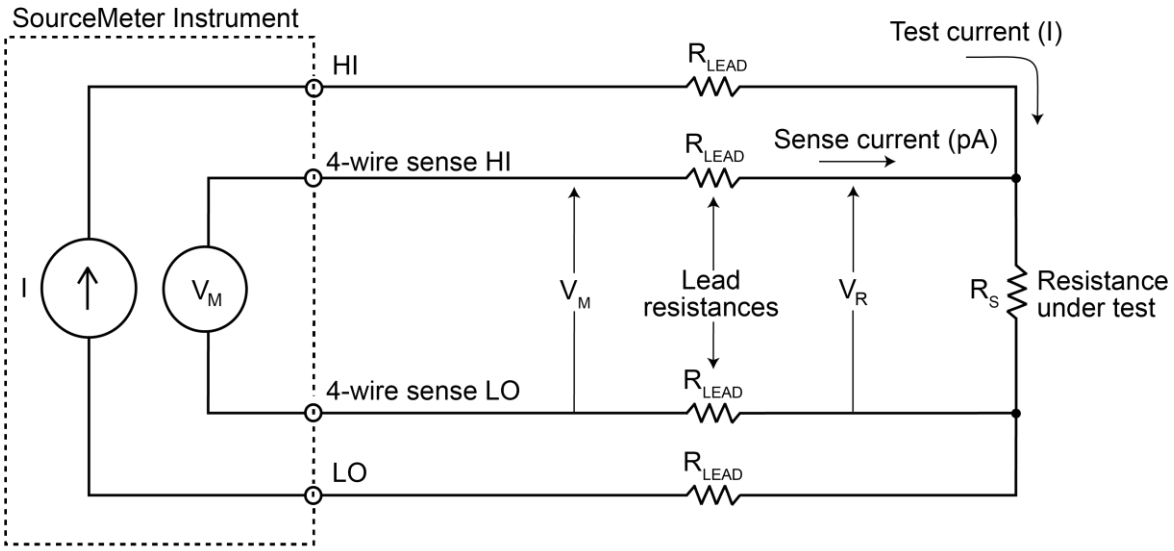
I = Current sourced
V_M = Voltage measured
V_R = Voltage across resistor

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

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

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

Figure 10: Four-wire resistance sensing



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

Because sense current is negligible, $V_M = V_R$

and measured resistance = $\frac{V_M}{I} = \frac{V_R}{I} = R_S$

Power measurements

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

$$P = V \times I$$

Where:

P is the calculated power

V is the measured voltage

I is the measured current

Basic power measurement procedure

If you need to customize the standard wattmeter configuration of the MODE key, perform the following steps to make power measurements. The following procedure assumes that the Model 2651A is already connected to the device under test (DUT) as explained in DUT test connections (on page 2-45).

⚠ WARNING

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2651A while the output is on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables. Putting the equipment into an output-off state does not guarantee that the outputs are powered off if a hardware or software fault occurs.

To perform power measurements from the front panel:

1. Set source function and value. Press the **SRC** key to select the voltage or current source function, as required.
2. Set the output voltage or current to an appropriate value.
3. Move the cursor to the digit to change, then press the navigation wheel to enter the EDIT mode.
4. Use the **RANGE** keys to select a range that accommodates the value you want to set. For best accuracy, use the lowest possible source range.
5. Enter the source value.
6. Press the **ENTER** key or the navigation wheel to complete editing.
7. Press the **LIMIT** key and set the voltage or current limit high enough for the expected voltage or current across the DUT to be measured.
8. Press the **LIMIT** key.
9. Move the cursor to the digit to change, then press the navigation wheel to enter the EDIT mode, as indicated by the EDIT indicator.
10. Enter the limit value, then press the **ENTER** key or the navigation wheel to complete editing.
11. Press the **MEAS** key as many times as needed to display power.
12. Press the **OUTPUT ON/OFF** control to turn the output on.
13. View the displayed reading (press the **TRIG** key if necessary).
14. When finished, press the **OUTPUT ON/OFF** control again to turn the output off.

Power measurements using the remote interface

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

Remote power reading command

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

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

See Remote source-measure commands (on page 2-29) for more commands necessary to set up source and measure functions and also Introduction to TSP operation (on page 5-1).

Power measurement programming example

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

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

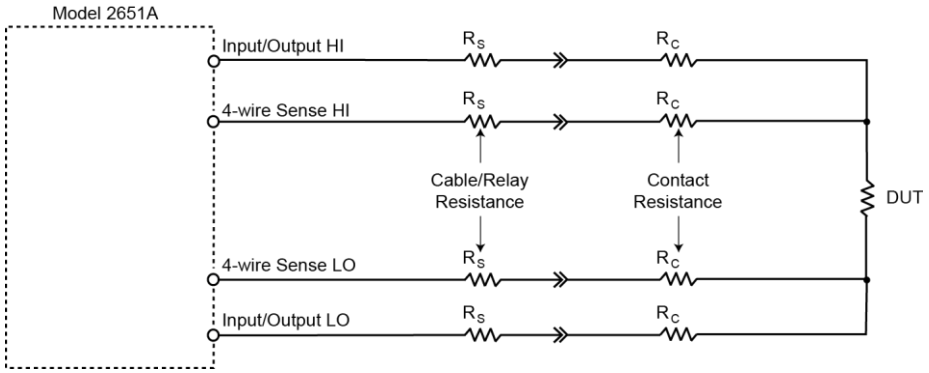
```
-- Restore Model 2651A defaults.
smua.reset()
-- Select voltage source function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Enable source autoranging.
smua.source.autorangev = smua.AUTORANGE_ON
-- Set voltage source to 5 V.
smua.source.levelv = 5
-- Set current limit to 50 mA.
smua.source.limiti = 50e-3
-- Set current range to autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Turn on the output.
smua.source.output = smua.OUTPUT_ON
-- Retrieve a power reading.
print(smua.measure.p())
-- Turn off the output.
smua.source.output = smua.OUTPUT_OFF
```

Contact check measurements

The contact check function prevents measurements that may be in error due to excessive resistance in the force or sense leads when making remotely sensed (Kelvin) measurements (see 4-wire remote sensing connections (on page 2-47) for more detail). Potential sources for this resistance include poor contact at the device under test (DUT), failing relay contacts on a switching card, and wires that are too long or thin. To use contact check, the current limit must be at least 1 mA (this allows enough current to flow when performing the test), and the source-measure unit (SMU) must not be in High-Z output-off mode.

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

Figure 11: Contact check



Contact check commands

The following table summarizes the basic contact check commands. For complete descriptions of these commands, refer to the TSP command reference (on page 7-1). For connection information, refer to Contact check connections (on page 2-49).

Basic contact check commands

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

Contact check programming example

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

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

Saved setups

You can restore the Model 2651A to one of six nonvolatile-memory setup configurations (five user setups and one factory default), or to a setup stored on an external USB flash drive. As shipped from the factory, the Model 2651A powers up with the factory default settings, which cannot be overwritten. The default settings are also in the five user setup locations, but may be overwritten. The factory default settings are listed in the command descriptions in the TSP command reference (on page 7-1).

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

Saving user setups

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

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

1. Configure the Model 2651A to the settings that you want to save.
2. Press the **MENU** key.
3. Select **SETUP** and then press the **ENTER** key.
4. Select the **SAVE** menu item and then press the **ENTER** key.
5. Select **INTERNAL** and then press the **ENTER** key.
6. Select the user number (1 through 5) and press the **ENTER** key.

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

1. Configure the Model 2651A to the settings that you want to save.
2. Insert the USB flash drive into the USB port on the front panel of the Model 2651A.
3. Press the **MENU** key.
4. Select **SETUP** and then press the **ENTER** key.
5. Select **SAVE** and then press the **ENTER** key.
6. Select **USB1**. The file name `setup000.set` is displayed.
7. Turn the navigation wheel to change the last three digits of the file name and then press the **ENTER** key.

To save and recall user setups using remote commands:

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

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

Recalling a saved setup using the front panel

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

To recall a saved setup from the front panel:

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

Start-up configuration

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

To select the power-on setup:

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

To select the power-on setup using remote commands:

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

```
setup.poweron = n
```

Where *n* is:

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

Restoring the factory default setups using remote commands

Use one of the reset functions to return the Model 2651A to the original factory defaults. An example of each type of reset is shown in the following program examples.

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

```
reset()
```

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

```
*rst
```

Restore all factory defaults:

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

Restore channel A defaults:

```
smua.reset()
```

Reset only the local TSP-Link node:

```
localnode.reset()
```

DUT test connections

NOTE

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

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

Input/output connectors

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

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

Basic connection sequence:

1. With the output off and the connector uninstalled from the Model 2651A rear panel, make the wire connections from a connector to the DUT.
2. Reinstall the connector onto the rear panel.
3. If using a screw terminal connector, tighten the two captive screws.

⚠ WARNING

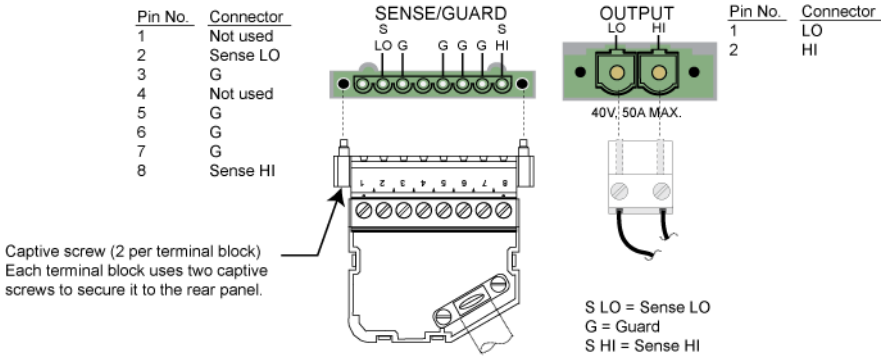
Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2651A while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of a Model 2651A before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.

Maximum floating (common mode) voltage for a SMU is 250 V. Exceeding this level could damage the instrument and create a shock hazard. See Floating a SMU (on page 2-63) later in this section for details on floating the SMUs.

The input/output connectors of the High Power System SourceMeter® Instrument are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500 V peak. Do not connect the Model 2651A terminals to CAT II, CAT III, or CAT IV circuits. Connections of the input/output connectors to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.

To prevent electric shock and/or damage to the High Power System SourceMeter® Instrument, when connecting to a source with a greater current capability than the Model 2651A, a user-supplied fuse, rated at no more than 20 A SLO-BLO, should be installed in-line with the Model 2651A input/output connectors.

Figure 12: Model 2651A input/output connectors



Input/output LO and chassis ground

WARNING

Connections to LO on the Model 2651A are not necessarily at 0 V. Hazardous voltages could exist between LO and chassis ground. Make sure that high-voltage precautions are taken throughout the test system. Alternatively, limit hazardous levels by adding external protection to limit the voltage between LO and chassis. Failure to make sure high-voltage precautions are used throughout the test system or a failure to limit hazardous levels could result in severe personal injury or death from electric shock. SMU input/output LO is available at the rear panel terminal block (labeled OUTPUT). Input/output LO is electrically isolated from chassis ground. See Rear panel (on page 2-6) for location of chassis ground connection.

NOTE

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

2-wire local sensing connections

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

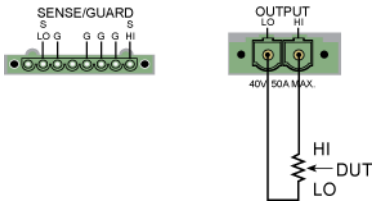
- Sourcing and measuring current.
- Sourcing and measuring voltage in high impedance (more than 1 kΩ) test circuits.

When using 2-wire local sensing connections, make sure to properly configure the Model 2651A Sense mode selection (on page 2-64).

WARNING

Guard voltage can be hazardous. With an unguarded device under test (DUT) connection, terminate the guard before the end of the cable. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

Figure 13: Two-wire resistance connections



4-wire remote sensing connections

By default, the Model 2651A instruments are configured to use 2-wire (local) voltage sensing. If you choose to enable 4-wire (remote) voltage sensing, it is critical that you establish and maintain the proper Kelvin connections between the corresponding force and sense leads to ensure the proper operation of the instrument and to make accurate voltage measurements. Sense HI must be connected to Force HI, and Sense LO must be connected to Force LO.

When sourcing voltage with remote sense, the instrument relies on the voltage detected with the sense lines to provide the proper closed-loop control of its output voltage and to properly limit the voltage across the device-under-test. If a sense line becomes disconnected from its corresponding force line, an erroneous voltage is sensed. The output voltage may be adjusted to a level that is radically different than the programmed voltage level (possibly to hazardous levels). In addition, the voltage across the device may exceed the programmed source limit voltage, possibly causing damage to the device or test fixture.

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

NOTE

You can use contact check to verify that the sense leads are connected. Refer to Contact check measurements (on page 2-40) for more information.

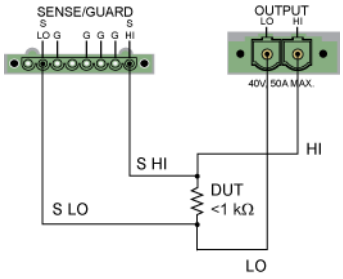
You can use 4-wire remote sensing at any time, however, it is strongly recommended for the following source-measure conditions:

- Sourcing or measuring voltage in low impedance (less than 1 kΩ) test circuits.
- Enforcing voltage compliance limit directly at the DUT.

When using 4-wire local sensing connections, make sure to properly configure the Model 2651A Sense mode selection (on page 2-64).

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

Figure 14: Model 2651A four-wire connections (remote sensing)



NOTE

You may need additional connections for redundant protective earth (safety ground) that are not shown in the preceding graphic.

⚠ WARNING

Guard voltage can be hazardous. With an unguarded device under test (DUT) connection, terminate the guard before the end of the cable. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

Contact check connections

The contact check function prevents measurement errors due to excessive resistance in the source or sense leads. See Contact check measurements (on page 2-40) for operation.

Contact check requires both source and sense connections. Refer to 4-wire remote sensing connections (on page 2-47) for the connection scheme.

Multiple SMU connections

WARNING

Connections to LO on the Model 2651A are not necessarily at 0 V. Hazardous voltages could exist between LO and chassis ground. Make sure that high-voltage precautions are taken throughout the test system. Alternatively, limit hazardous levels by adding external protection to limit the voltage between LO and chassis. Failure to make sure high-voltage precautions are used throughout the test system or a failure to limit hazardous levels could result in severe personal injury or death from electric shock.

CAUTION

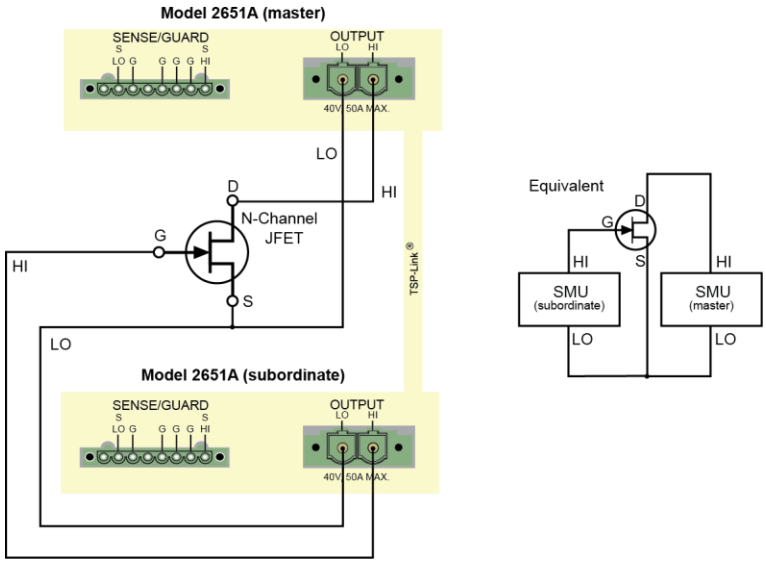
Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the Model 2651A to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other Model 2651A instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

The following figure shows how to use the SMUs of two Model 2651A instruments to test a 3-terminal device, such as an N-channel JFET (see TSP advanced features (on page 6-61) for information on using multiple Model 2651A instruments). A typical application is for the Model 263xB to source a range of gate voltages, while the Model 2651A sources voltage to the drain of the device and measures current at each gate voltage.

WARNING

Guard voltage can be hazardous. With an unguarded device under test (DUT) connection, terminate the guard before the end of the cable. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

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



The outputs of multiple SMUs can also be combined to obtain higher current and voltage levels (pulse only operation). For information including instrument configuration considerations and cautions, refer to Combining SMU outputs (on page 2-51).

Combining SMU outputs

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

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

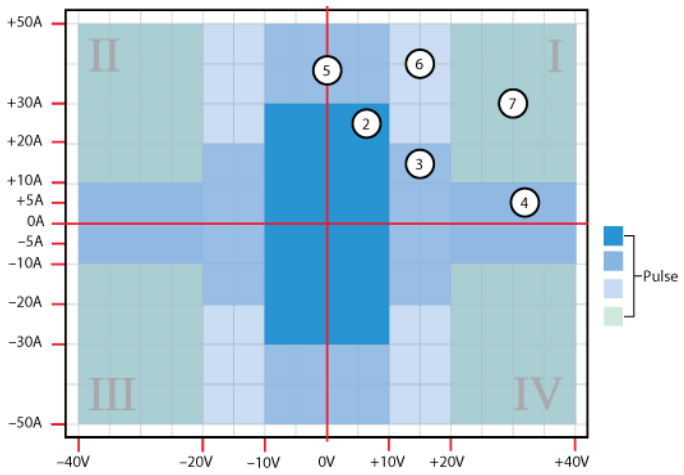
CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the Model 2651A to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other Model 2651A instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

Pulse characteristics of the Model 2651A

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

Figure 16: SMU pulse regions when combining SMUs

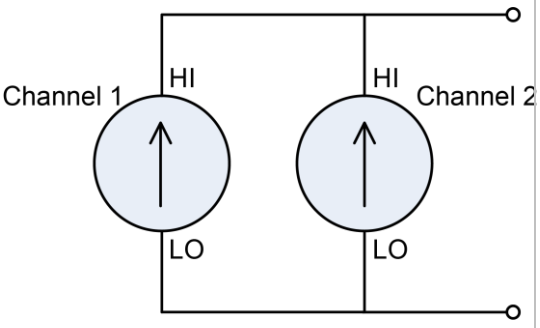


Pulse region specification			
Region (quadrant diagram)	**Region maximum**	**Maximum pulse width**	**Maximum duty cycle**
2	30 A at 10 V	1 ms	50%
3	20 A at 20 V	1.5 ms	40%
4	10 A at 40 V	1.5 ms	40%
5	50 A at 10 V	1 ms	35%
6	50 A at 20 V	330 µs	10%
7	50 A at 40 V	300 µs	1%

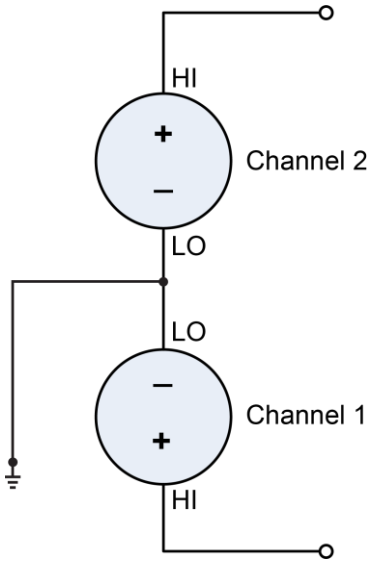
Configuration guidelines

Configuration guidelines are presented in the following table. Additional information, including examples on combining SMU channels, can be found in application notes on tek.com/keithley. Application notes include "Testing to 100A by Combining Keithley Model 2651A High Power SourceMeter Instruments."

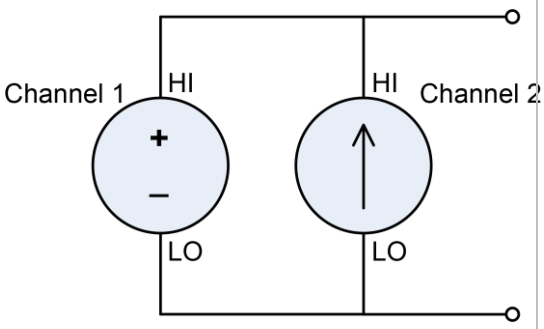
SourceMeter instrument configuration	Guidelines

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

SourceMeter instrument configuration	Guidelines

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

SourceMeter instrument configuration	Guidelines

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

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

Combining channels in series to output higher voltage

WARNING

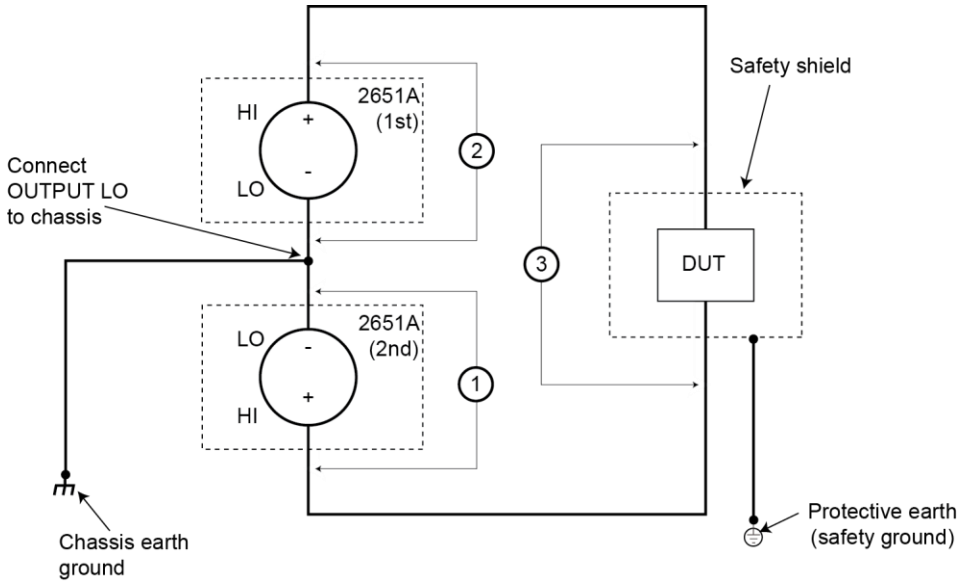
Channels in series cause hazardous voltage (>30 V_{RMS}, 42 V_{PEAK}) to be present and accessible at the Model 2651A output connector. A safety shield must be used whenever hazardous voltages will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use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

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

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

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

Figure 17: Connecting channels in series for higher voltage



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

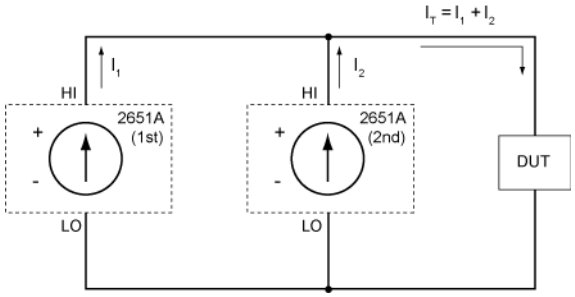
For further information, visit tek.com\keithley (tek.com/keithley) for application notes on combining SMU channels.

Combining channels in parallel to output higher current

You can output higher pulse current by connecting the channels of two Model 2651A instruments in parallel. When combining two SMU channels, make sure both SMUs have the same model number.

The figure below illustrates the connection scheme of two Model 2651A instruments connected in parallel. Two Model 2651A can output up to 100 A at 36 V (see Combining SMU outputs (on page 2-51)). The current delivered to the DUT is the sum of currents output by SMU channels (I_T). Combining the two Model 2651A instruments expands the power envelope. For further information, visit tek.com/keithley for application notes on combining the channels of two Model 2651A instruments.

Figure 18: Connecting channels in parallel for higher current



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

Guarding and shielding

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

Safety shield

WARNING

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

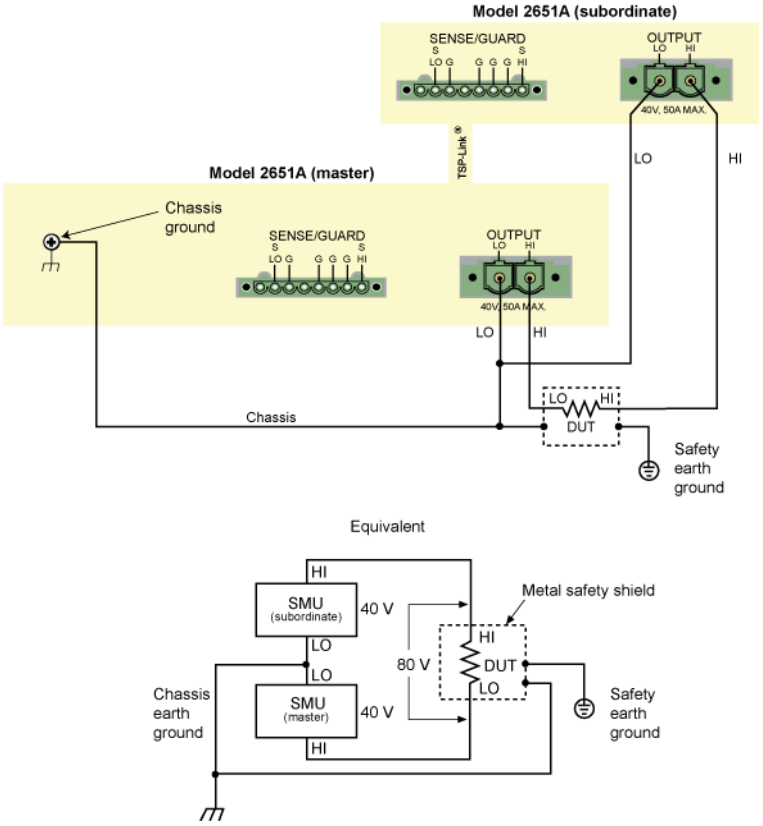
The safety shield can be metallic or nonconductive and must completely surround the device under test (DUT) test circuit. A metal safety shield must be connected to a known protective earth (safety ground). See Test fixtures (on page 2-61) for important safety information on the use of a metal or a nonconductive enclosure.

Safety shielding and hazardous voltages

The maximum output voltage for a Model 2651A channel is 40 V, which is considered a nonhazardous level. However, using two Model 2651A voltage sources in a series configuration or floating a SMU (on page 2-63) can cause test circuit voltage to exceed 42 V. In the following figure, the source-measure units (SMUs) of two Model 2651A instruments can be connected in series to apply 80 V to a device under test (DUT) (see TSP advanced features (on page 6-61) for information on using multiple Model 2651A).

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

Figure 19: Safety shield for hazardous voltage using two Model 2651A instruments



Guarding

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

Guarding (shown below) is recommended when test circuit impedance is >1 GΩ.

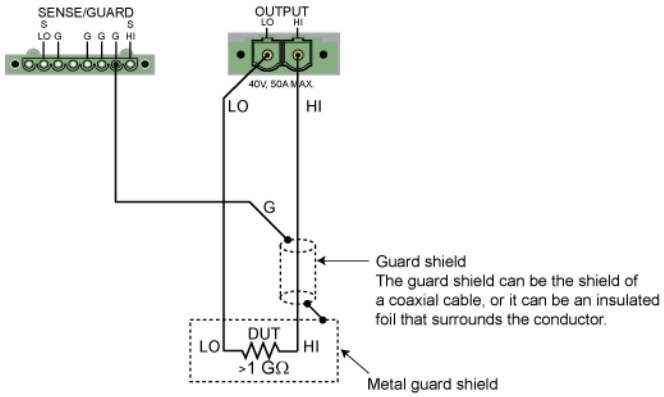
NOTE

See Guard (on page 4-23) for details on the principles of guarding.

WARNING

Guard voltage can be hazardous. With an unguarded device under test (DUT) connection, terminate the guard before the end of the cable. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

Figure 20: High-impedance guarding



Noise shield

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

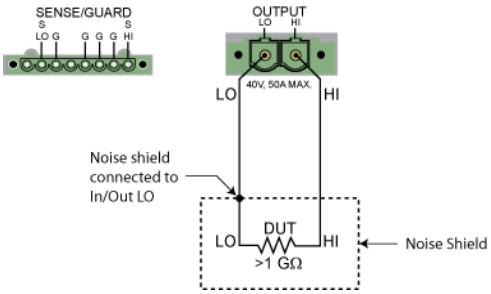
WARNING

Guard voltage can be hazardous. With an unguarded device under test (DUT) connection, terminate the guard before the end of the cable. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

WARNING

Connections to LO on the Model 2651A are not necessarily at 0 V. Hazardous voltages could exist between LO and chassis ground. Make sure that high-voltage precautions are taken throughout the test system. Alternatively, limit hazardous levels by adding external protection to limit the voltage between LO and chassis. Failure to make sure high-voltage precautions are used throughout the test system or a failure to limit hazardous levels could result in severe personal injury or death from electric shock.

Figure 21: Noise shield



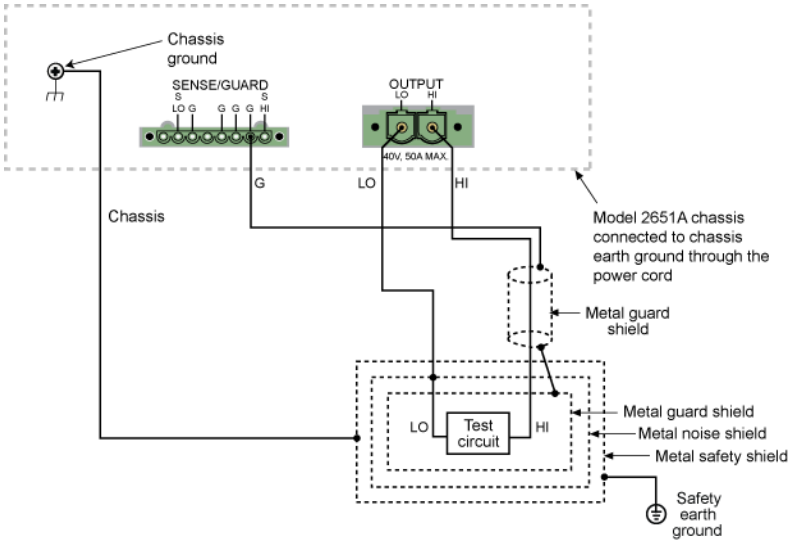
Using shielding and guarding together

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

WARNING

Guard voltage can be hazardous. With an unguarded device under test (DUT) connection, terminate the guard before the end of the cable. Connect the enclosure of all metal test fixtures to protective earth (safety ground). See your specific test fixture for information. Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system.

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



Test fixture

A test fixture can be used to house a device or test circuit. The test fixture can be a metal or nonconductive enclosure, and is typically equipped with a lid. When the test fixture is correctly connected, the output of the Model 2651A will turn off when the lid of the test fixture is opened.

You mount the test circuit inside the test fixture.

- **⚠ WARNING**

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

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

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

- **When hazardous voltages (>30 V_{RMS}, 42 V_{PEAK}) will be present, the test fixture must meet the following safety requirements:**
- **Construction material: A metal test fixture must be connected to a known protective earth (safety ground) as described in the above warning. A nonconductive test fixture must be constructed of materials that are suitable for flammability, voltage, and temperature conditions that may exist in the test circuit. The construction requirements for a nonconductive enclosure are also described in the warning above.**
- **Test circuit isolation: With the lid closed, the test fixture must completely surround the test circuit. A metal test fixture must be electrically isolated from the test circuit. Although the outer layer on a high voltage triaxial cable must be connected to the test fixture's metal chassis, the inner two layers of the cable (input/output connectors) must be isolated from the test fixture. Internally, Teflon standoffs are typically used to insulate the internal printed circuit board or guard plate for the test circuit from a metal test fixture.**
- **Interlock switch: The test fixture must have a normally-open interlock switch. The interlock switch must be installed so that when the lid of the test fixture is opened, the switch will open, and when the lid is closed, the switch will close. The Model 2651A digital I/O port provides an output enable line. When properly used with a test fixture, the output of the Model 2651A turns off when the lid of the test fixture is opened.**

WARNING

The digital I/O port of the Model 2651A is not suitable for control of safety circuits and should not be used to control a safety interlock.

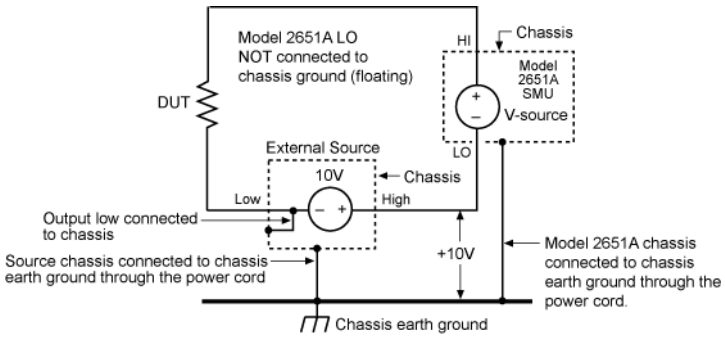
See Digital I/O (on page 3-92) for information on the digital I/O port.

Floating a SMU

Using an external source in the test system may require that a Model 2651A source-measure unit (SMU) float off chassis earth ground. An example of such a test system is shown below, which includes an external voltage source. Notice that output LO of the external voltage source is connected to chassis earth ground.

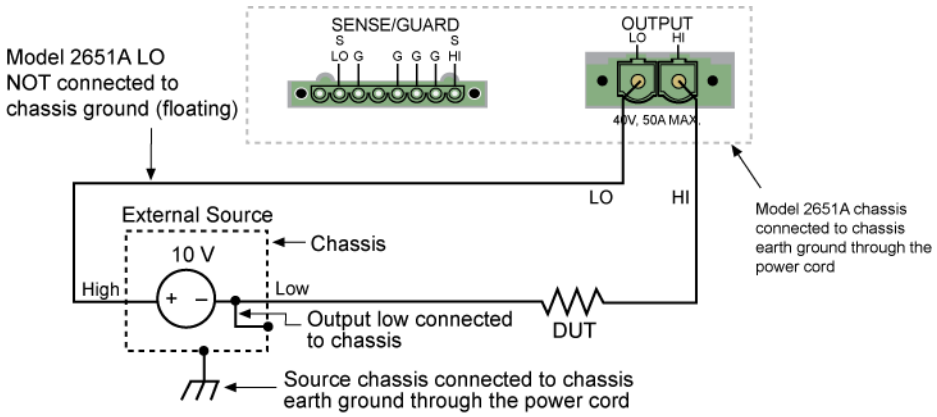
For the test circuit shown below, the Model 2651A must float off chassis earth ground. As shown, LO of the Model 2651A is floating +10 V above chassis earth ground. If LO of the Model 2651A was instead connected to chassis ground, the external voltage source would be shorted to the chassis ground.

Figure 23: Floating the Model 2651A schematic



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

Figure 24: Floating the 2651A SMU connections



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

⚠ WARNING

The maximum floating (common mode) voltage for a SMU is ±250 V. Exceeding this level may cause damage to the instrument and create a shock hazard.

Using an external source to float a SMU could create a shock hazard in the test circuit. A shock hazard exists whenever >42 V peak is present in the test circuit. Appropriately rated cables or insulators must be provided for all connections to prevent access to live parts.

When >42 V is present, the test circuit must be insulated for the voltage used or surrounded by a metal safety shield that is connected to a known protective earth (safety ground) and chassis ground (see Safety shield (on page 2-57)).

DUT connection settings

Make sure to properly configure the Model 2651A sense mode for the specific DUT test connection scheme. Use care to configure both the output-off state protection settings to supplement safe operation of your test setup.

Sense mode selection

You can set the sense mode to use 2-wire local sensing connections (on page 2-47) or 4-wire remote sensing connections (on page 2-47). The default sense setting is 2-wire local.

Front-panel sense mode selection

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

1. Press the **CONFIG** key.
2. Press the **SRC** or **MEAS** key. You can access and set the Model 2651A sense mode from either the V-SOURCE or the V-MEAS menu items.
3. If you pressed the SRC key: Select **V-SOURCE** > **SENSE-MODE**, and then press the **ENTER** key or the navigation wheel.

   If you pressed the MEAS key: Select **V-MEAS** > **SENSE-MODE**, and then press the **ENTER** key or the navigation wheel.
4. Select **2-WIRE** or **4-WIRE** as needed, and then press the **ENTER** key or the navigation wheel.

Selecting the sense from a remote interface

To select the remote sense from a remote interface:

Set the `smua.sense` attribute to control the sense state by remote. The programming example below illustrates how to configure the Model 2651A for 4-wire remote sensing:

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

See Remote source-measure commands (on page 2-29) and TSP command reference (on page 7-1) for details.

Output-off states

CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the Model 2651A to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other Model 2651A instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

Output-off modes

Turning off the Model 2651A output may not completely isolate the instrument from the external circuit. You can use the output-off mode to place the Model 2651A in a noninteractive state during idle periods. The available output-off modes are normal, high-impedance, and zero.

Normal output-off mode

The normal output-off mode is the default output-off mode setting. When the source-measure unit (SMU) is in the normal output-off mode, you can select either the current or the voltage output-off function (see Output-off function (on page 2-67)). You can also specify current and voltage output-off limits (Output-off limits (compliance) (on page 2-68)).

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

High-impedance output-off mode

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

Zero output-off mode

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

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

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

You can use the Model 2651A as a current meter when it is in zero output-off mode because it outputs 0 V but measures current.

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

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

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

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

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

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

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

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

Output-off function

This setting is used only when the output is turned off and the Model 2651A is set to the normal output-off mode (`smua.source.offmode = smua.OUTPUT_NORMAL`).

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

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

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

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

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

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

Selecting the output-off function

NOTE

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

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

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

To configure the output-off function remotely:

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

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

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

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

Output-off limits (compliance)

You can set output-off limits (compliance) for the current and voltage output-off functions using the CONFIG menu on the Model 2651A front panel, or by setting the `smua.source.offlimitY` attribute from a remote interface. The output-off limits only apply when the output-off mode is normal.

Setting output-off limits

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

To configure output-off limits from the front panel:

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

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

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

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

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

Remote programming output-off states quick reference

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

Output-off state programming quick reference

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

USB storage overview

The Model 2651A includes a USB port on the front panel. To store scripts and to transfer files from the instrument to the host computer, you need a USB flash drive.

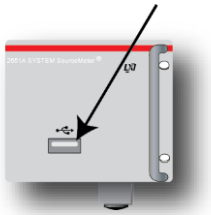
- For information about saving reading buffers to a USB flash drive, see Saving reading buffers (on page 3-9).
- For information about storing and loading scripts to and from a USB flash drive, see Save a user script (on page 6-11).
- For information about file I/O, see File I/O (on page 5-7).
- For information about saving user setups, see Saved setups (on page 2-42).

Connecting the USB flash drive

The Model 2651A supports flash drives that comply with USB 2.0 standards (as well as USB 1.0 and 1.1 standards). You can save data to the USB flash drive from the front panel, or you can create a script to save data to the USB flash drive.

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

Figure 25: USB port



File system navigation

The Model 2651A can use commands from the Lua `fs` library to navigate and list files that are available on a flash drive. These Lua commands are in the `fs` command group in the instrument.

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

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

```
"/usb1/"
```

NOTE

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

The instrument supports the following Lua `fs` commands:

fs.chdir() (on page 7-114)
fs.cwd() (on page 7-115)
fs.is_dir() (on page 7-116)
fs.is_file() (on page 7-117)
fs.mkdir() (on page 7-118)
fs.readdir() (on page 7-119)
fs.rmdir() (on page 7-120)

The following Lua `fs` commands are not supported:

```
fs.chmod()
fs.chown()
fs.stat()
```

Displayed error and status messages

During operation and programming, front-panel messages may be briefly displayed. Typical messages are either status or error notifications (refer to the Error summary list (on page 8-3) for a complete list of these messages and their meanings).

NOTE

Status and error messages are held in a queue. For information about retrieving messages from queues, refer to Queues (on page 15-2). For information about error messages, refer to the Troubleshooting guide (on page 8-1).

Range

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

Available ranges

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

Model 2651A source and measurement ranges

Voltage ranges	Current ranges
100 mV 1 V 10 V 20 V 40 V	1 µA 10 µA 100 µA 1 mA 10 mA 100 mA 1 A 5 A* 10 A* 20 A* 50 A**
* Refer to Operating boundaries (on page 4-5) for power derating information. ** 50 A range accessible only in pulse mode.	

Maximum source values and readings

The full-scale output for each voltage and current source range is 101 percent of the selected range, but the full-scale measurement is 102 percent of the range. For example, ±1.01 A is the full-scale source value for the 5 A range, and ±102 mA is the full-scale reading for the 100 mA measurement range. Input levels that exceed the maximum levels cause the overflow message to be displayed. Note, however, that the instrument will autorange at 100 percent of the range.

Measure autodelay

The measure delay is a specific delay that is applied before each measurement is made. This delay is disabled by default (measurements are made immediately). You can change the default delay by setting the smuX.measure.delay (on page 7-248) attribute either to a specific value or to an autodelay setting (set `smua.measure.delay = smua.DELAY_AUTO`). If the measure delay is set to the autodelay setting, a range-dependent delay is applied each time the instrument performs a current measurement. This delay also happens for the measurement that is made after changing current ranges during an autoranged measurement. The following table contains the measure autodelays associated with each current range.

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

You can increase or decrease the autodelay by changing the delay factor (for example, to reduce the delay across all ranges by half, set `smua.measure.delayfactor = 0.5`). For additional information, refer to smuX.measure.delayfactor (on page 7-249).

Ranging limitations

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

The maximum output power and source/sink limit for the Model 2651A is 202 W per channel (maximum). With the 40 V (V-Source) range selected, the highest current measurement range is 5 A. Refer to Operating boundaries (on page 4-5) for power derating information. When pulsing with the 50 A I-Source range selected, you can be at a larger voltage range than 4 V dependent on your duty cycle. See Pulse duty cycle (on page 3-30) for details.

Manual ranging

Use the range keys, 

and 

, to select a fixed range:

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

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

Autoranging

To use automatic source ranging, press **SRC** then the **AUTO** range key.

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

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

NOTE

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

Low range limits

The low range limit sets the lowest range that the Model 2651A uses when autoranging is enabled. This feature is useful for minimizing autorange settling times when measurements require numerous range changes.

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

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

Range considerations

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

If...	Then...	Notes

The source function is the same as the measurement function (for example, sourcing voltage and measuring voltage)	The measurement range is locked to be the same as the source range.	The setting for the voltage measure range is retained and used when the source function is changed to current.
		Model 2651A example:
		`smua.source.func = smua.OUTPUT_DCVOLTS`
		`smua.source.rangev = 1`
		`smua.measure.rangev = 10`
		`-- will print 1, the source range`
		`print(smua.measure.rangev)`
		`smua.source.func = smua.OUTPUT_DCAMPS`
		`-- will print 10, the measure range`
		`print(smua.measure.rangev)`

A source or measurement range for a function is explicitly set	Autoranging for that function is disabled.	Autoranging is controlled separately for each source and measurement function: source voltage, source current, measure voltage, and measure current. Autoranging is enabled for all four by default.
Source autoranging is enabled	The output level controls the range.	Querying the range after the level is set returns the range the instrument chose as appropriate.
You send a source level that is out of range while autorange is off (an example of this is sending 1 A on the 100 mA range)	The instrument will not return an error until the output is turned on.	When the output is turned on, the display will show a series of question marks: `???.???`
Measure autoranging is enabled	The measure range is changed only when a measurement is taken.	Querying the range after the measurement is taken will return the range that the instrument chose.

Range programming

Range commands

The following tables summarize commands necessary to control measure and source ranges. See the TSP command reference (on page 7-1) for more details about these commands.

Measure range commands*

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

* See Available ranges (on page 2-71)

Source range and limit commands*

Command	Description
`smua.source.autorangei = smua.AUTORANGE_ON`	Enable current source autorange.
`smua.source.autorangei = smua.AUTORANGE_OFF`	Disable current source autorange.
`smua.source.autorangev = smua.AUTORANGE_ON`	Enable voltage source autorange.
`smua.source.autorangev = smua.AUTORANGE_OFF`	Disable voltage source autorange.
`smua.source.limiti = level`	Set voltage source current limit (compliance).
`smua.source.limitv = level`	Set current source voltage limit (compliance).
`smua.source.limitp = level`	Set source power limit (compliance).
`smua.source.lowrangei = lowrange`	Set lowest current source range for autorange.
`smua.source.lowrangev = lowrange`	Set lowest voltage source range for autorange.
`smua.source.rangei = rangeval`	Select manual current source range.
`smua.source.rangev = rangeval`	Select manual voltage source range.

* See Available ranges (on page 2-71)

Range programming example

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

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

```
-- Restore Model 2651A defaults.
smua.reset()
-- Set V source range to auto.
smua.source.autorangev = smua.AUTORANGE_ON
-- Select 10 mA measure range.
smua.measure.rangei = 10e-3
-- Set limit level to 10 mA.
smua.source.limiti = 10e-3
```

Digits

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

The DIGITS setting has no effect on the format of readings returned by a `print()` command over a remote interface. To adjust the format of remote interface readings, see format.asciiprecision (on page 7-111).

The number of displayed digits does not affect accuracy or speed. Accuracy and speed are controlled by the SPEED setting (see Speed (on page 2-78)).

Setting display resolution from the front panel

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

Setting display resolution from a remote interface

The following table summarizes use of the `display.smua.digits` command. See the TSP command reference (on page 7-1) for more information.

Digits commands

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

Digits programming example

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

Speed

The Model 2651A has two analog-to-digital converters (ADCs): one integrating, one high speed. Fastest reading rates are achieved by selecting the fast ADC. You can select the fast ADC by pressing the SPEED key and choosing the FAST option.

You can also use the SPEED key to select the integrating ADC and set the period of time the input signal is measured (this is called the integration time or measurement aperture). The integration time affects the usable digits, the amount of reading noise, and the reading rate of the instrument. The integration time is specified in parameters based on the number of power line cycles (NPLC), where 1 PLC for 60 Hz is 16.67 ms (1/60) and 1 PLC for 50 Hz is 20 ms (1/50).

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

NOTE

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

Setting the speed from the front panel

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

- **FAST:** Selects the fast analog-to-digital converter (ADC) (fast performance, but accuracy is reduced)
- **MED:** Selects the integrating ADC and sets its measurement aperture to 0.10 PLC (speed and accuracy are balanced)
- **NORMAL:** Selects the integrating ADC and sets its measurement aperture to 1.00 PLC (speed and accuracy are balanced)
- **HI-ACCURACY:** Selects the integrating ADC and sets its measurement aperture to 10.00 PLC (high accuracy, but speed is reduced)
- **OTHER:** Selects the integrating ADC and sets its measurement aperture to any PLC value from 0.001 to 25

NOTE

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

Setting the speed using the remote interface

The following table summarizes commands to control speed. See TSP command reference (on page 7-1) for more information.

Speed commands

Command	Description
`smua.measure.nplc = nplc`	Sets the speed of the integrating ADC only (`nplc` can range from 0.001 to 25).
`smua.measure.adc = adc`	Selects analog-to-digital converter (`adc = smua.ADC_INTEGRATE` or `smua.ADC_FAST`).

Speed programming example

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

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

Sampling speed programming example

Use the measure interval to control the sampling speed of the fast ADC. The programming example below illustrates how to configure the fast ADC to take 1000 measurements with 10 µs sampling interval:

```
-- Select fast ADC.
smua.measure.adc = smua.ADC_FAST
-- Set measure count to 1000 readings.
smua.measure.count = 1000
-- Set sampling interval to 10 us (microseconds).
smua.measure.interval = 10e-6
```

Remote communications interfaces

You can choose from one of several communication interfaces to send commands to and receive responses from the Model 2651A.

You can control the Model 2651A from only one communications interface at a time. The first interface on which the instrument receives a message takes control of the instrument. If another interface sends a message, that interface can take control of the instrument. You may need to enter a password to change the interface, depending on the setting of interface access.

The Model 2651A automatically detects the type of communications interface (LAN, USB, GPIB, or RS-232) when you connect to the respective port on the rear panel of the instrument. In most cases, you do not need to configure anything on the instrument. In addition, you do not need to reboot if you change the type of interface that is connected.

Supported remote interfaces

The Model 2651A supports the following remote interfaces:

- GPIB. General purpose interface bus is an IEEE-488 instrumentation data bus.
- LAN. Local area network (LAN) communications provide the flexibility to build scalable and functional test or data acquisition systems with a large degree of flexibility.
- RS-232

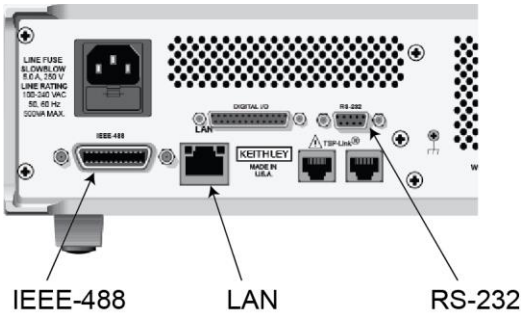
NOTE

The Model 2651A can be controlled from only one communication interface at a time. The first interface from which it receives a message takes control of the instrument. It ignores the other interfaces until the instrument is returned to local operation.

For more information about the remote interfaces, see:

- GPIB setup (on page 2-83)
- LAN concepts and settings (on page 2-81, on page 13-1)
- RS-232 interface operation (on page 2-90)

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



Output queue

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

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

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

LAN communications

The Model 2651A is an LXI version 1.4 Core 2011 compliant instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet). The LAN port on the rear panel of the Model 2651A supports full connectivity on a 10 Mbps or 100 Mbps network.

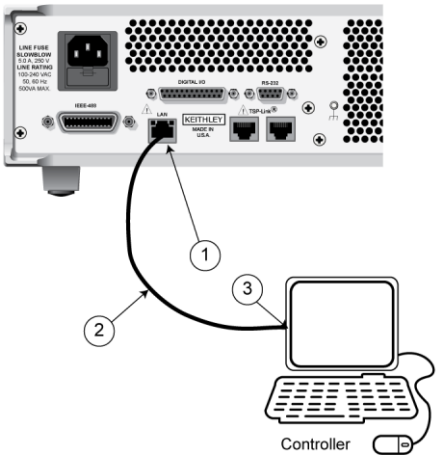
For detailed information about setting up your LAN interface, refer to LAN concepts and settings (on page 2-81, on page 13-1).

LAN cable connection

The Model 2651A includes a LAN crossover cable. You can use this cable for the TSP-Link® network or LAN communications.

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

Figure 27: LAN connection

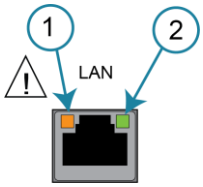


1 Model 2651A ethernet port (LAN)
2 Straight-through LAN cable or crossover LAN cable
3 Ethernet port (on the host computer)

LAN status LEDs

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

Figure 28: LAN status



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

Monitoring the LAN

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

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

- lan.autoconnect (on page 7-139)
- lan.linktimeout (on page 7-149)

GPIB operation

The following topics contain information about GPIB standards, bus connections, and primary address selection.

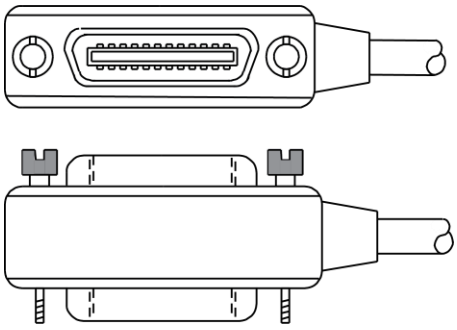
GPIB standards

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

Connect the GPIB cable

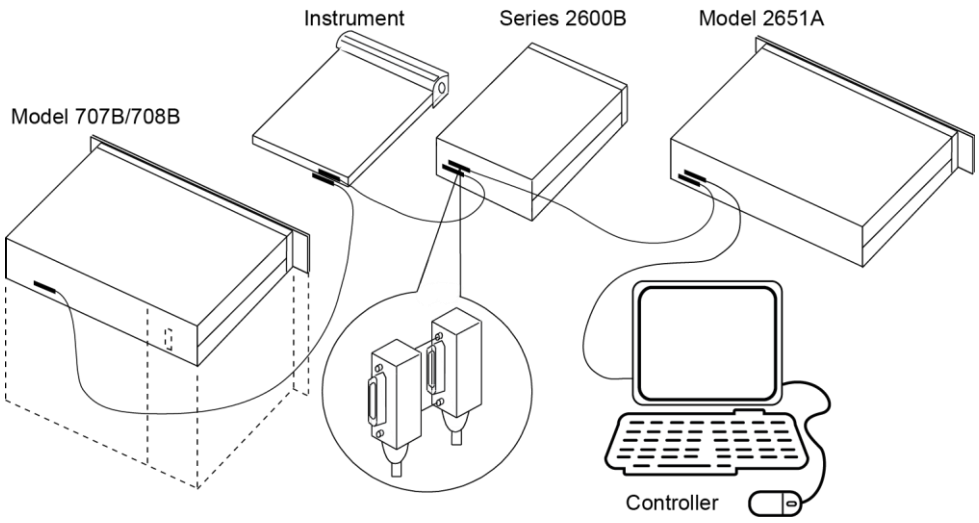
To connect an instrument to the GPIB bus, use a cable equipped with standard IEEE-488 connectors, as shown below.

Figure 29: GPIB connector



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

Figure 30: IEEE-488 connections

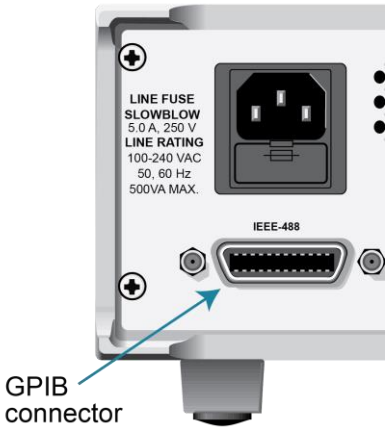


CAUTION

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

To connect the instrument to the IEEE-488 bus, line up the cable connector with the connector on the rear panel. Install and tighten the screws securely, making sure not to overtighten them. The following figure shows the location of the connector.

Figure 31: GPIB connector location



Connect any additional connectors from other instruments as required for your application. Make sure the other end of the cable is properly connected to the controller. You can have up to 15 devices connected to a GPIB interface, including the controller. The maximum cable length is the lesser of either:

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

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

Primary address

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

To set or check the primary address from the front panel:

1. Press the **MENU** key, select **GPIB**, and then press the **ENTER** key or the navigation wheel.
2. Select **ADDRESS**, then press the **ENTER** key or the navigation wheel.
3. Use the navigation wheel to set the primary address to the appropriate value, then press the **ENTER** key or the navigation wheel.
4. Press the **EXIT (LOCAL)** key twice to return to the normal display.

To set the primary address remotely:

```
gpib.address = address
```

To set the primary address remotely to 20:

```
gpib.address = 20
```

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

GPIB terminator

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

General bus commands

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

General bus commands

Command	Effect on Model 2651A
DCL	Returns the Model 2651A and all devices on the GPIB to known conditions. See DCL (on page 2-87) for details.
GET	Initiates a trigger. See GET (on page 2-87) for details.
GTL	Cancel remote; restore Model 2651A front-panel operation. See GTL (on page 2-87) for details.
IFC	Goes into talker and listener idle states. See IFC (on page 2-86) for details.
LLO	LOCAL key locked out. See LLO (on page 2-86) for details.
REN	Goes into remote operation when next addressed to listen. See REN (on page 2-86) for details.
SDC	Returns the Model 2651A to known conditions. See SDC (on page 2-87) for details.
SPE, SPD	Serial polls the Model 2651A. See SPE, SPD (on page 2-88) for details.

REN

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

IFC

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

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

LLO

When the instrument is in remote operation, all front-panel controls are disabled, except the LOCAL and OUTPUT OFF keys (and the POWER switch). The local lockout (LLO) command disables the LOCAL key, but does not affect the OUTPUT OFF switch, which cannot be disabled.

GTL

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

DCL

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

When the Model 2651A receives a DCL command, it:

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

The DCL command does not affect instrument settings and stored data.

SDC

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

When the Model 2651A receives an SDC command, it:

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

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

GET

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

SPE, SPD

Use the serial polling sequence to obtain the Model 2651A serial poll byte. The serial poll byte contains important information about internal functions (see Status model (on page 15-1)). Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line. The serial polling sequence may be performed at any time to obtain the status byte from the Model 2651A.

Front-panel GPIB operation

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

Error and status messages

The front-panel display may show error and status messages (see Displayed error and status messages (on page 2-71)). See Error summary list (on page 8-3) for a list of status and error messages that are associated with IEEE-488 programming.

Communication status indicators

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

Status indicator	Applies to
REM	GPIB, VXI-11, USB, RS-232
TALK	GPIB only
LSTN	GPIB only
SRQ	GPIB, VXI-11, USB

NOTE

The SRQ applies to all available communication buses, however, actual service requests only apply to GPIB, USB, and VXI-11 (see Status byte and service request (SRQ) (on page 15-15) for more information).

REM

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

TALK

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

LSTN

This indicator is on when the instrument is in the listener active state, which is activated by addressing the instrument to listen with the correct listen command. LSTN is off when the instrument is in the listener idle state. Place the instrument in the listener idle state by sending UNL (unlisten), addressing it to talk, or by sending the IFC (interface clear) command over the bus.

SRQ

You can program the instrument to generate a service request (SRQ) when one or more errors or conditions occur. When this indicator is on, a service request was generated. This indicator stays on until all conditions that caused the SRQ are cleared.

Note that while the SRQ indicator turns on when a service request is generated, it reflects the state of the master summary status (MSS) bit and not the request for service (RQS) bit. Therefore, performing a serial poll does not turn off the indicator. To turn off the indicator, you must use `*CLS` or `status.reset()` to clear all the conditions that caused the MSS bit to be set.

NOTE

The SRQ applies to all available communication buses. However, actual service requests only apply to GPIB, USB, and VXI-11.

For additional information on using the SRQ, refer to Status byte and service request (SRQ) (on page 15-15).

LOCAL key

The EXIT (LOCAL) key cancels the remote state and restores local operation of the instrument. Pressing the EXIT (LOCAL) key turns off the REM indicator and returns the display to normal if a user-defined message was displayed. Pressing the EXIT (LOCAL) key or the OUTPUT ON/OFF control also aborts any commands or scripts that are being processed.

If the LLO (local lockout setting) command is in effect, the EXIT (LOCAL) key is inoperative. For safety reasons, you can use the OUTPUT ON/OFF control to turn the output off while in LLO.

RS-232 interface operation

The following topics contain information about configuring RS-232 communication parameters, sending or receiving command messages, and requesting or retrieving data. To control the Model 2651A, connect a controller or personal computer to the Model 2651A RS-232 interface. Alternatively, you can use the Model 2651A to control another device over RS-232.

Setting RS-232 interface parameters

To set interface parameters from the front panel:

1. Press the **MENU** key, select **RS232**, and then press the **ENTER** key or the navigation wheel.
2. Select and enter the following interface parameters:
   - **BAUD:** Set baud rate (see Baud rate (on page 2-91))
   - **BITS:** Set number of bits (see Data bits and parity (on page 2-91))
   - **PARITY:** Set parity
   - **FLOW-CTRL:** Set Flow control and signal handshaking (on page 2-92)
   - **ENABLE:** Enable or disable the RS-232 interface
3. Press the **EXIT (LOCAL)** key twice to return to the normal display.

Remote RS-232 parameters

Commands to set RS-232 parameters are listed in the following table. See the TSP command reference (on page 7-1) for more information.

RS-232 interface commands

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

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

RS-232 programming example

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

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

Sending and receiving data

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

RS-232 terminator

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

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

Baud rate

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

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

The factory-selected baud rate is 9600.

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

Data bits and parity

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

Flow control and signal handshaking

Signal handshaking between the controller and the instrument allows the two devices to communicate to each other to determine if they are ready to receive data.

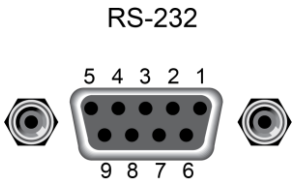
The RS-232 interface provides two control lines (request to send and clear to send) for this purpose. The instrument asserts the RTS signal when it is admissible for the computer to transmit to the instrument. It sends information to the computer when the CTS signal is asserted by the computer.

RS-232 connections

Connect the RS-232 serial port of the Model 2651A to the serial port of a computer using a straight-through RS-232 cable terminated with DB-9 connectors. Do not use a null modem cable. The serial port uses the transmit (TXD), receive (RXD), CTS and RTS (if flow control is enabled), and signal ground (GND) lines of the RS-232 standard. The connector location is shown in Remote communications interfaces (on page 2-80).

If your computer uses a DB-25 connector for the RS-232 interface, you need a standard cable or adapter with a DB-25 connector on one end and a DB-9 connector on the other.

Figure 32: RS-232 interface connector



RS-232 connector pinout

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

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

Computer serial port pinout

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

* The Model 2651A does not use all RS-232 signals. See Flow control and signal handshaking (on page 2-92).

Section 3

Functions and features

In this section:

Relative offset

When making measurements, you may want to subtract an offset value from a measurement.

The relative offset feature subtracts a set value or a baseline reading from measurement readings. When you enable relative offset, all measurements are recorded as the difference between the actual measured value and the relative offset value. The formula to calculate the offset value is:

Displayed value = Actual measured value − Relative offset value

When a relative offset value is established for a measure function, the value is the same for all ranges for that measure function. For example, if 0.5 A is set as a relative offset value on the 1 A range, the relative offset value is also 0.5 A on the lower current ranges.
Selecting a range that cannot accommodate the rel value does not cause an overflow condition, but it also does not increase the maximum allowable input for that range. For example, on the 1 A range, the Model 2651A still overflows for a more than 1.02 A input.

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

Enabling and disabling relative offset from the front panel

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

Defining a relative offset value from the front panel

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

To establish a relative offset value from the front panel:

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

Relative offset commands

Relative offset commands are summarized in the following table.

Relative offset commands

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

Relative offset programming example

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

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

Filters

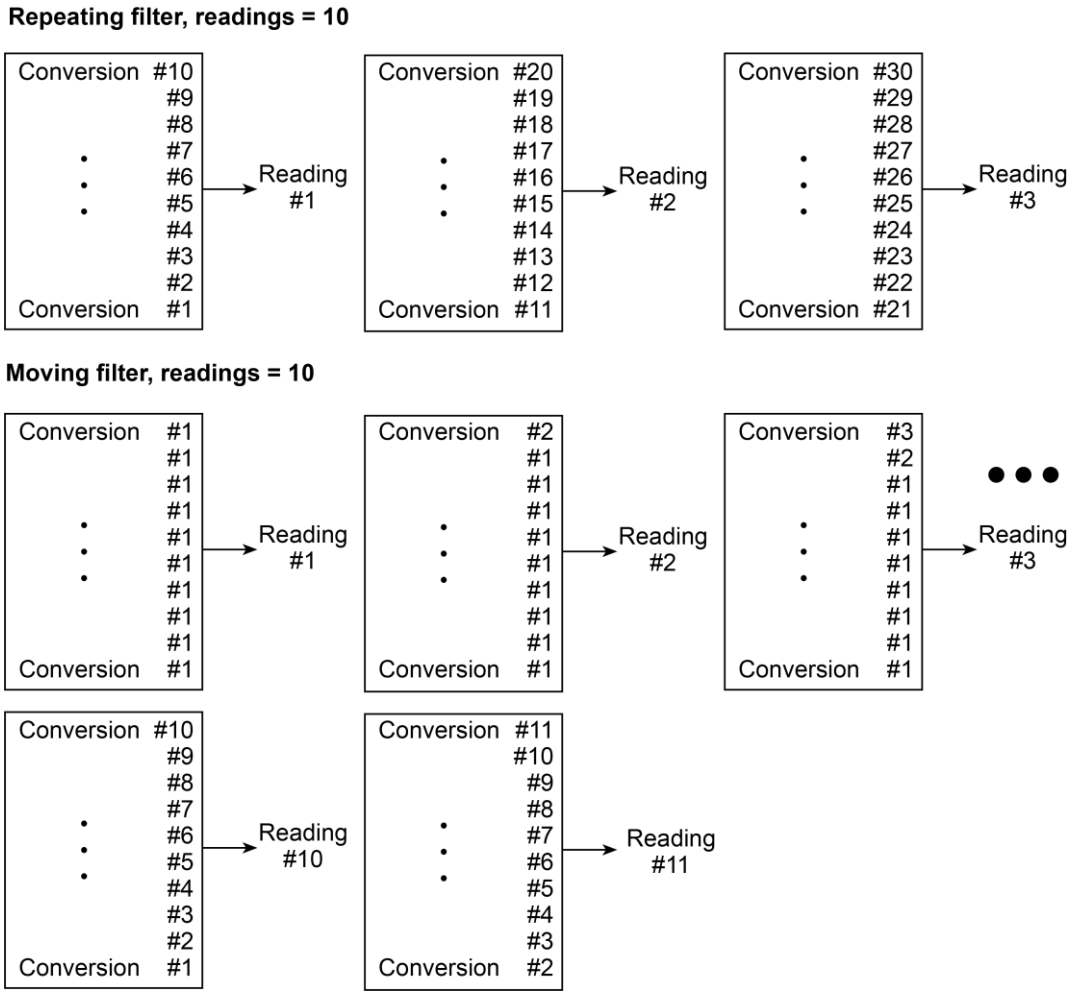
The filter feature lets you set the filter response to stabilize noisy measurements. The Model 2651A uses a digital filter, which is based on reading conversions. The displayed, stored, or transmitted reading is calculated using one or more reading conversions (from 1 to 100).

Filter types

The Model 2651A provides two averaging filters and a median filter. The power-on default is the repeating filter.

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

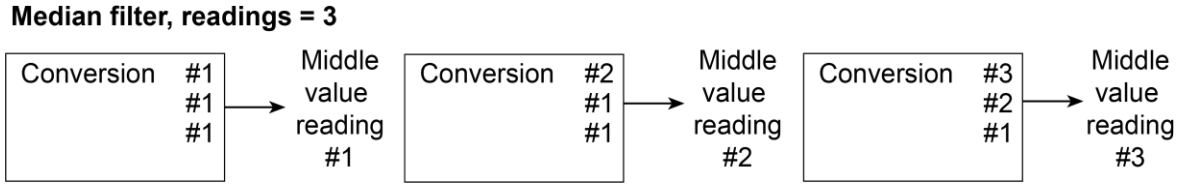
Figure 33: Repeating and moving average filters



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

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

Figure 34: Median filter



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

Response time

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

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

Enabling the filter from the front panel

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

Configuring the filter from the front panel

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

To configure the filter:

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

Setting the filter using a remote interface

The following table summarizes the filter commands. See the TSP command reference (on page 7-1) for details about commands.

Filter commands

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

Filter programming example

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

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

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

Reading buffers

Reading buffers capture measurements, ranges, instrument status, and output state of the Keithley Instruments Model 2651A. The Model 2651A has two default reading buffers. In addition to the default buffers, you can create user-defined reading buffers. You can use the reading buffers to acquire readings.

You can access reading buffers from the front panel or over the remote command interface.

The default reading buffers can store more than 60,000 readings if you enable the options for timestamps and source values. To store 140,000 readings internally, you can disable the timestamps and source values.

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

Once you save the reading buffers to a USB flash drive, insert the USB flash drive into the USB port on your computer to view the data in any compatible data analysis application or to transfer the data from the USB flash drive to your computer.

Front-panel reading buffer control

The dedicated reading buffers can be configured, stored, and recalled when in local mode operation. Use the front panel to navigate and configure the reading buffers options and to save and recall stored readings. See the Configuration menus (on page 2-15) (CHANA-BUF and CHANB-BUF) for information on reading buffer configuration.

Reading buffer options

The following list outlines the menu structure and menu items associated with front-panel reading buffer control. This section provides a description for each reading buffer option. Use the procedure in Configuring reading buffers (on page 3-7) as a guideline to configure these reading buffer options.

- **DEST**: Sets data storage destination (buffer 1, buffer 2, or NONE).
- **BUFFER1**: Configure buffer 1.
  - **CLEAR**: Clear buffer (YES or NO).
  - **ELEMENTS**: Enable (ON) or disable (OFF) data storage elements:
    - **SRC-VAL**: Enable or disable source values.
    - **TSTAMP**: Enable or disable timestamps.

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

Configuring reading buffers

To configure reading buffers from the front panel:

NOTE

Enabling or disabling the source value or the timestamp is optional.

1. Press the **CONFIG** key.
2. Select **STORE** > **CHANA-BUFF** > **DEST** and then choose one of the following:
   - **CHANA-BUFF1**
   - **CHANA-BUFF2**
   - **NONE**
3. Select **BUFFER1** or **BUFFER2**.
4. Clear the buffer by turning the navigation wheel to select **CLEAR > YES**.
5. Turn the navigation wheel to highlight **ELEMENTS**, and then press the navigation wheel (or the **ENTER** key).

NOTE

You must clear the reading buffer before you can enable or disable the source value or the timestamp options.

6. Configure the timestamp elements of the reading buffer:
   a. Turn the navigation wheel to highlight **TSTAMP**.
   b. Press the navigation wheel (or the **ENTER** key).
   c. Select **OFF** or **ON** and then press the navigation wheel (or the **ENTER** key).
7. Configure the source value elements of the reading buffer:
   a. Turn the navigation wheel to highlight **SRC-VAL**.
   b. Press the navigation wheel (or the **ENTER** key).
   c. Select **OFF** or **ON**.
8. Press the **EXIT (LOCAL)** key to return to the main menu.

Appending or overwriting existing reading buffers

When storing data to a reading buffer that already holds data, the new data can be appended to the reading buffer data, or it can overwrite the old data.

To configure the instrument to append or overwrite measurements the next time data is acquired:

1. Press the **CONFIG** key.
2. Select **STORE**, and then select **STORAGE-MODE**. The Storage Mode menu is shown.
3. Select one of the following:
   - **APPEND**
   - **OVERWRITE**
4. Press the **EXIT (LOCAL)** key to return to the main menu.

Storage operation

Use this option to initiate a storage operation and to configure the number of readings to acquire during the storage operation. The reading count can be more than 60,000 if timestamps and source values are enabled. The count can be more than 140,000 if timestamps and source values are disabled.

NOTE

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

To specify the number of readings and initiate the storing operation:

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

NOTE

If the output-off mode is ZERO or the output is already on, the instrument starts acquiring readings when the ENTER key is pressed. Otherwise, the instrument starts acquiring readings when the output is turned on.

Saving reading buffers

You can save the dedicated reading buffers to nonvolatile memory or you can save them to a USB flash drive.

The instrument restores the dedicated reading buffers from internal nonvolatile memory when the instrument is turned off and back on.

NOTE

You can also save reading buffer data to a .csv file using the web interface. Refer to Download reading buffer data using the web interface.

Saving the reading buffers to nonvolatile memory

After the measurements are complete, you can save the reading buffer data to the nonvolatile memory in the instrument.

To save the reading buffer data:

1. From the front panel, press the **STORE** key, and then select **SAVE**.
2. Select **INTERNAL** to save to internal nonvolatile memory.
3. Select one of the following:
   - **SMUA_BUFFER1**
   - **SMUA_BUFFER2**
4. The front panel displays `Saving... This may take awhile`.
5. Press the **EXIT (LOCAL)** key to return to the main menu.

Saving the reading buffer to a USB flash drive

After the measurements are complete, you can save the reading buffer data to a USB flash drive.

To save the reading buffer data to a USB flash drive:

1. Insert the USB flash drive into the USB port.
2. Press the **STORE** key and use the navigation wheel to select **SAVE**.
3. Select **USB1**.
4. Select one of the following file formats:
   - **CSV**
   - **XML**
5. Use the navigation wheel to select the reading buffer.
6. Use the navigation wheel to change the file name.
7. Press the navigation wheel or the **ENTER** key to save the file.
8. Press the **EXIT (LOCAL)** key to return to the main menu.

Recalling readings

To recall the data stored in a reading buffer:

1. Press the **RECALL** key.
2. Select **DATA** or **STATISTICS**.
3. Select the buffer to display. The data or statistics are displayed.
   - If you are recalling data, the reading display is on the top left, and the buffer location number is on the right. The source values are on the lower left side of the display (if enabled); the timestamp (if used) is on the lower right side.
   - If you are recalling statistics, the information includes values for MEAN, STD DEV, SAMPLE SIZE, MINIMUM, MAXIMUM, and PK-PK.
   - The source display field identifies the buffer: SrcA1 (buffer 1) or SrcA2 (buffer 2).

Buffer location number

The buffer location number indicates the memory location of the source-measure reading. For example, location #000001 indicates that the displayed source-measure reading is stored at the first memory location.

Timestamp

If the timestamp is enabled, the first source-measure reading stored in the buffer (#0000001) is timestamped at 0.000 seconds. Subsequent readings are timestamped relative to when the first measurement was made. The interval between readings depends on the reading rate.

Displaying other buffer readings and statistics

To display other readings and statistics in the reading buffer:

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

Remote reading buffer programming

You can get readings by making overlapped or sequential measurements. Overlapped commands do not finish executing before the next command starts. Sequential commands complete execution before the next command starts executing.

The measured value is not the only component of a reading. The measurement status (for example, “In Compliance” or “Overranged”) is also an element of data associated with a particular reading.

All routines that return measurements can store the measurements in the reading buffers. Overlapped measurements always return readings in a reading buffer. Nonoverlapped measurement functions can return single-point measurement values or store multiple values in a reading buffer.

A reading buffer is based on a Lua table. The measurements are accessed by ordinary array accesses. If `rb` is a reading buffer, the first measurement is accessed as `rb[1]` and the ninth measurement as `rb[9]`. The additional information in the table is accessed as additional members of the table.

The load, save, and write operations for reading buffers function differently in the remote state. From a remote command interface, you can extract data from reading buffers as the instrument acquires the data.

Dedicated reading buffer designations

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

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

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

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

Reading buffer commands

The following tables summarize commands associated with the reading buffers. See TSP command reference (on page 7-1) for detailed reading buffer command information.

Reading buffer commands

Command	Description
Commands to save and clear readings	
`smuX.savebuffer(smuX.nvbufferY)`	Saves the reading buffer to the nonvolatile memory on the Model 2651A.
`smuX.nvbuffer1.clear()`	Clears buffer 1.
`smuX.nvbuffer2.clear()`	Clears buffer 2.
`mybuffer = smuX.makebuffer(n)`	Creates a dynamically allocated buffer for *n* readings.
`mybuffer = nil`	Deletes the dynamically allocated buffer.
`savebuffer(smuX.nvbuffer1,"csv", "/usb1/mybuffer.csv")`	Saves the reading buffer to a USB flash drive.

Command	Description
Commands to store readings	
`smuX.measure.count = count`	The number of measurements to acquire.
`smuX.measure.overlappedi(rbuffer)`	Makes current measurements; stores readings in *rbuffer*.
`smuX.measure.overlappediv(ibuffer, vbuffer)`	Makes both current and voltage measurements; stores current readings in *ibuffer* and stores voltage readings in *vbuffer*.
`smuX.measure.overlappedp(rbuffer)`	Makes power measurements; stores readings in *rbuffer*.
`smuX.measure.overlappedr(rbuffer)`	Makes resistance measurements; stores readings in *rbuffer*.
`smuX.measure.overlappedv(rbuffer)`	Makes overlapped voltage measurements; stores readings in *rbuffer*.
`smuX.measure.v(rbuffer)`	Makes voltage measurements; stores readings in *rbuffer*.
`smuX.measure.i(rbuffer)`	Makes current measurements; stores readings in *rbuffer*.
`smuX.measure.iv(ibuffer, vbuffer)`	Makes both current and voltage measurements; stores current readings in *ibuffer* and stores voltage readings in *vbuffer*.
`smuX.measure.r(rbuffer)`	Makes resistance measurements; stores readings in *rbuffer*.
`smuX.measure.p(rbuffer)`	Makes power measurements; stores readings in *rbuffer*.
`smuX.trigger.measure.v(rbuffer)`	Configures voltage measurements to be made during a sweep, including where readings are stored (*rbuffer*).
`smuX.trigger.measure.i(rbuffer)`	Configures current measurements to be made during a sweep, including where readings are stored (*rbuffer*).

Command	Description
Commands to store readings	
`smuX.trigger.measure.r(rbuffer)`	Configures resistance measurements to be made during a sweep, including where readings are stored (*rbuffer*).
`smuX.trigger.measure.p(rbuffer)`	Configures power measurements to be made during a sweep, including where readings are stored (*rbuffer*).
`smuX.trigger.measure.iv(ibuffer, vbuffer)`	Configures both current and voltage measurements to be made during a sweep, including where readings are stored; current readings are stored in *ibuffer* and voltage readings are stored in *vbuffer*.

Command	Description
Commands to access readings	
`printbuffer(start_index, end_index, st_1, st_2, ... st_n)`	Prints data from buffer subtables: ■ *start_index* (starting index of values to print) ■ *end_index* (ending index of values to print) ■ *st_1, st_2, ... st_n* (subtables from which to print, each separated by a comma)

Buffer storage control attributes

The following table contains buffer storage control attributes.

NOTE

Before changing the `collectsourcevalues`, `collecttimestamps`, or `timestampresolution` attributes, you must clear the buffer using the `smua.nvbuffer1.clear()` or `smua.nvbuffer2.clear()` command.

Buffer storage attribute	Description
appendmode	The append mode is either off or on. When the append mode is off, a new measurement to this buffer overwrites the previous contents. When the append mode is on, the first new measurement is stored at the end of the existing data. This attribute is off when the buffer is created.
cachemode	When this attribute is on, the reading buffer cache improves access speed to reading buffer data. When running successive operations that overwrite reading buffer data without running any commands that automatically invalidate the cache, the reading buffer may return stale cache data. This attribute is initialized to on when the buffer is created.
collectsourcevalues	When this attribute is on, source values are stored with readings in the buffer. This value, off or on, can be changed only when the buffer is empty. When the buffer is created, this attribute is initialized to off.
collecttimestamps	When this attribute is on, timestamps are stored with readings in the buffer. This value, off or on, can be changed only when the buffer is empty. When the buffer is created, this attribute is initialized to off.
fillcount	The reading buffer fill count sets the number of readings to store before restarting at index 1. If the value is 0, then the capacity of the buffer is used. This attribute is only used when the fillmode attribute is set to FILL_WINDOW.
fillmode	The reading buffer fill mode controls how new data is added to the reading buffer. When this attribute is set to FILL_ONCE, the reading buffer does not overwrite readings. If the buffer fills up, new readings are discarded. When this attribute is set to FILL_WINDOW, new readings are added after existing data until the buffer holds fillcount elements. Once there are fillcount elements, new data starts overwriting data starting at index 1.
timestampresolution	The timestamp resolution, in seconds. When the buffer is created, its initial resolution is 0.000001 seconds. At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. This value can be increased for long tests.

Buffer read-only attributes

The following table contains buffer read-only attributes.

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

Attribute	Description
basetimestamp	The timestamp of when the reading at rb[1] was stored, in seconds from midnight January 1, 1970 GMT. See Time and date values (on page 7-2) for additional details.
capacity	The total number of readings that can be stored in the reading buffer.
n	The number of readings in the reading buffer.
next	This attribute indicates where the next element that is added to the reading buffer is stored.

Buffer storage control programming examples

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

Buffer control programming examples

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

Buffer read-only attribute programming examples

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

Buffer read-only attribute programming examples

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

Statistic attributes

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

Attributes for accessing reading buffer data

Attribute	When returned	Description
`n`	Always	The number of data points on which the statistics are based
`mean`	When `n` > 0	The average of all readings added to the buffer
`stddev`	When `n` > 1	The standard deviation of all readings (samples) added to the buffer
`min`	When `n` > 0	A table containing data about the minimum reading value added to the buffer
`max`	When `n` > 0	A table containing data about the maximum reading value added to the buffer

If `n` equals zero (0), all other attributes are `nil` because there is no data to base any statistics on. If `n` equals 1, the `stddev` attribute is `nil` because the standard deviation of a sample size of 1 is undefined.

The `min` and `max` entries have the attributes described in the following table (*`bufferVar`* is the name of the buffer). See smuX.buffer.getstats() (on page 7-223) for additional information.

Min and max entry attributes

Attribute	Description
`measurefunction`	String indicating the function that was measured for the reading (current, voltage, ohms, or watts)
`measurerange`	The full-scale range value for the measurement range used when the measurement was made
`reading`	The reading value
`sourcefunction`	String indicating the source function at the time of the measurement (current or voltage)
`sourceoutputstate`	String indicating the state of the source (off or on)
`sourcerange`	Full-scale range value for the source range used when the measurement was made
`sourcevalue`	If *`bufferVar`*`.collectsourcevalues` is enabled, the sourced value in effect at the time of the reading
`status`	Status value for the reading; the status value is a floating-point number that encodes the status value into a floating-point value
`timestamp`	If *`bufferVar`*`.collecttimestamps` is enabled, the timestamp, in seconds, between when the reading was acquired and when the first reading in the buffer was acquired; adding this value to the base timestamp produces the actual time the measurement was acquired

Example:

The following programming example illustrates how to output mean and standard deviation statistics from buffer 1:

```
statistics = smua.buffer.getstats(smua.nvbuffer1)
print(statistics.mean, statistics.stddev)
```

Reading buffer attributes

Use the reading buffer attributes to access the reading buffer data. The table below displays the attributes that you can use to access the reading buffer data.

Recall attribute*	Description
`measurefunctions`	An array (a Lua table) of strings indicating the function measured for the reading (current, voltage, ohms, or watts).
`measureranges`	An array (a Lua table) of full-scale range values for the measure range used when the measurement was made.
`readings`	An array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly; that is, `rb[2]` and `rb.readings[2]` access the same value.
`sourcefunctions`	An array (a Lua table) of strings indicating the source function at the time of the measurement (current or voltage).
`sourceoutputstates`	An array (a Lua table) of strings indicating the state of the source (off or on).
`sourceranges`	An array (a Lua table) of full-scale range values for the source range used when the measurement was made.
`sourcevalues`	If enabled, an array (a Lua table) of the sourced values in effect at the time of the reading.
`statuses`	An array (a Lua table) of status values for all the readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value. See Buffer status (on page 3-18).
`timestamps`	If enabled, an array (a Lua table) of timestamps, in seconds, of when each reading occurred. These are relative to the `basetimestamp` for the buffer. See Reading buffer commands (on page 3-12).

* The default attribute is `readings`, which can be omitted.

Examples:

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

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

Similarly, the following outputs 100 corresponding source values from buffer 1:

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

The default reading attribute is `readings`, which can be omitted. If `readings` is omitted, the following also outputs 100 readings from buffer 1:

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

Buffer status

The buffer reading status attribute includes the status information as a numeric value; see the following table for values. For example, to access status information for the second element of SMU buffer 1, use the following command:

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

Buffer status bits

Bit	Name	Hex value	Description
B0	FastADC	0x01	Fast ADC was used to make the reading
B1	Overtemp	0x02	Overtemperature condition
B2	AutoRangeMeas	0x04	Measure range was autoranged
B3	AutoRangeSrc	0x08	Source range was autoranged
B4	4Wire	0x10	4-wire (remote) sense mode was enabled
B5	Rel	0x20	Relative offset was applied to a reading
B6	Compliance	0x40	Source function was in compliance
B7	Filtered	0x80	Reading was filtered

Dynamic reading buffers

Reading buffers can also be allocated dynamically. You create and allocate the dynamic reading buffers with the `smua.makebuffer(n)` command, where *n* is the number of readings the buffer can store. For example, the following command allocates a reading buffer named `mybuffer` that can store 100 readings:

```
mybuffer = smua.makebuffer(100)
```

You can delete allocated reading buffers by sending the following command:

```
mybuffer = nil
```

You can use dynamically allocated reading buffers interchangeably with the `smua.nvbufferY` buffers that are described in Dedicated reading buffer designations (on page 3-11).

Buffer examples

Dedicated reading buffer example

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

```
-- Restore Model 2651A defaults.
smua.reset()
-- Set display.
display.screen = display.SMUA
-- Display current.
display.smua.measure.func = display.MEASURE_DCAMPS
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Clear buffer 1.
smua.nvbuffer1.clear()
-- Enable append buffer mode.
smua.nvbuffer1.appendmode = 1
-- Enable source value storage.
smua.nvbuffer1.collectsourcevalues = 1
-- Set the count to 1.
smua.measure.count = 1
-- Select the source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set the bias voltage to 0 V.
smua.source.levelv = 0.0
-- Turn on the output.
smua.source.output = smua.OUTPUT_ON
-- Loop for voltages from 0.01 V to 1 V.
for v = 1, 100 do
   -- Set the source voltage.
   smua.source.levelv = v * 0.01
   -- Measure the current and store in nvbuffer1.
   smua.measure.i(smua.nvbuffer1)
end
-- Turn off the output.
smua.source.output = smua.OUTPUT_OFF
-- Output readings 1 to 100.
printbuffer(1, smua.nvbuffer1.n, smua.nvbuffer1.readings)
-- Output source values 1 to 100.
printbuffer(1, smua.nvbuffer1.n, smua.nvbuffer1.sourcevalues)
```

Dual buffer example

The programming example below shows a script that stores current and voltage readings using buffer 1 for current and buffer 2 for voltage readings. The Model 2651A stores 100 current and voltage readings and then recalls all 100 sets of readings.

```
-- Restore Model 2651A defaults.
smua.reset()
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select measure V autorange.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Clear buffer 1.
smua.nvbuffer1.clear()
-- Clear buffer 2.
smua.nvbuffer2.clear()
-- Set buffer count to 100.
smua.measure.count = 100
-- Set measure interval to 0.1 s.
smua.measure.interval = 0.1
-- Select source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Output 1 V.
smua.source.levelv = 1
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Store current readings in buffer 1, voltage readings in buffer 2.
smua.measure.overlappediv(smua.nvbuffer1, smua.nvbuffer2)
-- Wait for buffer to fill.
waitcomplete()
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
-- Output buffer 1 readings 1 to 100.
printbuffer(1, 100, smua.nvbuffer1)
-- Output buffer 2 readings 1 to 100.
printbuffer(1, 100, smua.nvbuffer2)
```

Dynamically allocated buffer example

The programming example below illustrates how to store data to an allocated buffer called `mybuffer`. The Model 2651A stores 100 current readings in `mybuffer` and then recalls all the readings.

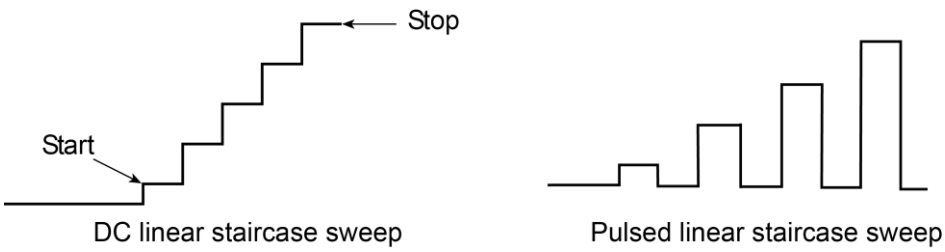
```
-- Restore Model 2651A defaults.
smua.reset()
-- Select measure I autorange.
smua.measure.autorangei = smua.AUTORANGE_ON
-- Select measure V autorange.
smua.measure.autorangev = smua.AUTORANGE_ON
-- Select ASCII data format.
format.data = format.ASCII
-- Set the buffer count to 100.
smua.measure.count = 100
-- Set the measure interval to 0.1 s.
smua.measure.interval = 0.1
-- Select the source voltage function.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Set the source voltage to output 1 V.
smua.source.levelv = 1
-- Turn on the output.
smua.source.output = smua.OUTPUT_ON
-- Create a temporary reading buffer.
mybuffer = smua.makebuffer(smua.measure.count)
-- Store current readings in mybuffer.
smua.measure.overlappedi(mybuffer)
-- Wait for the buffer to fill.
waitcomplete()
-- Turn off the output.
smua.source.output = smua.OUTPUT_OFF
-- Output readings 1 to 100 from mybuffer.
printbuffer(1, 100, mybuffer)
-- Delete mybuffer.
mybuffer = nil
```

Sweep operation

The Model 2651A can generate dc and pulsed sweeps to perform source-only sweeps, source-and-measure sweeps, or measure-only sweeps. The following information describes the sweep types of dc and pulsed linear staircase, dc and pulsed logarithmic staircase, and dc and pulsed list.

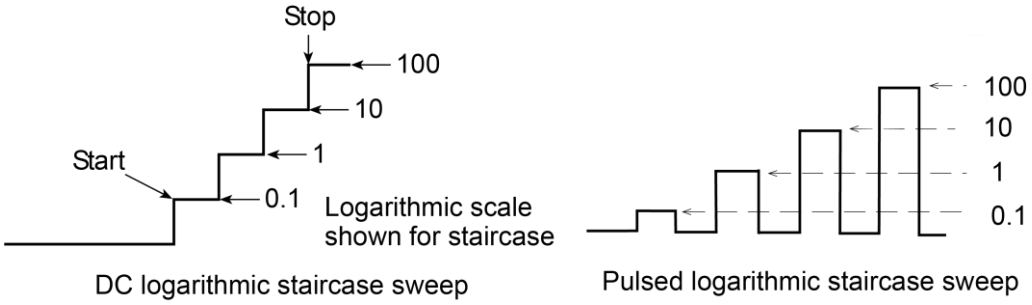
DC and pulsed linear staircase sweeps: With this type of sweep, the voltage or current increases or decreases in fixed steps, beginning with a start voltage or current and ending with a stop voltage or current. The figure below shows an increasing linear staircase sweep and a pulsed staircase sweep.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

Figure 35: DC and pulsed linear staircase sweeps



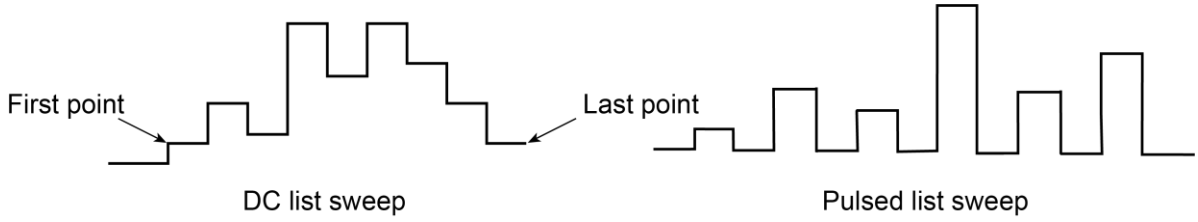
DC and pulsed logarithmic staircase sweeps: In this type of sweep, the current or voltage increases or decreases geometrically, beginning with a start voltage or current and ending with a stop voltage or current. The figure below shows an increasing logarithmic staircase sweep and a pulsed logarithmic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

Figure 36: DC and pulsed logarithmic staircase sweeps



DC and pulsed list sweeps: The list sweep allows you to program arbitrary sweep steps anywhere within the output voltage or current range of the Model 2651A. The following figure shows a list sweep with arbitrary steps and a pulsed list sweep. Pulsed list sweeps function the same way that dc list sweeps function, except that pulsed list sweeps return to the idle level between pulses.

Figure 37: DC and pulsed list sweeps



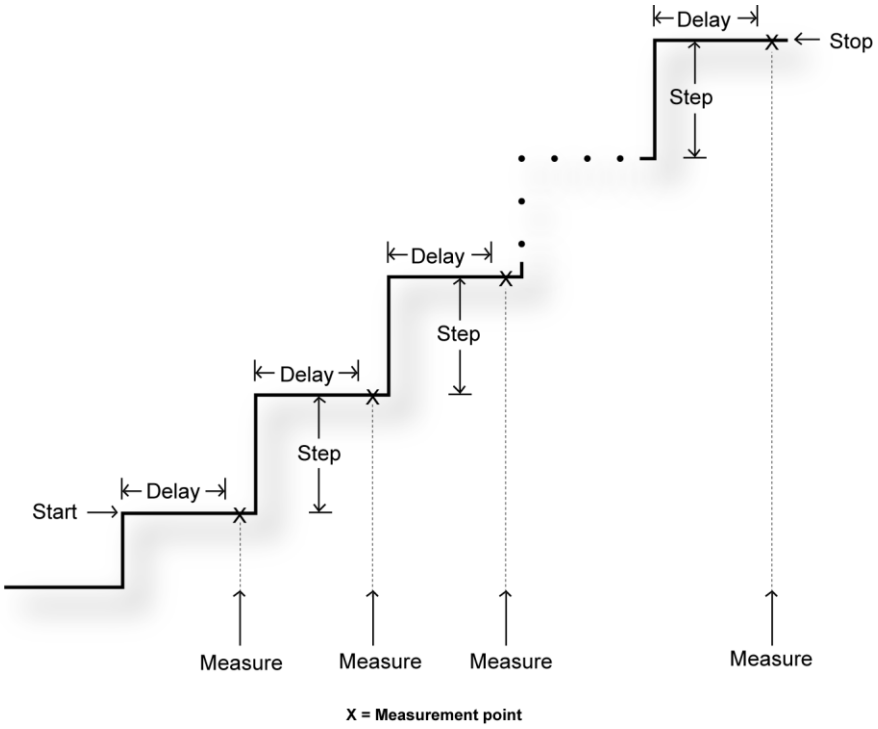
Sweep characteristics

For any of the sweep types, program a pulse sweep by configuring the end pulse action. Refer to Pulse mode sweeps (on page 3-29) for more information.

Linear staircase sweeps

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

Figure 38: Linear staircase sweep



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

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

NOTE

The number of sweep steps actually performed is determined by the trigger count. Refer to Triggering (on page 3-36) for more information.

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

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

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

Example:

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

For more information, see smuX.trigger.source.linearY() (on page 7-302).

Logarithmic staircase sweeps

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

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

NOTE

The number of sweep steps actually performed is determined by the trigger count. See Triggering (on page 3-36) for more information.

The formula for a logarithmic sweep is:

$v_i = A + kb^i$

Where:

v_i = The source value at source point i

i = The index of points in the sweep (ranges from 0 to $N - 1$), where N is the number of points in the sweep

k = The initial source value as an offset from the asymptote

b = The step size ratio

A = The asymptote value

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

Figure 39: Increasing logarithmic sweep

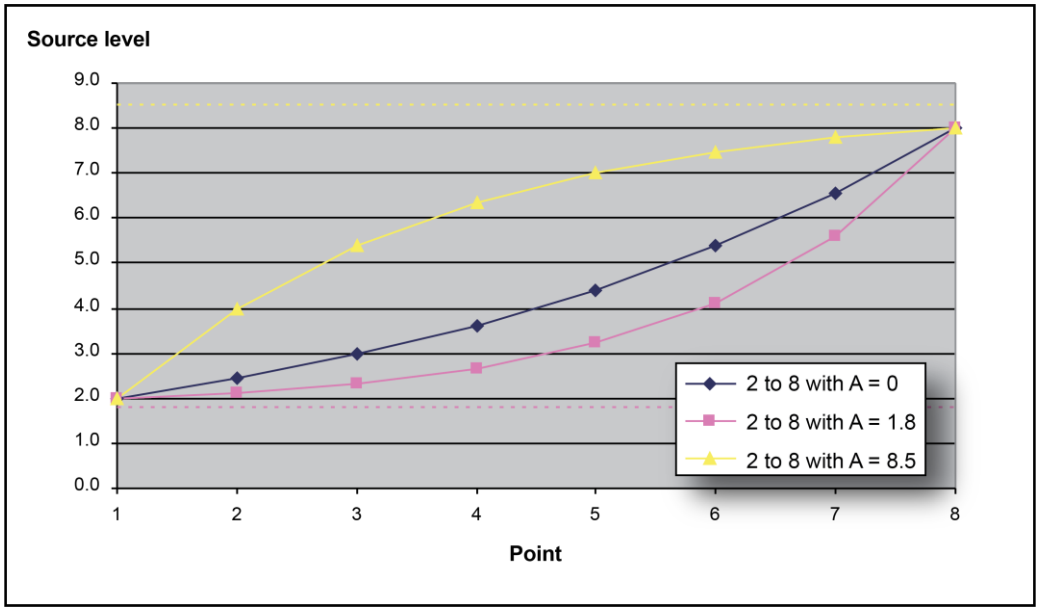
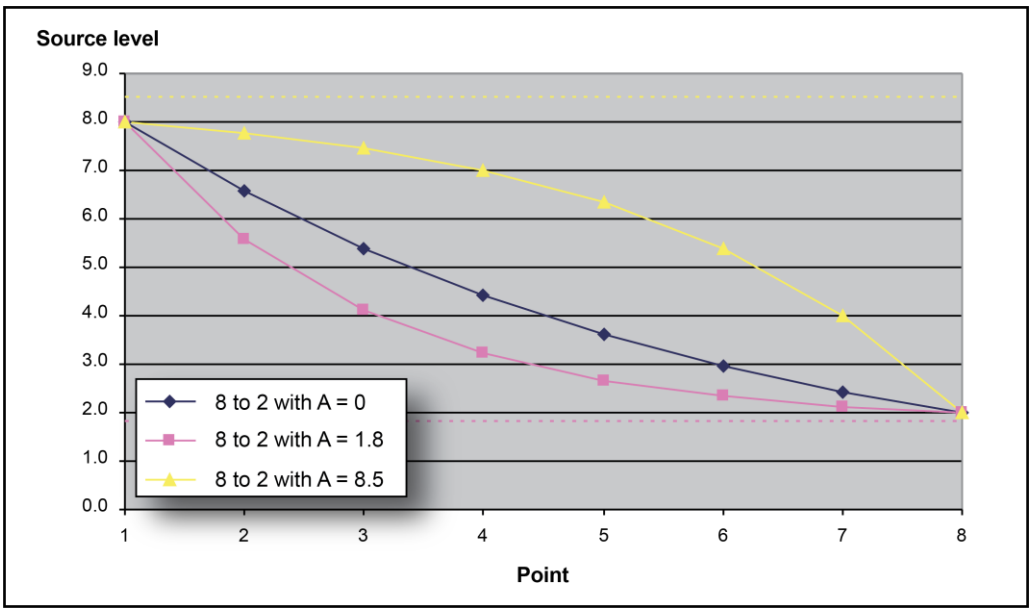


Figure 40: Decreasing logarithmic sweep



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

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

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

Where:

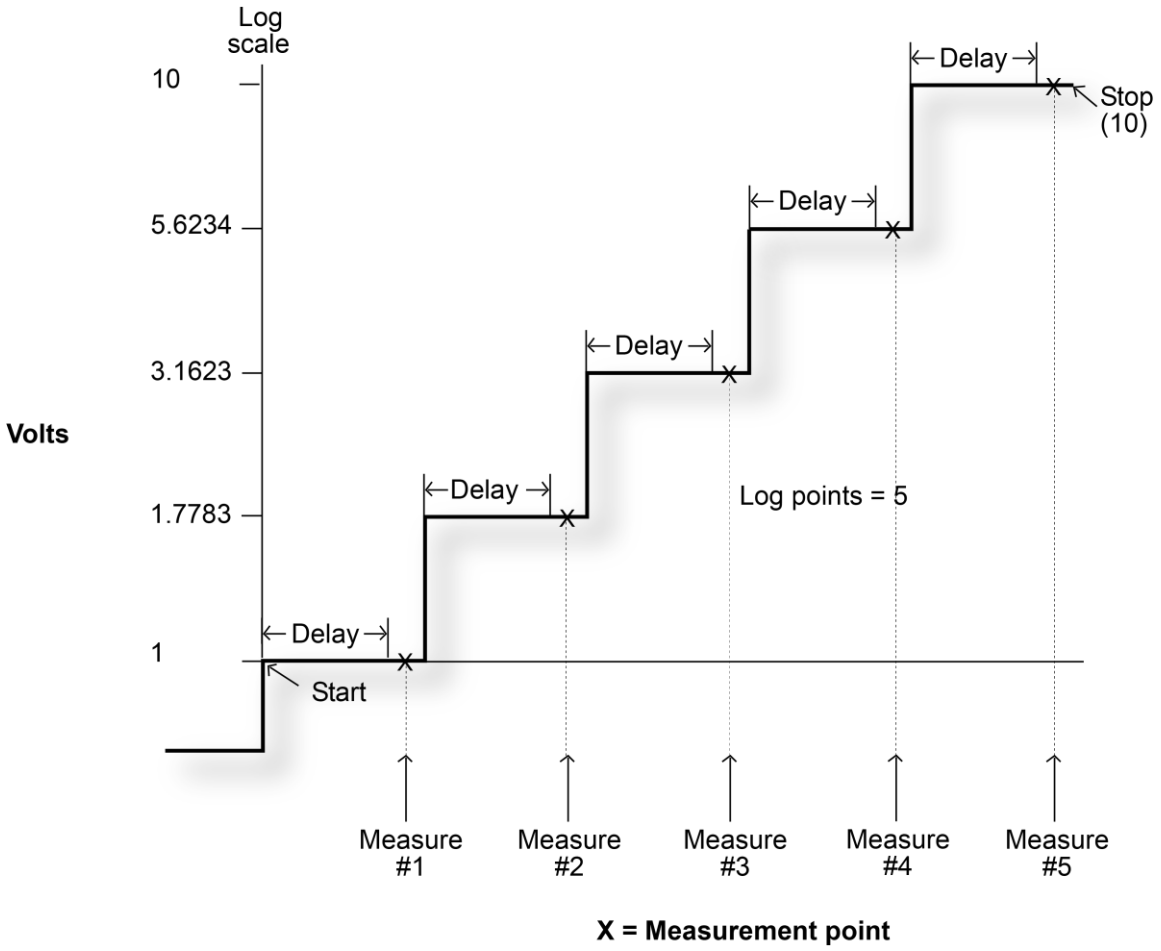
V_{end} = The source value at the end point
V_{start} = The source value at the start point
N = The number of points in the sweep
A = The asymptote value

NOTE

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

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

Figure 41: Logarithmic staircase sweep (1 V to 10 V, five steps)



In this example:

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

Using the formula above, $k = 1$

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

$$\text{Log step size} = \frac{\log10(\text{stop}) - \log10(\text{start})}{\text{Points} - 1}$$

$$= \frac{\log10(10) - \log10(1)}{5 - 1}$$

$$= \frac{1 - 0}{4}$$

$$= 0.25$$

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

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

Logarithmic sweep points

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

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

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

`smua.trigger.source.action`

Example:

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

For more information, see smuX.trigger.source.logY() (on page 7-304).

List sweeps

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

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

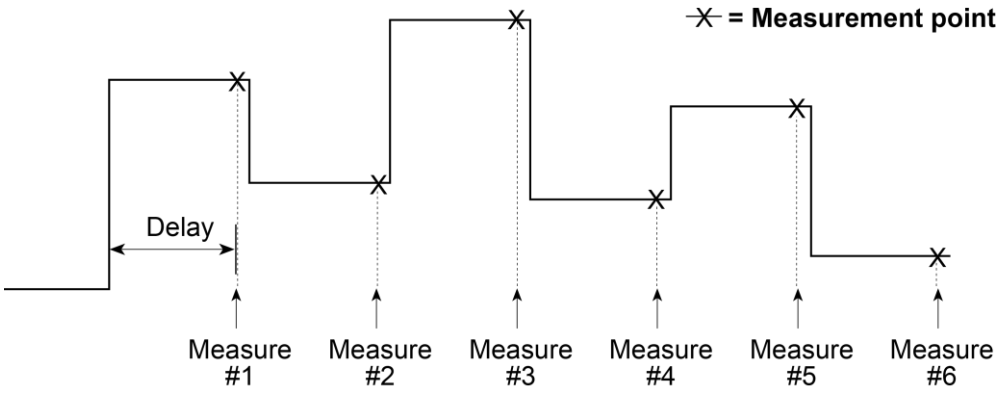
Example:

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

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

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

Figure 42: List sweep example



Pulse mode sweeps

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

To configure a pulse sweep, send:

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

To configure a dc sweep, send:

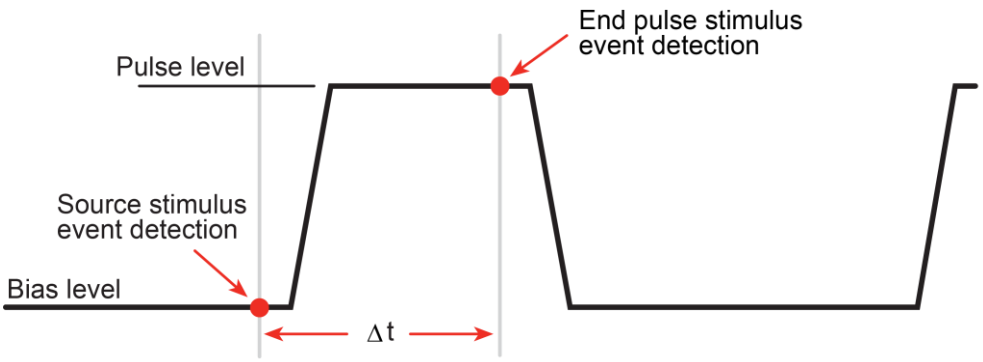
```
smua.trigger.endpulse.action = smua.SOURCE_HOLD
```

Timers must be used to configure the pulse width and period. Refer to Using timers to perform pulsed sweeps (on page 3-50) for details.

The pulse width is managed by controlling the duration between the source stimulus event and the end pulse stimulus event. A latency exists between these stimulus events and their resulting source level transitions. This trigger latency can vary based on factors such as the source range and the electrical characteristics of the device under test (DUT). The fast ADC mode can be used to characterize this latency, in order to better control the shape of the pulse under a particular set of test conditions.

The figure below shows the source and end pulse stimulus events in relationship to the pulse (see Triggering (on page 3-36) for information on stimulus events). Any change in Δt results in a corresponding change in the pulse width.

Figure 43: Pulse width control



Pulse duty cycle

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

Duty cycle = Pulse width / (Pulse width + Off time)

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

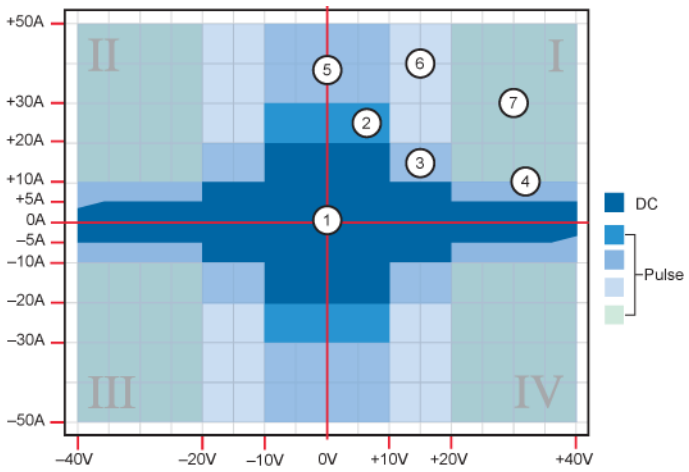
Duty cycle = 10 ms / (10 ms + 90 ms)
= 10 ms / 100 ms
= 0.10
= 10 percent

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

Pulsing in the extended operating area

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

Figure 44: Pulsing in the extended operating area



Pulse region specification

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

Configuring and running sweeps

Use the following topics to configure and run a sweep.

Configuring compliance limits remotely

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

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

Configuring end sweep actions remotely

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

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

Configuring measurements during a sweep

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

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

To recall sweep data using the front panel:

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

NOTE

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

To recall sweep data using remote commands:

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

See Reading buffers (on page 3-6) for details about recalling data from the buffer.

Source and measurement delays

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

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

Initiating and running sweeps

To run a sweep, you must configure the number of sweep points to output and the number of sweeps to perform. See Triggering (on page 3-36) for more information.

Examples:

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

To sweep 15 source points:

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

To perform eight sweeps:

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

Aborting a sweep

You can use the `smuX.abort()` function to terminate all overlapped operations on a source-measure unit (SMU), including sweeps. It returns the SMU to the idle state of the remote trigger model. See Triggering (on page 3-36) for more information.

Sweeping using factory scripts

Factory script functions that perform linear staircase, logarithmic staircase, and list sweeps are described in Factory scripts (on page 5-21). You can use the factory script functions to execute simple sweeps or use them as examples for programming your own custom sweeps.

To run a sweep from the front panel:

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

See Factory scripts (on page 5-21) for more information about using factory scripts.

Press the **RECALL** key to access sweep data stored in dedicated reading buffer 1. See Reading buffers (on page 3-6) for more details about the buffer.

Sweep programming examples

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

NOTE

You can retrieve the source code for the factory scripts by using the scriptVar.list() (on page 7-207) or scriptVar.source (on page 7-211) commands.

Sweep example parameters

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

Linear staircase sweep example

The programming example below illustrates a staircase sweep.

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

`-- Linear staircase sweep`
`-- 1 mA to 10 mA, 0.1 second delay,`
`-- 10 points.`
`SweepILinMeasureV(smua, 1e-3, 10e-3, 0.1, 10)`
	2. Configure and execute the sweep.
	Configures a linear staircase current sweep from 1 mA to 10 mA with 10 points and a 0.1 second settling time.

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

Pulse current sweep example

The programming example below illustrates a pulse sweep.

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

`-- Pulse current sweep, 1 mA bias,`
`-- 10 mA level, 10 ms pulse on,`
`-- 50 ms pulse off, 10 cycles.`
`PulseIMeasureV(smua, 1e-3, 10e-3, 20e-3, 50e-3, 10)`
	2. Configure and execute the sweep.
	Configures a 10 mA pulse current sweep with a 10 ms pulse-on time, a 50 ms pulse-off time, and 10 pulse-measure cycles.

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

List sweep example

The programming example below illustrates a list sweep.

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

`-- Define voltage list.`
`vlist = {3, 1, 4, 5, 2}`
`-- List sweep, channel A, 3 V, 1 V, 4 V,`
`-- 5 V, 2 V steps, 0.1 s delay, 5 points.`
`SweepVListMeasureI(smua, vlist, 0.1, 5)`
	2. Configure and execute the sweep.
	Configures a list sweep with 3 V, 1 V, 4 V, 5 V, and 2 V points using a 0.1 second settling time.

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

Triggering

Triggering allows you to source signals and capture measurements when an input signal or combination of input signals meets a set of conditions that you set. Triggering controls the timing of when source and measure operations happen during a sweep. See Sweep operation (on page 3-21) for details on sweeping.

Remote triggering overview

There are two programming methods for triggering:

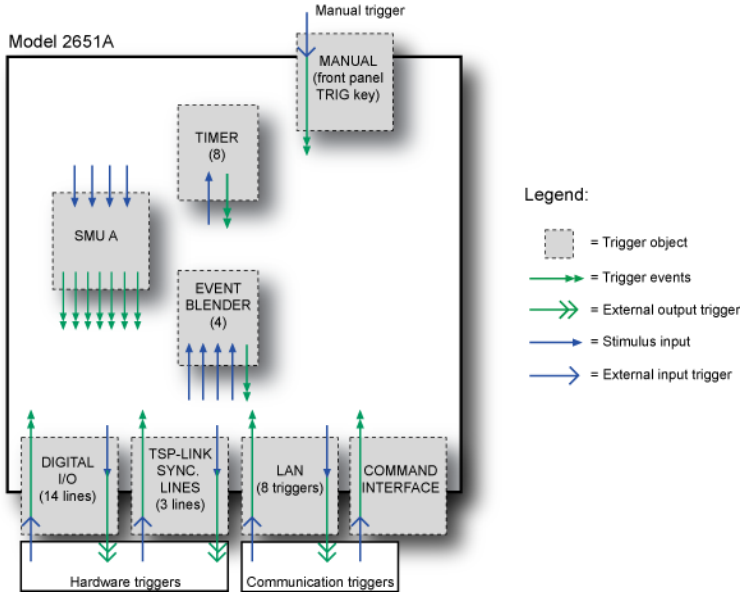
- Using the trigger model
- Interactive triggering

You can obtain very precise timing and synchronization between channels of multiple instruments using the trigger model to control the actions of the source-measure unit (SMU). To achieve such precise timing, use a static trigger configuration. When a static trigger configuration is not possible, you can use the interactive triggering method to control the timing and actions of the SMU.

Both programming methods use trigger objects. Trigger objects generate and monitor trigger events. External triggers are possible using digital I/O, TSP-Link® trigger lines, LAN, command interface, and the manual trigger (the TRIG key).

The following figure graphically represents all the trigger objects of the Model 2651A instrument.

Figure 45: Triggering overview



Trigger events are identified by means of an event ID. The following table describes the trigger event IDs.

Trigger event IDs*	
Event ID	**Event description**
`smua.trigger.SWEEPING_EVENT_ID`	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
`smua.trigger.ARMED_EVENT_ID`	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
`smua.trigger.SOURCE_COMPLETE_EVENT_ID`	Occurs when the SMU completes a source action
`smua.trigger.MEASURE_COMPLETE_EVENT_ID`	Occurs when the SMU completes a measure action
`smua.trigger.PULSE_COMPLETE_EVENT_ID`	Occurs when the SMU completes a pulse
`smua.trigger.SWEEP_COMPLETE_EVENT_ID`	Occurs when the SMU completes a sweep
`smua.trigger.IDLE_EVENT_ID`	Occurs when the SMU returns to the idle state
`digio.trigger[N].EVENT_ID`	Occurs when an edge is detected on a digital I/O line
`tsplink.trigger[N].EVENT_ID`	Occurs when an edge is detected on a TSP-Link line
`lan.trigger[N].EVENT_ID`	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed
`trigger.EVENT_ID`	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command `device_trigger`; reference the VXI-11 standard for additional details on the device trigger operation
`trigger.blender[N].EVENT_ID`	Occurs after a collection of events is detected
`trigger.timer[N].EVENT_ID`	Occurs when a delay expires

* Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).

Using the remote trigger model

The source-measure unit (SMU) in the Model 2651A has a remote trigger model that supports a wide range of triggering features for source sweeps, triggered measurements, and pulse actions.

Measurements using the trigger model can be made synchronously with sourcing actions or they can be made asynchronously. The following figures graphically illustrate both modes of the remote trigger model.

Figure 46: Remote trigger model: Normal (synchronous) mode

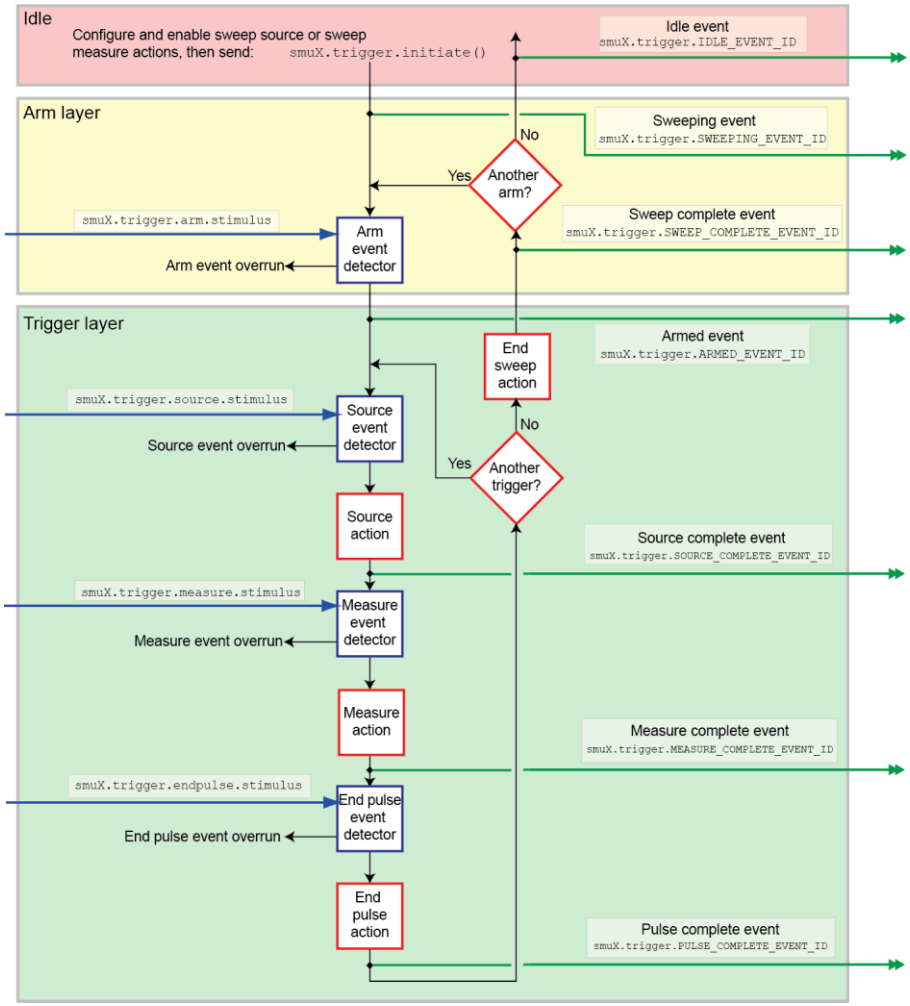
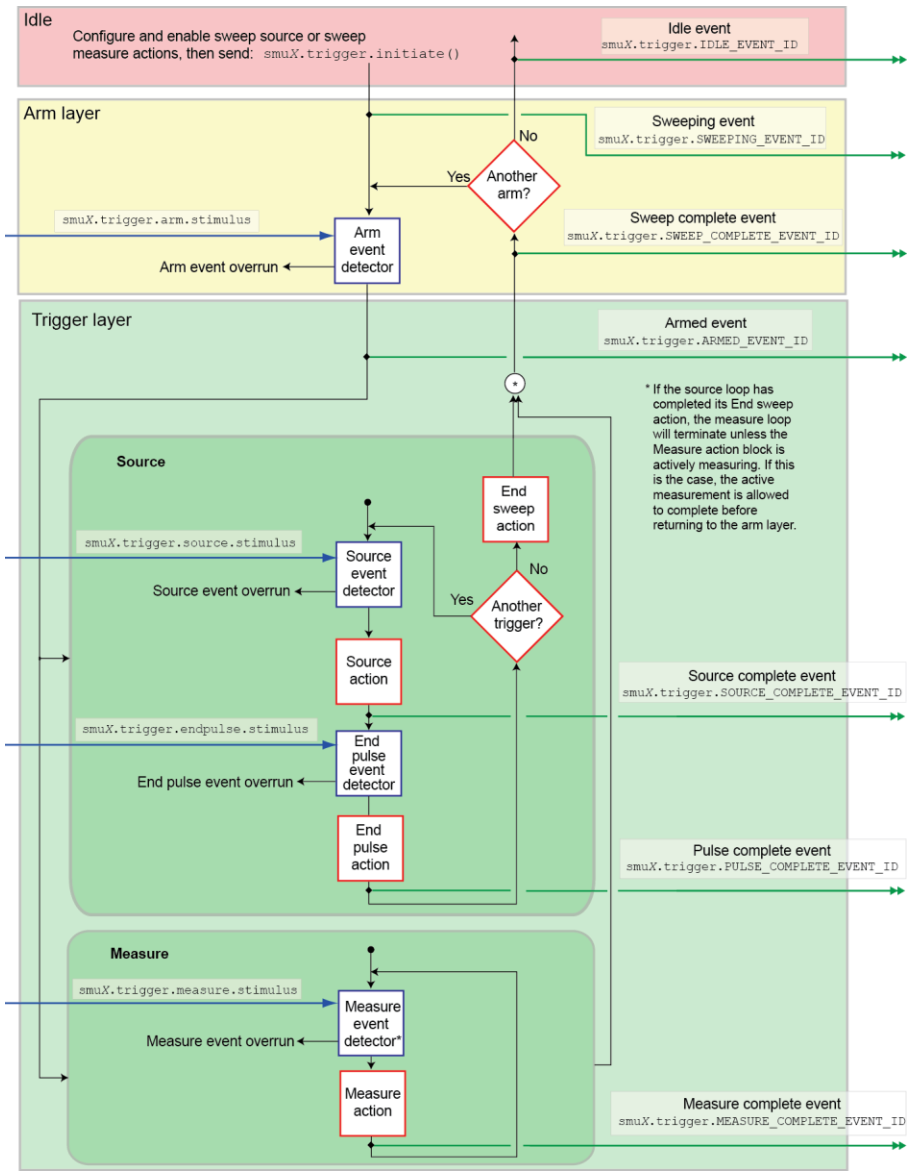


Figure 47: Remote trigger model: Asynchronous mode



When the `smuX.trigger.measure.action` attribute is set to `smuX.DISABLE` or `smuX.ENABLE`, the trigger model operates in synchronous measurement mode. When it is set to `smuX.ASYNC`, it operates in asynchronous mode.

Each section of the trigger model performs a function:

Idle state	If a sweep is not in process, the SMU is in the idle state. Use the `smuX.trigger.initiate()` function to move the SMU from the idle state to the arm layer.
Arm layer	Begins a sweep. Each sweep starts and ends in the arm layer.
Trigger layer	All source, measurement, and pulse actions occur in the trigger layer. ■ **Source**: Outputs the programmed voltage or current source value. ■ **Measurement**: Where the current, voltage, resistance, and power measurements occur. ■ **End pulse**: The end pulse action sources the idle (or bias) level if the pulse mode is enabled.

The remote trigger model dictates the sequence of operation for the SMU when it is configured to perform a sweep. When the SMU comes to an event detector, it suspends operation and waits for the event you have assigned to the stimulus input. If no event is assigned, the SMU continues uninterrupted past the event detector and through the trigger model. When the SMU comes to an action block, it performs the appropriate action, if enabled. The SMU loops through the arm and trigger layers until the programmed arm and trigger counts are satisfied.

Configuring source and measure actions

You can configure the source action using any of the following functions:

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

Where:

Y = Source function (`v` = voltage, `i` = current)

Source functions cannot be changed within a sweep. See Sweep operation (on page 3-21) for more details about the sweep functions.

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

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

NOTE

In asynchronous mode, trigger your measurements before the source completes the sweep (before the end sweep action occurs). If the source loop has completed its end sweep action, the measure loop terminates unless the measure action block is actively measuring. If this is the case, the active measurement is allowed to complete before returning to the arm layer.

Configured source and measure delays are imposed when the SMU executes the source and measure action blocks. Additionally, if the measure count setting is greater than one, then the measure count is satisfied each time the measure action is performed. Refer to Sweep operation (on page 3-21) for information about configuring source and measure sweeps.

The arm and trigger counts must be set to control how many times the SMU executes the source and measure actions. The arm count indicates the number of times to execute the complete sweep. The trigger count sets the number of loops in the trigger layer. Typically, you set the trigger count to be equal to the number of points in the configured sweep. If the trigger count is not equal to the number of points configured in the sweep, then one of the following occurs:

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

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

Enabling pulse mode sweeps using the end pulse action

Enable pulse mode sweeps using the end pulse action. The example command below illustrates how to configure pulse mode sweeps by setting the end pulse action:

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

Timers can be used to configure the pulse width and period (see Timers (on page 3-48) for more information). To disable pulse mode sweeps, set the `smua.trigger.endpulse.action` attribute to `smua.SOURCE_HOLD`.

SMU event detectors

As shown in Using the remote trigger model (on page 3-38), the source-measure unit (SMU) has multiple event detectors to control the timing of various actions, as shown in the table below. Each event detector monitors for the trigger event assigned to the associated stimulus input. Operation through the trigger model is delayed at the event detector until the programmed trigger event occurs.

If the stimulus input is set to zero (0), the SMU continues uninterrupted through the remote trigger model.

Event detectors

Event detector	Function
Arm	Controls entry into the trigger layer of the trigger model.
Source	Controls execution of the source action.
Measure	Controls execution of the measurement action.
End pulse	Controls execution of the end pulse action.

For the SMU, action overruns occur when a new trigger is detected before the previous trigger is acted upon. When the trigger model is configured for asynchronous measurements, a measurement trigger generates an overrun if the SMU is not ready to start a new measurement.

Clearing SMU event detectors

When an event detector is cleared, the event detector discards previously detected trigger events. This prevents the source-measure unit (SMU) from using trigger events that were detected during the last sweep or while it is in the arm layer, and allows it to start monitoring for new trigger events.

SMU event detectors are automatically cleared when:

- A sweep is initiated using the `smuX.trigger.initiate()` function.
- The SMU moves from the arm layer into the trigger layer and the `smuX.trigger.autoclear` attribute is enabled.

Using the TRIG key to trigger a sweep

The source-measure unit (SMU) can be configured to perform a sweep where each source step is triggered by the front-panel TRIG key. The source action is preceded by the source event detector. The SMU pauses operation at an event detector until a programmed event occurs. The SMU can be programmed to wait at the source event detector (that is, not start the source action) until the TRIG key is pressed.

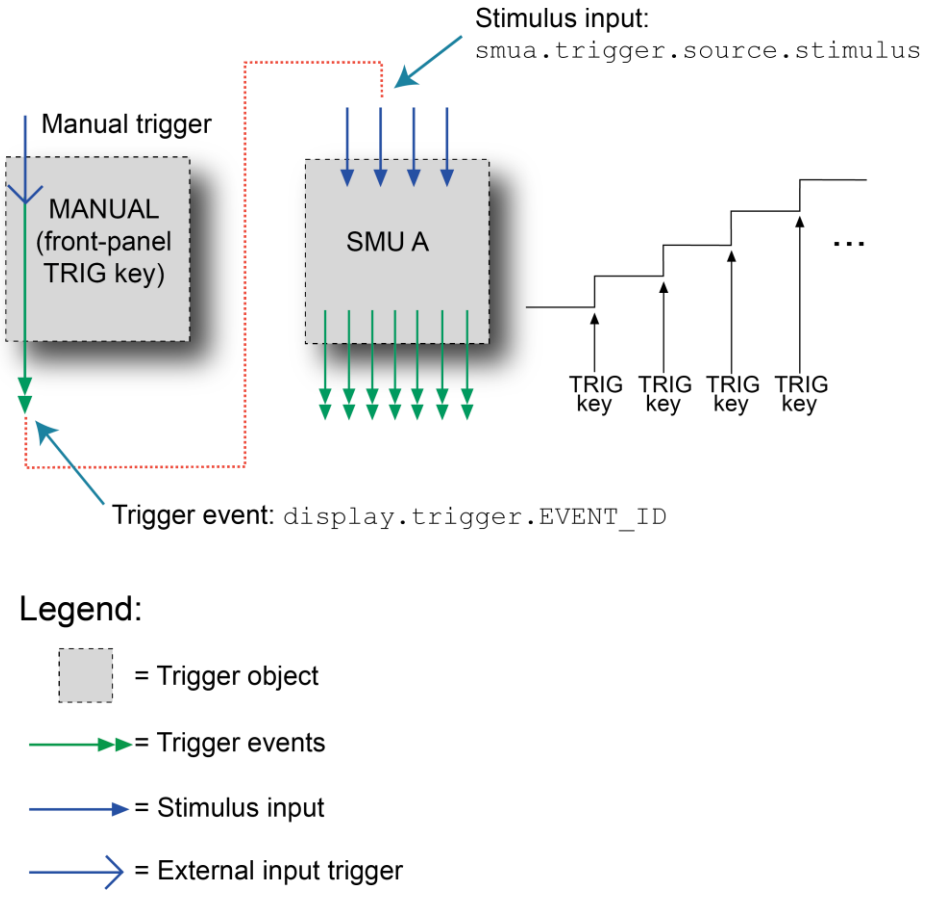
To configure the front panel TRIG key to trigger the source action, assign the trigger event created by the TRIG key (`display.trigger.EVENT_ID`) to the source stimulus input (`smua.trigger.source.stimulus`).

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

```
-- Configure a 10-point source voltage sweep.
smua.trigger.source.linearv(10, 100, 10)
smua.trigger.source.action = smua.ENABLE
-- Configure the TRIG key press as an input trigger for source action.
smua.trigger.source.stimulus = display.trigger.EVENT_ID
-- Configure the SMU to execute a single 10-point sweep.
smua.trigger.count = 10
smua.trigger.arm.count = 1
-- Turn on the output in preparation for the sweep.
smua.source.output = smua.OUTPUT_ON
-- Start the sweep and clear the event detectors.
smua.trigger.initiate()
-- The SMU waits for the  front-panel TRIG key press before executing
-- each source action.
-- Wait for the sweep to complete.
waitcomplete()
```

The following figure graphically illustrates this example. See Sweep operation (on page 3-21) for more information about sweep operation.

Figure 48: Front-panel TRIG key triggering



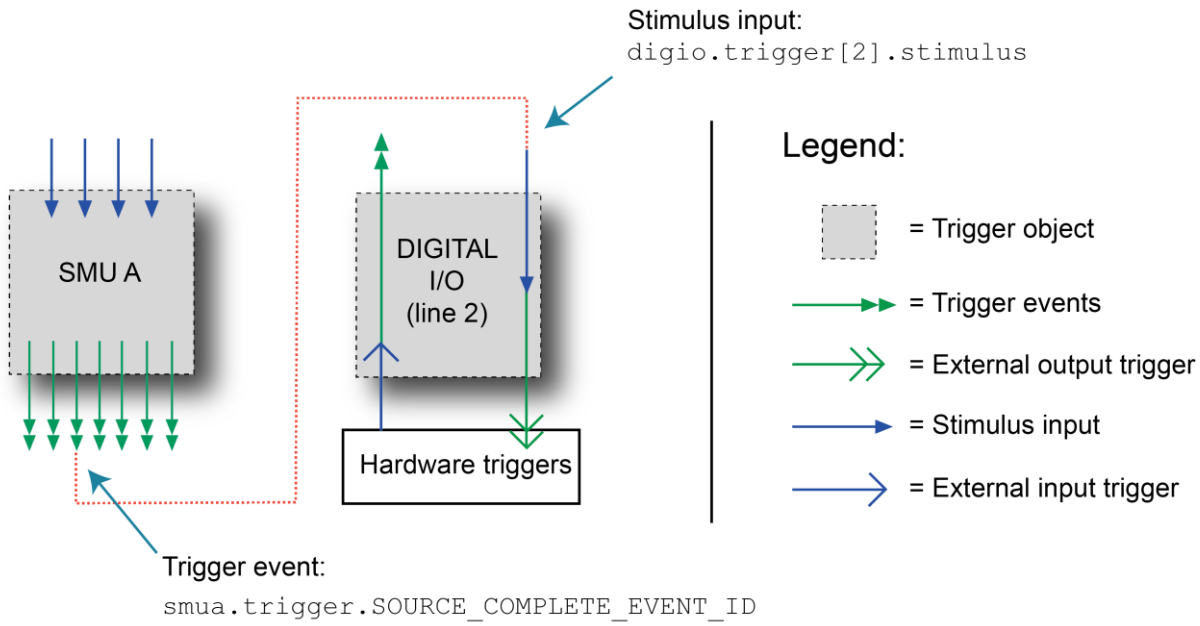
Using trigger events to start actions on trigger objects

You can configure trigger objects to respond to events generated by other trigger objects, such as using a digital I/O trigger to initiate a sweep. To configure a trigger object to monitor for an event, assign the event ID of the trigger event to the stimulus input. When the specified trigger event occurs, the trigger object performs an action. The programming example below illustrates how to generate a digital I/O line 2 output trigger pulse for each SMU A source complete event:

```
-- Configure digio line 2 to generate an output trigger pulse each
-- time SMU A generates a source complete event.
digio.trigger[2].stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID
```

The following figure illustrates this example.

Figure 49: Using trigger events to start actions



A stimulus input can be configured to monitor for only one trigger event ID at a time. To monitor more than one event, use an event blender. See Event blenders (on page 3-57) for more information.

Action overruns

An action overrun occurs when a trigger object receives a trigger event and is not ready to act on it. The action overruns of all trigger objects are reported in the operation event registers of the status model. Refer to Status model (on page 15-1) and the sections on each trigger object for details on the conditions that generate an action overrun.

Digital I/O port and TSP-Link synchronization lines

The Model 2651A has two sets of hardware lines that can be used for triggering: 14 digital I/O lines and three TSP-Link® synchronization lines. These trigger objects are configured and controlled in the same way.

See Digital I/O (on page 3-92) for more information about connections and direct control of the digital I/O and TSP-Link synchronization lines.

Mode

The mode indicates the type of edge the hardware lines detect as an external input trigger. Mode also indicates the type of signal generated as an external output trigger. The following table describes the hardware trigger modes for the hardware trigger lines. For additional detail, refer to Hardware trigger modes (on page 3-65).

NOTE

To disable triggering on the hardware trigger lines, set the mode to bypass. This allows direct control of the line.

Hardware trigger mode summary

Trigger mode	Output		Input
	Unasserted	Asserted	Detects
Bypass	N/A	N/A	N/A
Either Edge	High	Low	Either
Falling Edge	High	Low	Falling
Rising Edge	The programmed state of the line determines if the behavior is similar to RisingA or RisingM: ■ High similar to RisingA ■ Low similar to RisingM		
RisingA	High	Low	Rising
RisingM	Low	High	Not available
Synchronous	High latching	Low	Falling
SynchronousA	High latching	High	Falling
SynchronousM	High	Low	Rising

Pulse width

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

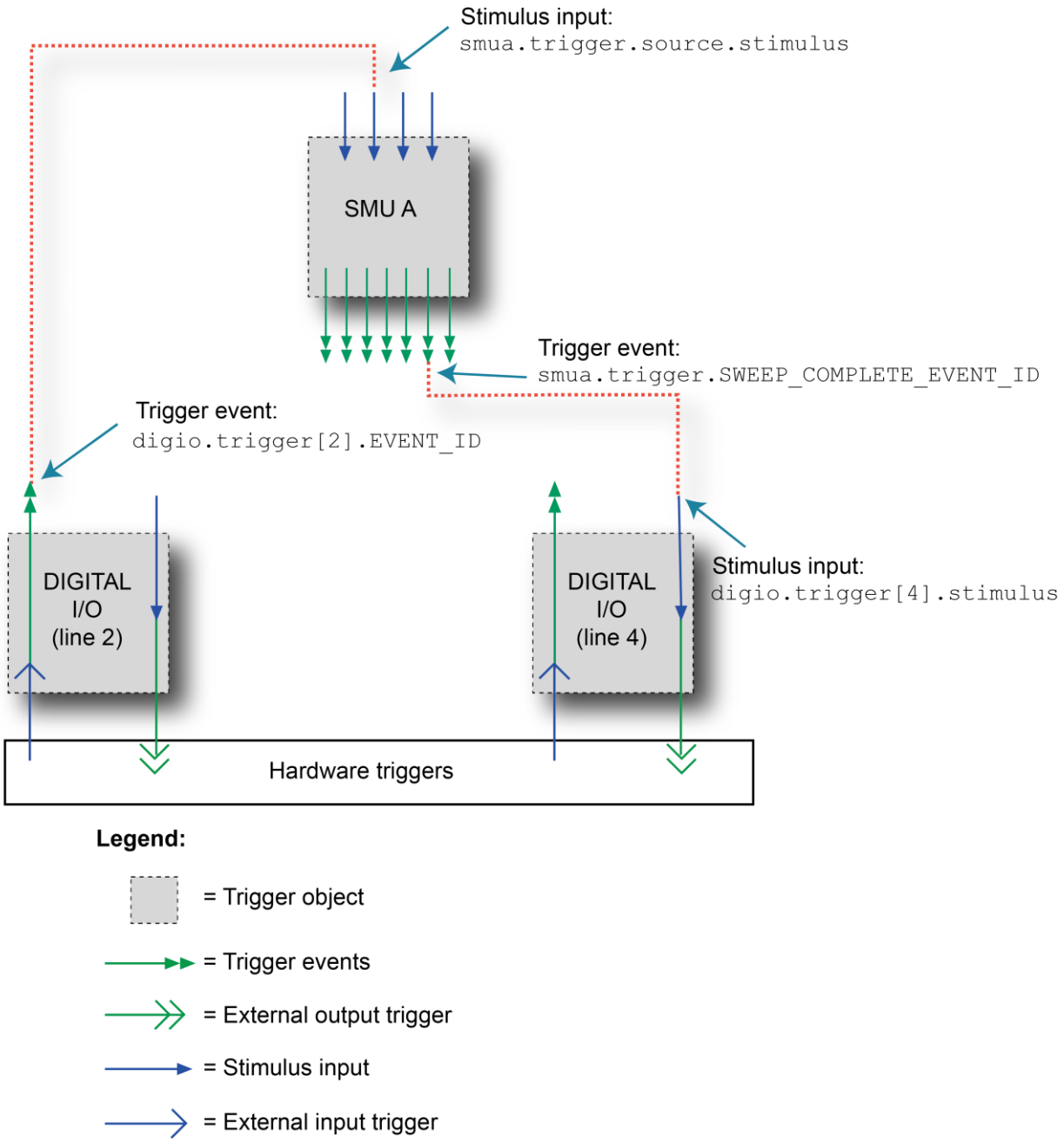
Trigger configuration on hardware lines

You can configure the Model 2651A to send digital signals to trigger external instruments. You can link the output triggers to the completion of certain source-measure actions to enable hardware handshaking. The programming example below illustrates this.

```
-- Configure the Model 2651A to detect a rising
-- edge on digital I/O line 2.
digio.trigger[2].mode = digio.TRIG_RISINGA
digio.trigger[2].clear()
-- Configure SMU A to start its source action when a
-- trigger event occurs on digital I/O line 2.
smua.trigger.source.stimulus = digio.trigger[2].EVENT_ID
-- Configure digital I/O line 4 to output a 1 ms
-- rising-edge trigger pulse at the completion of
-- the SMU sweep.
digio.trigger[4].mode = digio.TRIG_RISINGM
digio.trigger[4].pulsewidth = 0.001
digio.trigger[4].stimulus = smua.trigger.SWEEP_COMPLETE_EVENT_ID
```

The triggering setup for this example is shown in the following figure.

Figure 50: External instrument triggering



Action overruns on hardware lines

An action overrun occurs when a trigger event is received before the digital I/O or TSP-Link® line is ready to process it. The generation of an action overrun is dependent upon the trigger mode selected for that line. For more details on the causes of action overruns, see Hardware trigger modes (on page 3-65). Use the status model to monitor for the occurrence of action overruns. For details, see Status model (on page 15-1).

Timers

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

Timer attributes

Each timer has attributes that you can configure. These attributes are described in the following sections.

Count

The count sets the number of events to generate each time the timer generates a trigger event. Each event is separated by the delay set by the `trigger.timer[N].delay` command.

To configure the count, use the `trigger.timer[N].count` command.

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

Timer delays

You can configure timers to perform the same delay each time or set up a delay list that allows the timer to sequence through an array of delay values. All delay values are specified in seconds.

A delay is the period after the timer is triggered and before the timer generates a trigger event. The programming example below illustrates how to configure timer 3 for a 10 s delay:

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

You can configure a custom delay list to allow the timer to use a different interval each time it performs a delay. Each time the timer generates a trigger event, it uses the next delay value in the list. The timer repeats the delay list after all the elements in the delay list have been used. The programming example below illustrates how to configure timer 3 for delays of 2, 10, 15, and 7 seconds:

```
-- Configure timer 3 to complete delays of 2 s, 10 s,
-- 15 s, and 7 s.
trigger.timer[3].delaylist = {2, 10, 15, 7}
```

NOTE

Assigning a value to the delay attribute is the same as configuring it with a one-element delay list.

Pass-through mode

When enabled, the timer generates a trigger event immediately when it is triggered. The timer generates additional trigger events each time a delay expires. If the pass-through attribute is disabled, the timer does not generate a trigger event until after the first delay elapses. The programming example below illustrates how to configure timer 3 by enabling pass-through mode:

```
trigger.timer[3].passthrough = true
```

Triggering a timer

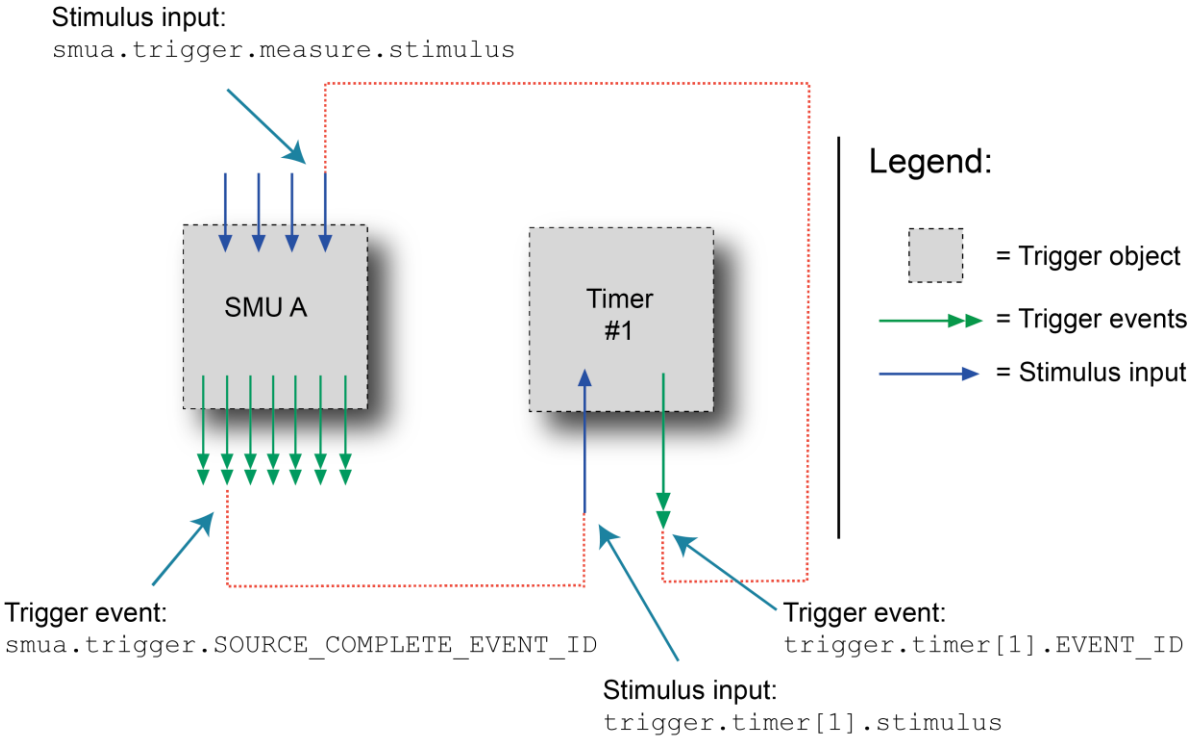
You can configure a timer to start a delay when a trigger object generates a trigger event. Timers cannot be started with a command. A trigger event from a trigger object must be used to initiate a delay.

Assigning the stimulus attribute

Assign an event ID to the `trigger.timer[N].stimulus` attribute to configure the timer to start a delay when a specific trigger event occurs. The programming example below illustrates how to configure a source-delay-measure (SDM) cycle.

```
-- Configure the timer to begin when source action completes.
trigger.timer[1].stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID
-- SMUA delay before a measurement begins.
smua.trigger.measure.stimulus = trigger.timer[1].EVENT_ID
```

Figure 51: Using a timer for an SDM cycle



Timer action overruns

The timer receives an action overrun when it generates a trigger event while a timer delay is still in progress. Use the status model to monitor for the occurrence of action overruns. For details, see Status model (on page 15-1).

Using timers to perform pulse mode sweeps

You can use timers to control the pulse width during a pulsed sweep. To create a pulse train, a second timer must be used to configure the pulse period. The following topics provide examples that show a single pulse output and a pulse train output.

NOTE

To create a pulse, the SMU end pulse action `smuX.trigger.endpulse.action` must be set to `smuX.SOURCE_IDLE`.

Pulsing from a positive to a negative pulse level

The following single pulse and pulse train examples pulse from a zero bias level to a positive pulse level (+5 V). If you change the pulse level to a negative value, such as −5 V, the pulse width is nominally 100 µs shorter than expected. The pulse width is shortened because the SMU source must change polarity when pulsing from zero to a negative level, and there is an internal 100 µs delay associated with the change. This polarity change is required because the number zero (0) is treated as a positive value. Therefore, pulsing from zero to +5 V does not require a polarity change, but pulsing from zero to −5 V does. There are multiple ways to obtain the correct pulse width, but the simplest is to define a negative zero and set the bias level equal to that value.

A negative zero is a value that is mathematically negative and functionally equivalent to zero. For a SMU, a source level setting that is functionally equivalent to zero is one that is significantly less than the programming resolution of the source range being used. Programming resolution values are listed by range in the SMU instrument specifications. A suitable value that works for all voltage and current source ranges in the Model 2651A instruments is $-1e^{-18}$. Therefore, to pulse from a zero bias level to a negative pulse level, make the following change to the examples:

```
-- Set the voltage source range and the bias source level and limit.
smua.source.rangev = 5
neg_zero = -1e-18
smua.source.levelv = neg_zero
smua.source.limiti = 0.1
```

Single pulse example

The SMU programming example below illustrates how to use a single timer to control the pulse width of a single-shot pulse measurement. The programming example configures the timer and SMU as follows:

Timer 1: Pulse-width timer

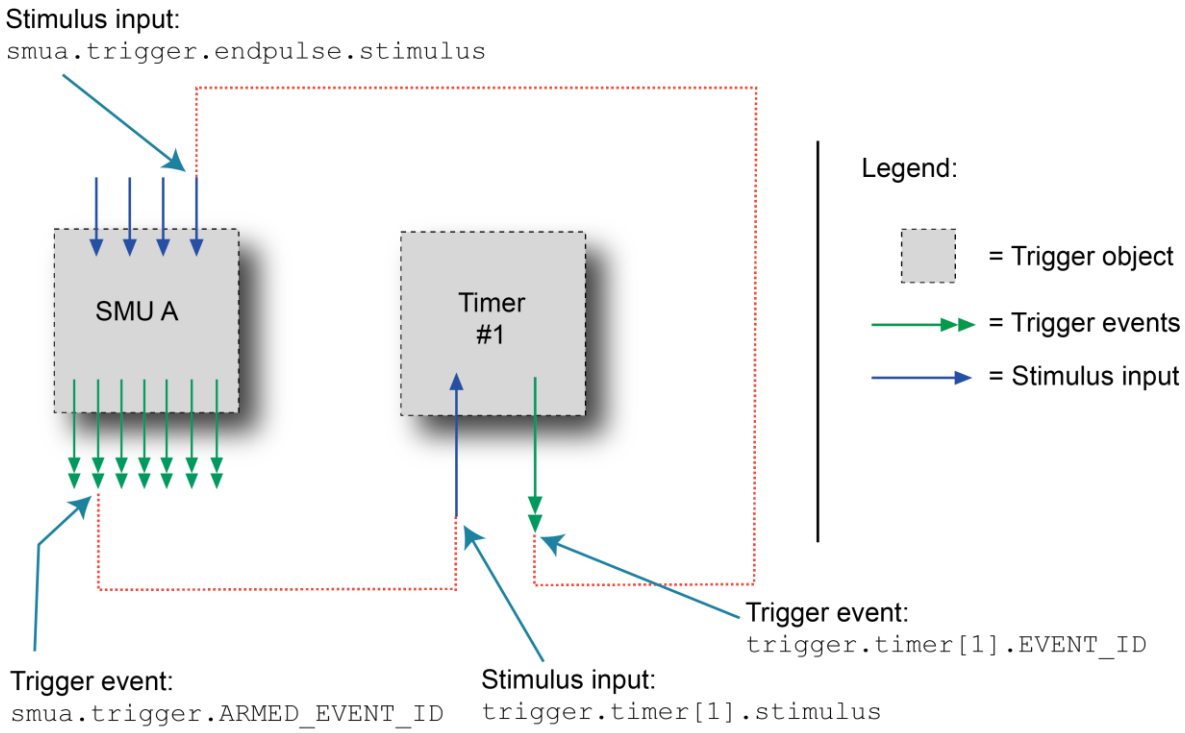
- Set the delay attribute of a timer equal to the appropriate pulse width.
- Configure the timer to trigger when the SMU moves out of the arm layer of the trigger model.
- Assign the trigger event generated by the timer to the stimulus input of the SMU end pulse event detector.

SMU

- Configure the source action to start immediately by setting the stimulus input of the source event detector to 0.
- Set the end pulse action to `SOURCE_IDLE`.

The following figure shows the trigger setup for this example.

Figure 52: Single-pulse triggering



Single pulse example code

NOTE

Even though no measurements are made in this example, a measure range is set. When sourcing voltage, it is good practice to set the current measure range equal to the triggered source limit range. This is especially important when the triggered limit is greater than 100 mA. If the measure range is not set, it may affect the shape of the pulse. This step is not necessary when sourcing current.

```
-- Reset SourceMeter instrument to default conditions.
reset()

-- Generate a single pulse with the following characteristics:
--      *  Bias (idle) level = 0 V
--      *        Pulse level = 5 V
--      *        Pulse width = 500 us

-- Configure the source function.
smua.source.func = smua.OUTPUT_DCVOLTS

-- Set the voltage source range and the idle or bias source level and limit.
smua.source.rangev = 5
smua.source.levelv = 0
smua.source.limiti = 0.1

-- Configure the trigger-timer parameters to output a single 500 us pulse.
trigger.timer[1].delay = 0.0005
trigger.timer[1].count = 1
trigger.timer[1].passthrough = false
-- Start the timer when the SMU moves from the ARM layer to the TRIGGER layer.
trigger.timer[1].stimulus = smua.trigger.ARMED_EVENT_ID

-- Configure the trigger model to execute a single-point voltage pulse list sweep.
-- No measurements are made.
smua.trigger.source.listv({5})
smua.trigger.source.action = smua.ENABLE
smua.trigger.measure.action = smua.DISABLE
-- Set the trigger source limit to the same value as the bias limit.
smua.trigger.source.limiti = smua.LIMIT_AUTO
smua.measure.rangei = 0.1
-- Configure the source action to start immediately.
smua.trigger.source.stimulus = 0
-- Configure the endpulse action to achieve a pulse.
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
-- Set the appropriate counts for the trigger model.
smua.trigger.arm.count = 1
smua.trigger.count = 1

-- Turn on the SMU output and initiate the trigger model to output a single pulse.
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
-- Wait for the sweep to complete.
waitcomplete()

-- Turn off SMU output.
smua.source.output = smua.OUTPUT_OFF
```

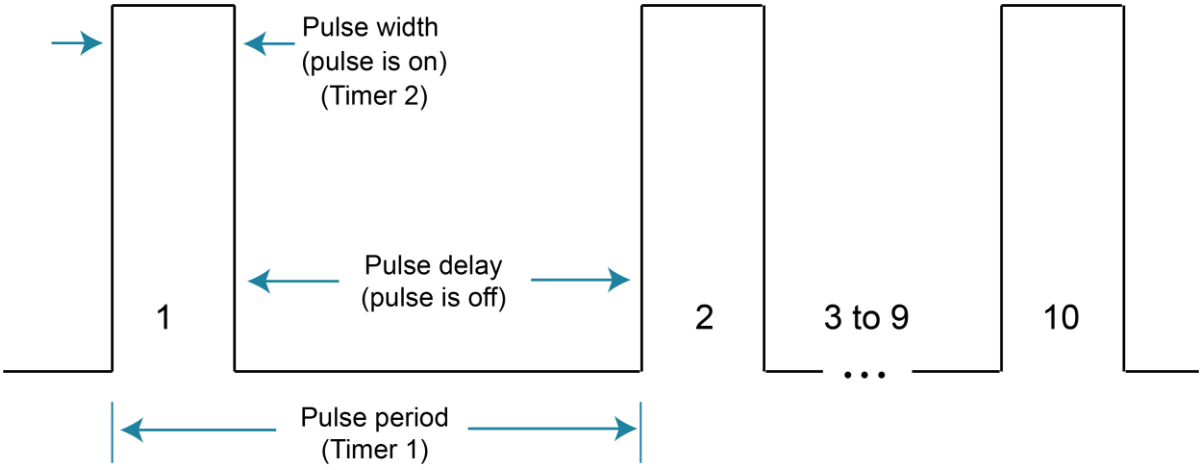
Pulse train example

The SMU programming example below illustrates how to use two timers: One to control the pulse period, a second to control the pulse width. The example configures the timers and SMU as follows:

Timer 1: Pulse period timer

- Set the delay attribute to the appropriate pulse period (see the following figure).
- Configure the timer to start when the sweep is initiated.
- Enable the pass-through attribute so that the timer generates a trigger event at the start of the first delay.
- Set the count equal to one less than the total number of pulses to output.

Figure 53: Pulse train



Timer 2: Pulse width timer

- Set the delay attribute to an appropriate pulse width (see the following figure).
- Set the stimulus input to the event ID of timer 1 (the start of each pulse is the start of the pulse period).
- Set the count equal to 1 so that only one pulse is issued per period.

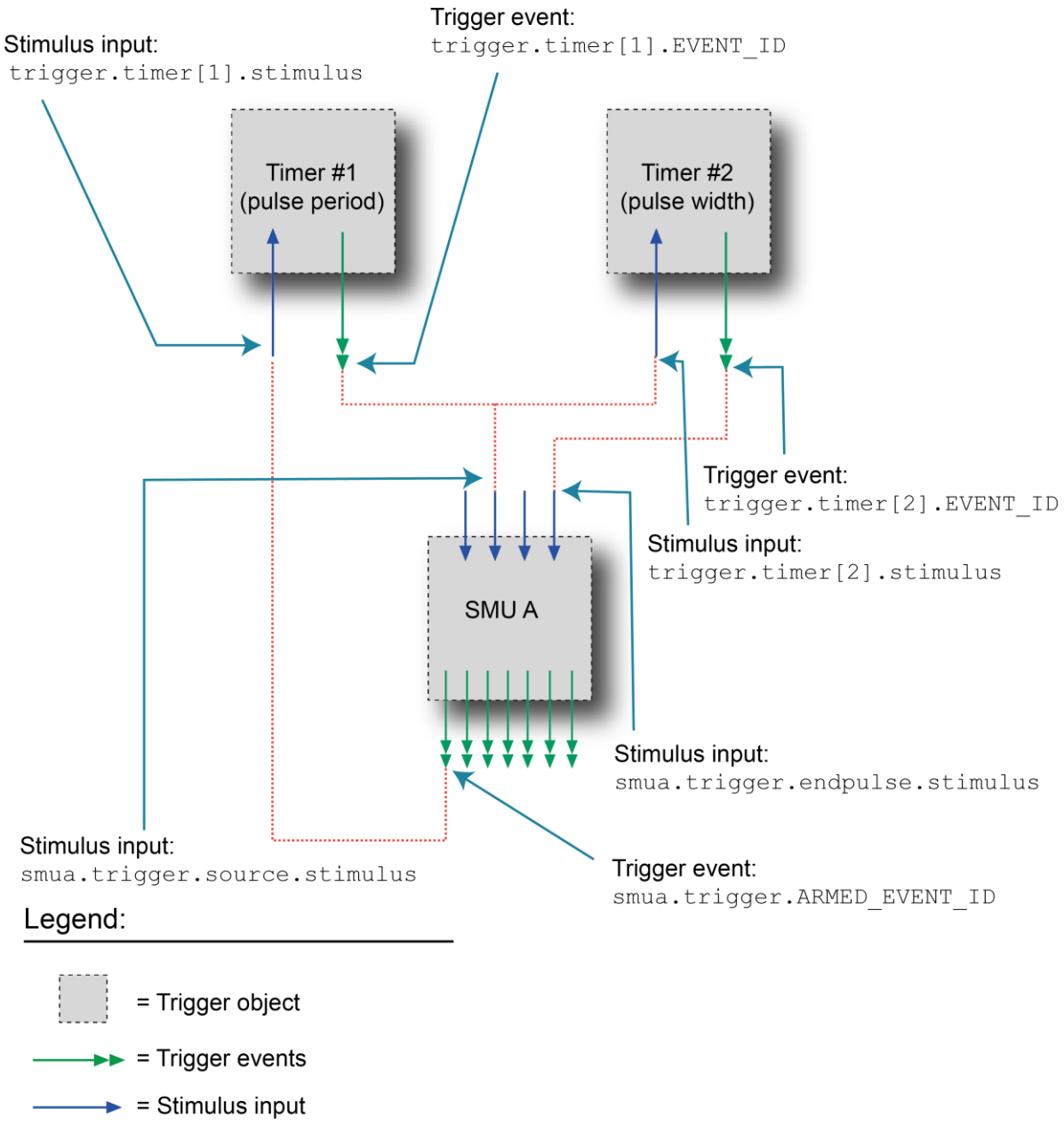
SMU A

- Set the source stimulus input to the event ID of timer 1 so that the source action starts when the period starts.
- Set the end pulse action to `smua.SOURCE_IDLE` so that the output is returned to the idle level after the pulse completes.
- Set the end pulse stimulus input to the event ID of timer 2 so that the end pulse action executes when the pulse width timer expires.

- Set the trigger count equal to the total number of pulses to output.
- Set the arm count to 1.

The following figure shows the trigger setup for this example.

Figure 54: Pulse train triggering



Pulse train example code

NOTE

Even though no measurements are made in this example, a measure range is set. When sourcing voltage, it is good practice to set the current measure range equal to the triggered source limit range. This is especially important when the triggered limit is greater than 100 mA. If the measure range is not set, it may impact the shape of the first pulse in the train. This step is not necessary when sourcing current.

```
-- Reset the SourceMeter instrument to default conditions.
reset()

-- Generate a 10-point pulse train with the following characteristics:
--      *  Bias (Idle) Level = 0 V
--      *        Pulse Level = 5 V
--      *        Pulse Width = 600 us
--      *       Pulse Period = 5 ms

-- Configure the source function.
smua.source.func = smua.OUTPUT_DCVOLTS

-- Set the voltage source range and the bias source level and limit.
smua.source.rangev = 5
smua.source.levelv = 0
smua.source.limiti = 0.1

-- Use trigger timer 1 to control the period and trigger timer 2 to control the
-- pulse width. Alias the timers for convenience and clarity.
period_timer = trigger.timer[1]
pulsewidth_timer = trigger.timer[2]

-- Configure the period timer to output 10 total trigger events.
period_timer.delay = 0.005
-- The effective count is 10 because the passthrough setting is true.
period_timer.count = 9
-- Configure the timer to immediately output a trigger event when it is started.
period_timer.passthrough = true
-- Start the timer when the SMU moves from the ARM layer to the TRIGGER layer.
period_timer.stimulus = smua.trigger.ARMED_EVENT_ID

-- Configure the pulse width timer to output one trigger event for each period.
pulsewidth_timer.delay = 0.0006
pulsewidth_timer.count = 1
-- Do not immediately output a trigger event when pulse width timer is started.
pulsewidth_timer.passthrough = false
-- Start the pulse width timer with the period timer output trigger event.
pulsewidth_timer.stimulus = period_timer.EVENT_ID

-- Configure the trigger model to execute a 10-point fixed-level voltage pulse
-- train. No measurements are made.
smua.trigger.source.listv({5})
smua.trigger.source.action = smua.ENABLE
smua.trigger.measure.action = smua.DISABLE
```

```
-- Set the trigger source limit, which can be different than the bias limit.
-- This is an important setting for pulsing in the extended operating area.
smua.trigger.source.limiti = 1
smua.measure.rangei = 1
-- Trigger SMU source action with the period timer event.
smua.trigger.source.stimulus = period_timer.EVENT_ID
-- Configure the endpulse action to achieve a pulse.
smua.trigger.endpulse.action = smua.SOURCE_IDLE
-- Trigger the SMU end pulse action with a pulse width timer event.
smua.trigger.endpulse.stimulus = pulsewidth_timer.EVENT_ID
-- Set the trigger model count to generate one 10-point pulse train.
smua.trigger.arm.count = 1
smua.trigger.count = 10

-- Turn on the SMU output and initiate the trigger model to output the pulse train.
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
-- Wait for the sweep to complete.
waitcomplete()

-- Turn off SMU output.
smua.source.output = smua.OUTPUT_OFF
```

Event blenders

The ability to combine trigger events is called event blending. You can use an event blender to wait for up to four input trigger events to occur before responding with an output event.

You set the event blender operation using remote commands. You cannot set them up through the front panel.

You can program up to four event blenders for the Model 2651A.

Event blender modes

Event blenders perform logical AND and logical OR functions on trigger events. For example, trigger events can be triggered when either a manual trigger or external input trigger is detected.

- **Or:** Generates an event when an event is detected on *any* one of the four stimulus inputs
- **And:** Generates an event when an event is detected on *all* of the assigned stimulus inputs

Set the `trigger.blender[N].orenable` attribute to configure the event blender mode. Setting the attribute to `true` enables OR mode; setting the attribute to `false` enables AND mode.

Assigning input trigger events

Each event blender has four stimulus inputs. You can assign a different trigger event ID to each stimulus input. The programming example below illustrates how to assign the source complete event ID of SMU A and the trigger event ID of digital I/O line 1 to stimulus inputs 1 and 2 of event blender 1:

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

Action overruns

Action overruns are generated by event blenders depending on the mode, as shown in the following table. Use the status model to monitor for the occurrence of action overruns. For details, see Status model (on page 15-1).

Action overruns

Mode	Action overrun
And	Generates an overrun when a second event on any of its inputs is detected before generating an output event.
Or	Generates an overrun when two events are detected simultaneously.

LAN triggering overview

Triggers can be sent and received over the LAN interface. The Model 2651A supports LAN extensions for instrumentation (LXI) and has eight LAN triggers that generate and respond to LXI trigger packets.

Understanding hardware value and pseudo line state

LAN triggering is similar to hardware synchronization except that LXI trigger packets are used instead of hardware signals. A bit in the LXI trigger packet called the hardware value simulates the state of a hardware trigger line. The Model 2651A stores the hardware value of the last LXI trigger packet that was sent or received as the pseudo line state.

The stateless event flag is a bit in the LXI trigger packet that indicates if the hardware value should be ignored. If it is set, the Model 2651A ignores the hardware value of the packet and generates a trigger event. The Model 2651A always sets the stateless flag for outgoing LXI trigger packets. If the stateless event flag is not set, the hardware value indicates the state of the signal.

Changes in the hardware value of consecutive LXI trigger packets are interpreted as edge transitions. Edge transitions generate trigger events. If the hardware value does not change between successive LXI trigger packets, the Model 2651A assumes an edge transition was missed and generates a trigger event. The following table illustrates edge detection in LAN triggering.

LXI trigger edge detection

Stateless event flag	Hardware value	Pseudo line state	Falling edge	Rising edge
0	0	0	Detected	Detected
0	1	0	-	Detected
0	0	1	Detected	-
0	1	1	Detected	Detected
1	-	-	Detected	Detected

Set the LAN trigger mode to configure the edge detection method in incoming LXI trigger packets. The mode that is selected also determines the hardware value in outgoing LXI trigger packets. The following table lists the LAN trigger modes.

LAN trigger modes

Trigger mode	Input detected	Output generated	Notes
Either edge	Either	Negative	
Falling edge	Falling	Negative	
Rising edge	Rising	Positive	
RisingA	Rising	Positive	Same as Rising edge
RisingM	Rising	Positive	Same as Rising edge
Synchronous	Falling	Positive	Same as SynchronousA
SynchronousA	Falling	Positive	
SynchronousM	Rising	Negative	

The programming example below illustrates how to configure the LAN trigger mode.

```
-- Set LAN trigger 2 to falling edge.
lan.trigger[2].mode = lan.TRIG_FALLING
```

Understanding LXI trigger event designations

LAN trigger objects generate LXI trigger events, which are LAN0 to LAN7 (zero based). In the command table, the LXI trigger events can be accessed using `lan.trigger[1]` through `lan.trigger[8]`.

`lan.trigger[1]` corresponds to LXI trigger event LAN0 and `lan.trigger[8]` corresponds to LXI trigger event LAN7.

Generating LXI trigger packets

You can configure the Model 2651A to output an LXI trigger packet to other LXI instruments.

To generate LXI trigger packets:

1. Call the `lan.trigger[N].connect()` function.
2. Select the event that triggers the outgoing LXI trigger packet by assigning the specific event ID to the LAN stimulus input.

Make sure to use the same LXI domain on both the Model 2651A instrument and the other instrument. If the Model 2651A has a different LXI domain than the instrument at the other end of the trigger connection, the LXI trigger packets are ignored by both instruments.

Command interface triggering

A command interface trigger occurs when:

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

Use `trigger.EVENT_ID` to monitor for command interface triggers. To ensure that commands and triggers issued over the command interface are processed in the correct order, a trigger event is not generated until:

- The trigger command is executed
- `trigger.wait()` retrieves the trigger command from the command queue before it would normally be executed

Command interface triggering does not generate action overruns. The triggers are processed in the order that they are received in the Model 2651A command queue. The Model 2651A only processes incoming commands when no commands are running. Unprocessed input triggers can cause an overflow in the command queue. It is important to make sure a script processes triggers while it is running.

NOTE

The command queue can fill up with trigger entries if too many `*TRG` messages are received while a test script is running, even if the script is processing triggers. You can avoid this by using the `localnode.prompts4882` attribute (see TSP command reference (on page 7-1) for more information), and by using `trigger.wait()` calls that remove the `*TRG` messages from the command queue. If the command queue fills with too many trigger entries, messages like `abort` are not processed.

Manual triggering

The TRIG key is used for manual triggering. Each time the TRIG key is pressed, a trigger event is generated. You can monitor for a manual trigger event using the event ID `display.trigger.EVENT_ID`. See Using the TRIG key to trigger a sweep for an example of how to use a manual trigger.

There are no action overruns for manual triggering.

Interactive triggering

The complexity of some test system configurations may not allow a static trigger setup. These configurations require more dynamic control of triggering than the static trigger setup provides. For such cases, a setup providing interactive trigger programming allows the generation and detection of trigger events that can be controlled on demand under remote control. For example, you can use interactive triggering when you need to make multiple source function changes or implement conditional branching to other test setups based on recent measurements.

Detecting trigger events using the wait() function

Most of the Model 2651A trigger objects, except for SMU trigger objects, have built-in event detectors that monitor for trigger events. The event detector only monitors events generated by that object and cannot be configured to monitor events generated by any other trigger object. Using the `wait()` function of the trigger object causes the Model 2651A instrument to suspend command execution until a trigger event occurs or until the specified timeout period elapses.

For example, use `trigger.blender[N].wait(Y)` to suspend command execution until an event blender generates an event, where *N* is the specific event blender and *Y* is the timeout period. After executing the `wait()` function, the event detector of the trigger object is cleared.

The following programming example illustrates how to suspend command execution while waiting for various events to occur:

```
-- Wait up to 10 seconds for a front-panel TRIG key press.
display.trigger.wait(10)
-- Wait up to 60 seconds for timer 1 to complete its delay.
trigger.timer[1].wait(60)
-- Wait up to 30 seconds for input trigger to digital I/O line 10.
digio.trigger[10].wait(30)
```

Using the assert function to generate output triggers

You can use certain trigger objects to generate output triggers on demand. These trigger objects are the digital I/O lines, TSP-Link synchronization lines, and the LAN.

The programming example below illustrates how to generate an output trigger using the assert function of the trigger object.

NOTE

Connection parameters and commands that establish a connection are not shown in this example.

```
-- Generate a falling-edge trigger on digital I/O line 3.
digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[3].assert()
-- Generate a rising edge trigger on TSP-Link sync line 1.
tsplink.trigger[1].mode = tsplink.TRIG_RISINGM
tsplink.trigger[1].assert()
-- Generate a LAN trigger on LAN pseudo line 6.
lan.trigger[6].mode = lan.TRIG_EITHER
lan.trigger[6].assert()
```

Using the release function of the hardware lines

Use the release function to allow the hardware line to output another external trigger when the pulse width is set to 0.

Setting the pulse width to 0 results in an indefinite length pulse when the assert function is used to output an external trigger. When an indefinite length pulse is used, the release function must be used to release the line before another external trigger can be output.

The release function can also be used to release latched input triggers when the hardware line mode is set to Synchronous. In Synchronous mode, the receipt of a falling edge trigger latches the line low. The release function releases this line high in preparation for another input trigger.

The programming example below illustrates how to output an indefinite external trigger.

```
-- Set digio line 1 to output an indefinite external trigger.
digio.trigger[1].mode = digio.TRIG_FALLING
digio.trigger[1].pulsewidth = 0
digio.trigger[1].assert()
-- Release digio line 1.
digio.trigger[1].release()
-- Output another external trigger.
digio.trigger[1].assert()
```

For information about hardware lines, see Digital I/O port and TSP-Link synchronization lines (on page 3-45).

Using the set function to bypass SMU event detectors

The set functions are useful whenever you want the source-measure unit (SMU) to continue operation without waiting for a programmed trigger event.

There is a set function for each SMU event detector. When called, the function immediately satisfies the event detector, allowing the SMU to continue through the trigger model.

For example, you can use a set function when you want the SMU to immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur. You can use a set function to start actions on the SMU if there is a missed trigger event.

The programming example below illustrates how to have the SMU immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur.

```
-- Immediately sets the arm event detector of SMU A
-- to the detected state.
smua.trigger.arm.set()
-- Sets the measure event detector of SMU A.
smua.trigger.measure.set()
```

Event detector overruns

If a second trigger event is generated before an event detector clears, the trigger object generates a detector overrun. You can check for detector overruns by reading the `overrun` attribute of the trigger object. The attribute is set to `true` when an overrun occurs. You can use the `clear()` function to immediately clear the event detector, discarding any history of previous trigger events. The `clear()` function also clears any detector overruns.

NOTE

Detector overruns are not the same as the action overruns that are reported in the status model.

The programming example below illustrates how to check for and respond to detector overruns.

```
testOver = digio.trigger[4].overrun
if testOver == true then
   print("Digital I/O overrun occurred.")
end
```

Examples using interactive triggering

The following examples demonstrate how to use interactive triggering.

Command interface interactive trigger example

The programming example below illustrates how to clear triggers, turn on the SMU output, and then enable a 30-second timeout to wait for a command interface trigger. When the trigger is received, the Model 2651A performs a voltage reading.

```
-- Clear any previously detected command interface triggers.
trigger.clear()
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Wait 30 seconds for a command interface trigger.
triggered = trigger.wait(30)
-- Get a voltage reading.
reading = smua.measure.v()
-- Send a command interface trigger to trigger the measurement.
*TRG
```

NOTE

`*TRG` cannot be used in a script.

Manual triggering example

The programming example below illustrates how to pause a script and prompt the operator to press the TRIG key when the operator is ready to continue. If the TRIG key is not pressed, the test continues after waiting 10 minutes (600 seconds).

```
display.clear()
display.trigger.clear()
display.setcursor(1, 1)
display.settext("Take a Break")
display.setcursor(2, 1)
display.settext("Press TRIG to continue")
display.trigger.wait(600)
display.clear()
```

Digital I/O triggering interactive example

The programming example below illustrates how to configure digital I/O line 2 as an input trigger and digital I/O line 14 as an output trigger. The Model 2651A to wait for an external input trigger on digital I/O line 2. If a trigger event occurs, the Model 2651A outputs an external trigger on digital I/O line 14. If no trigger event is received on digital I/O line 2, the test is aborted.

```
-- Configure digital I/O lines 2 and 14 for input trigger detection
-- and output trigger generation, respectively.
digio.trigger[2].mode = digio.TRIG_RISINGA
digio.trigger[2].clear()
digio.trigger[14].mode = digio.TRIG_FALLING
digio.trigger[14].pulsewidth = 0.0001
-- Wait 15 seconds for a trigger event to occur on digital I/O line 2.
trigInput = digio.trigger[2].wait(15)
-- If a trigger event occurs on digital I/O line 2, assert an output
-- trigger on digital I/O line 14. If a trigger event does
-- not occur, turn off the output of smua and issue a message
-- on the front-panel display.
if trigInput == true then
   digio.trigger[14].assert()
else
   smua.source.output = smua.OUTPUT_OFF
   display.screen = display.USER
   display.clear()
   display.setcursor(1, 1)
   display.settext("No trigger received. Test aborted.")
   exit()
end
```

Hardware trigger modes

You can use different hardware trigger modes for digital I/O and TSP-Link® synchronization. Use hardware triggers to integrate Keithley instruments and non-Keithley instruments in a test system. The Model 2651A supports 14 digital I/O lines and three TSP-Link synchronization lines that can be used for input or output triggering. For additional information about the hardware trigger modes, see TSP command reference (on page 7-1).

NOTE

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

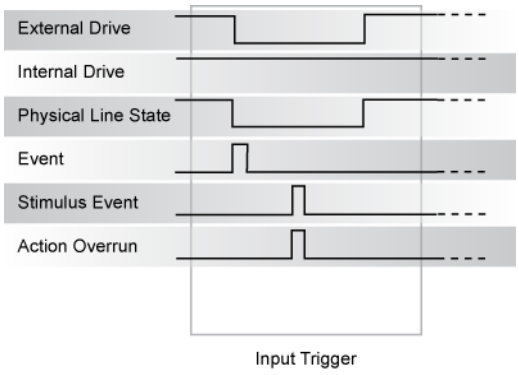
Falling edge trigger mode

The falling edge trigger mode generates low pulses and detects all falling edges. The figure titled "Falling edge input trigger" shows the characteristics of the falling edge input trigger; the figure titled "Falling edge output trigger" shows the falling edge output trigger.

Input characteristics:

- Detects all falling edges as input triggers.

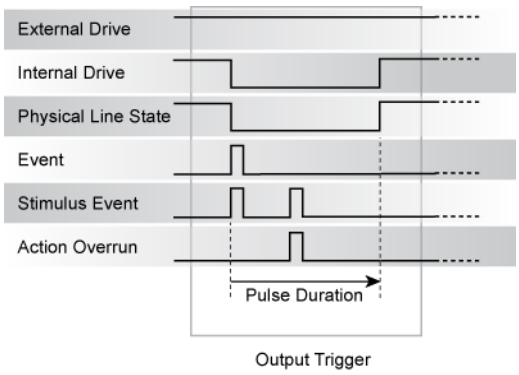
Figure 55: Falling edge input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse for the programmed pulse duration.
- An action overrun occurs if the physical line state is low and a source event occurs.

Figure 56: Falling edge output trigger



Rising edge master trigger mode

Use the rising edge master (RisingM) trigger mode (see the figure titled "RisingM output trigger") to synchronize with non-Keithley instruments that require a high pulse. Input trigger detection is not available in this trigger mode. You can use the RisingM trigger mode to generate rising edge pulses.

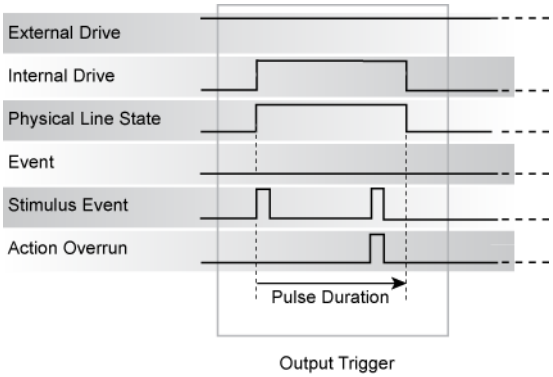
NOTE

The RisingM trigger mode does not function properly if the line is driven low by an external drive.

Output characteristics:

- Configured trigger events, as well as the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands, cause the physical line state to float high during the trigger pulse duration.
- An action overrun occurs if the physical line state is high when a stimulus event occurs.

Figure 57: RisingM output trigger



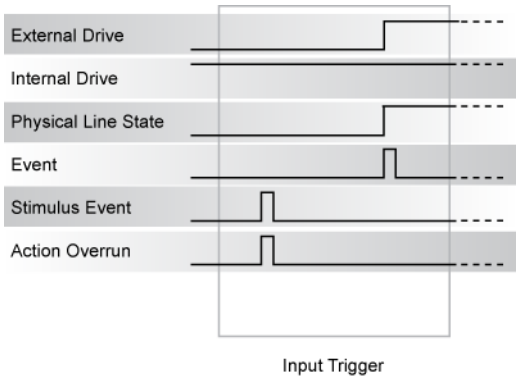
Rising edge acceptor trigger mode

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

Input characteristics:

- All rising edges generate an input event.

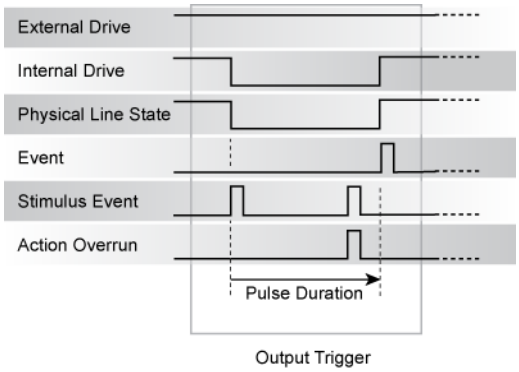
Figure 58: RisingA input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse that is similar to the falling edge trigger mode.

Figure 59: RisingA output trigger



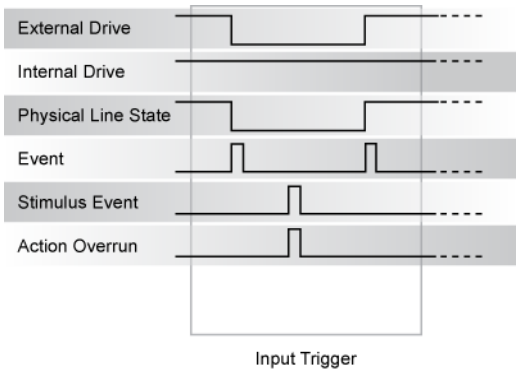
Either edge trigger mode

The either edge trigger mode generates a low pulse and detects both rising and falling edges.

Input characteristics:

- All rising or falling edges generate an input trigger event.

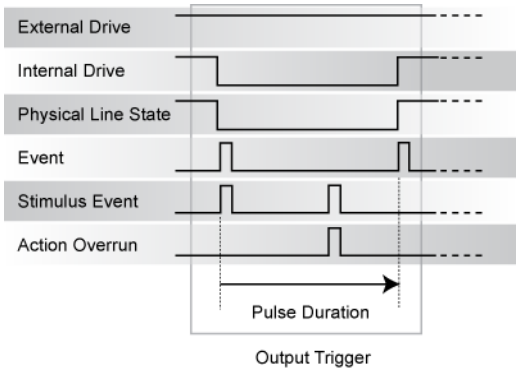
Figure 60: Either edge input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse that is similar to the falling edge trigger mode.
- An action overrun occurs if the physical line state is low while a stimulus event occurs.

Figure 61: Either edge output trigger



Understanding synchronous triggering modes

Use the synchronous triggering modes to implement bidirectional triggering, to wait for one node, or to wait for a collection of nodes to complete all triggered actions.

All non-Keithley instrumentation must have a trigger mode that functions similar to the SynchronousA or SynchronousM trigger modes.

To use synchronous triggering, configure the triggering master to SynchronousM trigger mode or the non-Keithley equivalent. Configure all other nodes in the test system to SynchronousA trigger mode or a non-Keithley equivalent.

Synchronous master trigger mode (SynchronousM)

Use the synchronous master trigger mode to generate falling edge output triggers, to detect the rising edge input triggers, and to initiate an action on one or more external nodes with the same trigger line.

In this mode, the output trigger consists of a low pulse. All non-Keithley instruments attached to the synchronization line in a trigger mode equivalent to SynchronousA must latch the line low during the pulse duration.

To use the SynchronousM trigger mode, configure the triggering master as SynchronousM and then configure all other nodes in the test system as Synchronous, SynchronousA, or to the non-Keithley Instruments equivalent.

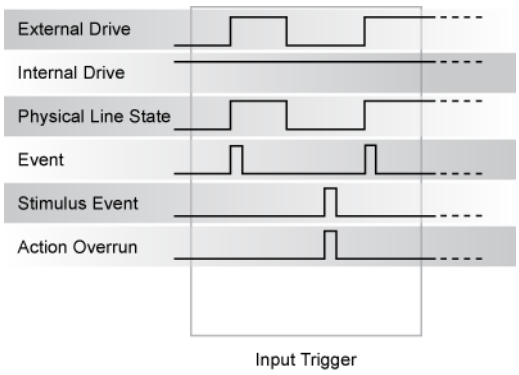
NOTE

Use the SynchronousM trigger mode to receive notification when the triggered action on all nodes is complete.

Input characteristics:

- All rising edges are input triggers.
- When all external drives release the physical line, the rising edge is detected as an input trigger.
- A rising edge is not detected until all external drives release the line and the line floats high.

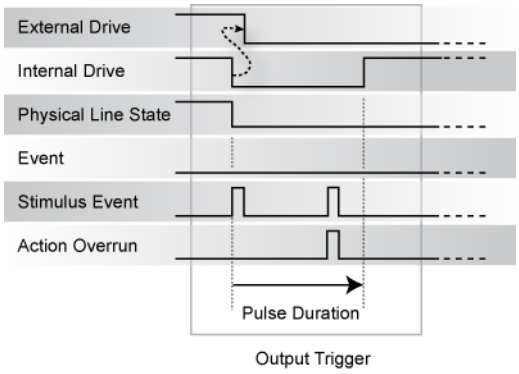
Figure 62: Synchronous master input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions generate a low pulse that is similar to the falling edge trigger mode.
- An action overrun occurs if the physical line state is low when a stimulus event occurs.

Figure 63: Synchronous master output trigger



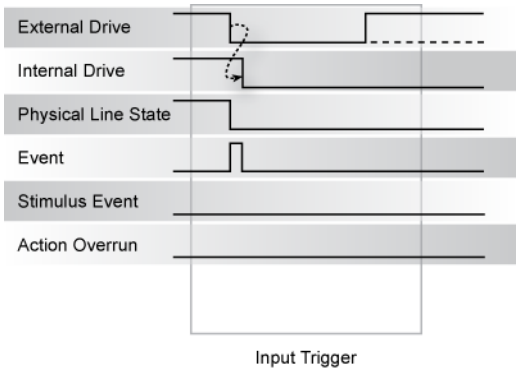
Synchronous acceptor trigger mode (SynchronousA)

Use the synchronous acceptor trigger mode (SynchronousA) on a trigger subordinate that operates with a trigger master configured for the SynchronousM trigger mode. The roles of the internal and external drives are reversed in the SynchronousA trigger mode.

Input characteristics:

- The falling edge is detected as the external drive pulses the line low, and the internal drive latches the line low.

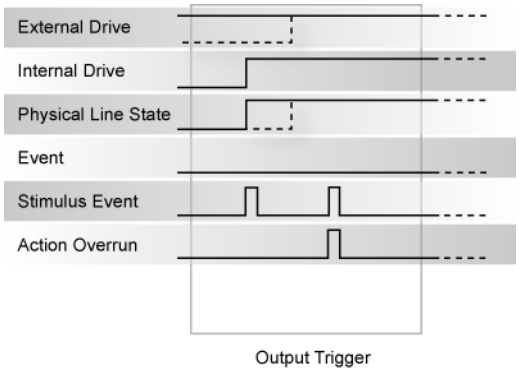
Figure 64: Synchronous acceptor input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions release the line if the line is latched low. The pulse width is not used.
- The physical line state does not change until all drives (internal and external) release the line.
- Action overruns occur if the internal drive is not latched low and a source event is received.

Figure 65: Synchronous acceptor output trigger



Synchronous trigger mode

The synchronous trigger mode is a combination of SynchronousA and SynchronousM trigger modes.

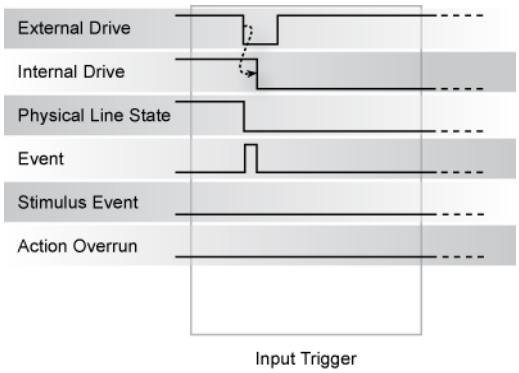
NOTE

Keithley Instruments recommends using SynchronousA and SynchronousM modes only.

Input characteristics:

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

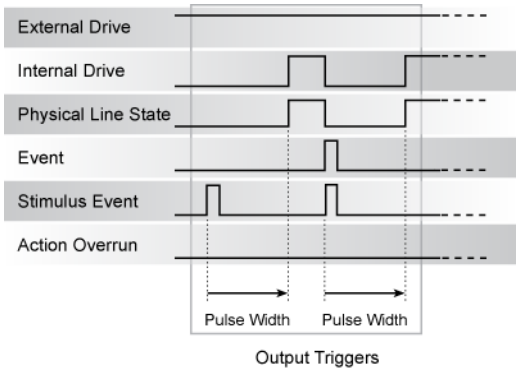
Figure 66: Synchronous input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions generate a low pulse for the programmed pulse duration if the line is latched low; a falling edge does not occur.
- A normal falling edge pulse generates when the internal drive is not latched low and the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions are issued.
- To mirror the SynchronousA trigger mode, set the pulse width to 1 µs or any small nonzero value.
- Action overruns are disabled.

Figure 67: Synchronous output trigger



High-capacitance mode

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

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

Based on the type, some test applications may require capacitors larger than 100 nF. For this purpose, you can use the high-capacitance mode to minimize overshoot, ringing, and instability.

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

Understanding high-capacitance mode

The source-measure unit (SMU) in the Model 2651A drives 10 nF of capacitance in normal operation. Typically, an internal capacitor across the current measuring element provides phase lead to compensate for the phase lag caused by the load capacitance on the output. This internal capacitance across the range resistance limits the speed for a specific measurement range.

The SMU in the Model 2651A implements frequency compensation to achieve the highest throughput possible for a 10 nF or less load. In addition, you must consider the settling time, voltage range, measure delay, the quality of the capacitor, the current measure range resistor, and the load resistor.

In normal operation, the SMU in the Model 2651A can drive capacitive loads as large as 10 nF. In high-capacitance mode, the SMU can drive a maximum of 50 µF of capacitance.

NOTE

When high-capacitance mode is enabled, a minimum load capacitance of 100 nF is recommended. In absence of this minimum load capacitance, overshoot and ringing may occur.

Highest throughput is achieved by using normal operation. In high-capacitance mode, the speed of the Model 2651A SMU is reduced to compensate for the larger load capacitance. Stability is achieved by inserting an internal capacitance across the current measuring element of the SMU. This internal capacitor limits the speed for the source and measurement ranges. Therefore, when optimizing the speed of your test configuration in high-capacitance mode, you must consider the settling time, voltage, and current ranges, measure delay, quality of the load capacitor, and load resistance.

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

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

Understanding source settling times

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

Model 2651A source settling times

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

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

Adjusting the voltage source

When driving large capacitive loads with high-capacitance mode enabled, the response time may be lengthened by the current limit. For example, see the table titled "Current measure and source settling times" in Understanding source settling times (on page 3-74). If a 1 µF capacitor charges to 10 V in 10 µs with a 1 A limit and the limit is set to 100 nA, the charging time is 100 seconds, as shown in the following equation.

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

The total response times while in high-capacitance mode are a combination of the time spent charging the capacitor (current limit) or the response time, whichever is greater. There is a direct relationship between the current limit and the charging time. As the current limit decreases, the amount of time required to charge the capacitor increases.

Understanding the capacitor

Based on the capacitor dielectric absorption, the settling time may change and the values in the "Current measure and source settling times" table in Understanding source settling times (on page 3-74) may differ.

NOTE

Tantalum or electrolytic capacitors are well known for long dielectric absorption settling times. Film capacitors and ceramics perform better, with NPO/COG dielectric ceramics yielding the best settling response.

Charging the capacitor and making readings

To charge and read a capacitor in high-capacitance mode:

1. Set the current limit to a value that is higher than the value that is used for the measurement (for example, if measuring at 10 µA, the initial current limit can be set to 1 A).
2. After the capacitor charges, lower the current limit and measure range to obtain the current measurement.

Enabling high-capacitance mode

Before enabling high-capacitance mode, note the following:

- It is important to read High-capacitance mode (on page 3-73) to understand the impact of high-capacitance mode.
- Test the device under test (DUT) and the capacitor to determine the best current limit and range of output voltages.
- The settling times can vary based on the DUT. It is important to test the limits of the DUT before you use high-capacitance mode.
- Failure to test the DUT for the appropriate current limit and output voltages can result in damage to or destruction of the DUT.
- For optimal performance, do not continuously switch between normal mode and high-capacitance mode.
- Before you charge the capacitor, start with 0 (zero) voltage across the capacitor.

Front panel

To enable high-capacitance mode from the front panel:

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

To enable high-capacitance mode using a remote interface:

Turning on high-capacitance mode has the following effects on the SMU settings:

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

Measuring current using high-capacitance mode

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

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

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

  Where: *i* is the `limiti` setting (current limit)
- **measurei:** Sets the current measure range
- **measuredelay:** Defines the delay to wait after lowering the current limit before making the measurement

Script example

Use the `smua.source.highc` attribute to set and control the options for high-capacitance mode.

The programming examples and figure below illustrate how to enable high-capacitance mode.

1. To enable high-capacitance mode, send:

```
-- Enables high-capacitance mode.
smua.source.highc = smua.ENABLE
```

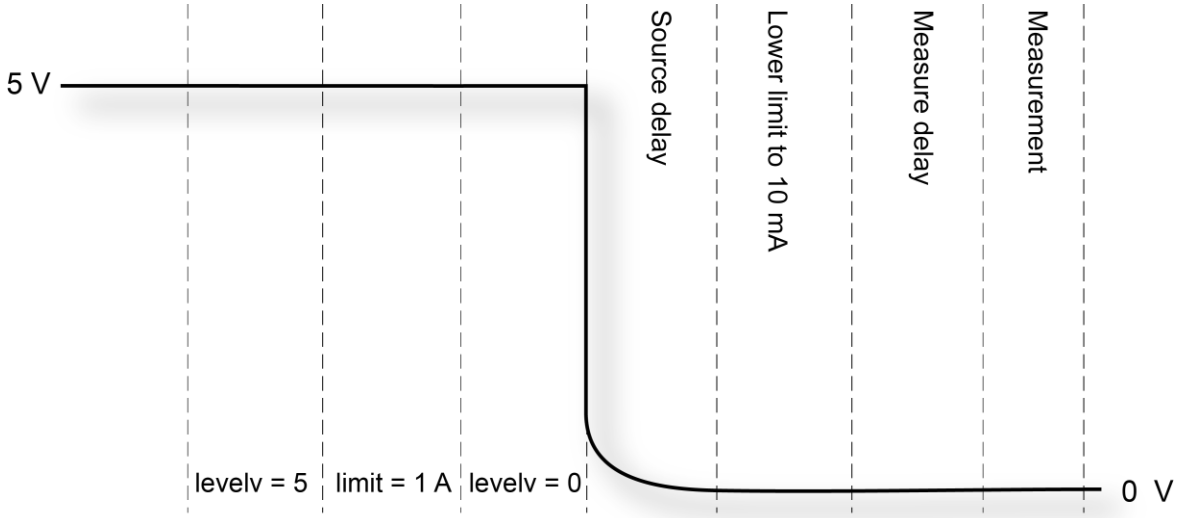
2. To run the `i_leakage_measure()` function in the KIHighC factory script, send:

```
-- Charges the capacitor.
smua.source.levelv = 5
smua.source.output = smua.OUTPUT_ON
delay(1)
imeas = i_leakage_measure(smua, 0, 1, 300e-3, 10e-6, 100e-3)
-- The parameters in the i_leakage_measure() function represent
-- the following:
-- smu = smua
-- levelv = 0 V
-- limiti = 1 A
-- sourcedelay = 300 ms
-- measurei = 10 uA range
-- measuredelay = 100 ms
```

NOTE

Adjust the voltage level and source delays based on the value and type of capacitor along with the magnitude of the voltage step and the current measure range.

Figure 68: Enabling high-capacitance mode



Display operations

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

Display functions and attributes

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

The TSP command reference (on page 7-1) provides additional information about the display functions and attributes.

Cross-referencing functions and attributes to section topics

Function or attribute	Section topic
`display.clear()`	Clearing the display (on page 3-81)
`display.getannunciators()`	Indicators (on page 3-87)
`display.getcursor()`	Cursor position (on page 3-82)
`display.getlastkey()`	Capturing key-press codes (on page 3-90)
`display.gettext()`	Displaying text messages (on page 3-83)
`display.inputvalue()`	Parameter value prompting (on page 3-86)
`display.loadmenu.add()` `display.loadmenu.delete()`	Load test menu (on page 3-88)
`display.locallockout`	LOCAL lockout (on page 3-88)
`display.menu()`	Menu (on page 3-85)
`display.prompt()`	Parameter value prompting (on page 3-86)
`display.screen`	Display screen (on page 3-80)
`display.sendkey()`	Sending key codes (on page 3-90)
`display.setcursor()`	Cursor position (on page 3-82)
`display.settext()`	Displaying text messages (on page 3-83)
`display.smua.digits`	Display resolution (on page 3-80)
`display.smua.measure.func`	Measurement functions (on page 3-80)
`display.trigger.clear()` `display.trigger.wait()`	Display trigger wait and clear (on page 3-80)

Display features

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

Display screen

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

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

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

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

Measurement functions

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

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

Limit functions

Configure the type of limit function displayed by setting the `display.smua.limit.func` attribute. The following programming example illustrates how to set SMU A to display its power limit setting:

```
display.smua.limit.func = display.LIMIT_P
```

Display resolution

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

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

Display trigger wait and clear

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

Display messages

You can define text messages that can be displayed on the front panel of the instrument. Most of the display functions and attributes that are associated with display messaging automatically select the user screen. The attribute for the display screen is explained in Display screen (on page 3-80).

For example, while a test is running, the following message can be displayed on the Model 2651A front panel:

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

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

NOTE

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

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

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

Clearing the display

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

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

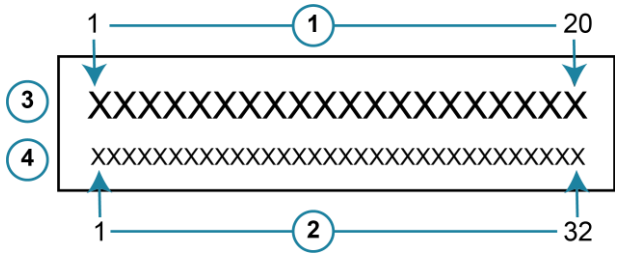
```
display.clear()
```

Cursor position

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

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

Figure 69: Row and column format for display messaging



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

The function to set cursor position has the following options:

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

Where:

row `1` or `2`

column `1` to `20` (row 1)
`1` to `32` (row 2)

style `0` (invisible)
`1` (blink)

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

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

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

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

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

Output:

```
2.00000e+00 1.00000e+00
```

Displaying text messages

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

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

Character codes

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

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

In addition to displaying alphanumeric characters, you can display other special characters. Refer to Display character codes (on page 16-1) for a listing of special characters and their corresponding codes.

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

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

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

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

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

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

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

NOTE

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

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

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

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

Returning a text message

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

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

Where:

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

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

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

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

Input prompting

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

Menu

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

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

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

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

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

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

Parameter value prompting

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

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

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

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

Where:

format — String that creates an editable input field on the user screen at the present cursor position (examples: `+0.00 00`, `+00`, `0.00000E+0`)
Value field:
+ = Include for positive/negative value entry; omitting the + prevents negative value entry
0 = Defines the digit positions for the value (up to six zeros (0))
Exponent field (optional):
E = include for exponent entry
+ = Include for positive/negative exponent entry; omitting the + prevents negative value entry
0 = Defines the digit positions for the exponent

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

min — Option to specify minimum limits for the input field

- When NOT using the "+" sign for the value field, the minimum limit cannot be set to less than zero
- When using the "+" sign, the minimum limit can be set to less than zero (for example, `-2`)

max — Option to specify maximum limits for the input field

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

help — Informational text string to display on the bottom line (32 characters maximum)

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

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

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

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

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

NOTE

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

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

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

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

Indicators

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

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

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

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

Bit identification for indicators

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

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

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

Not all the indicators shown in above table may be used by the Model 2651A.

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

```
1000010100001101
```

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

Local lockout

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

```
display.locallockout = lockout
```

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

`0` or `display.UNLOCK`

`1` or `display.LOCK`

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

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

Load test menu

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

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

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

User tests

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

Adding USER TESTS menu entries

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

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

Where:

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

Scripts, functions, and variables that are used in the *code* are not saved when `display.SAVE` is used. Functions and variables need to be saved with the script. If the script is not saved in nonvolatile memory, it is lost when the Model 2651A is turned off. See Example 1 below.

Example 1:

Assume a script with a function named `DUT1` has been loaded into the Model 2651A, and the script has not been saved in nonvolatile memory.

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

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

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

Now assume you turn the Model 2651A power off and then on again. Because the script was not saved in nonvolatile memory, the function named `DUT1` is lost. When `Test` is again run from the front panel, the beeper beeps, but `DUT1` does not execute because it is no longer in the runtime environment.

Example 2:

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

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

Deleting USER TESTS menu entries

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

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

Where:

displayname Name to delete from the menu.

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

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

Running a test from the front panel

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

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

Key-press codes

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

Sending key codes

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

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

Capturing key-press codes

A history of the key code for the last pressed front-panel key is maintained by the Model 2651A. When the instrument is turned on (or when transitioning from local to remote operation), the key code is set to `0` (`display.KEY_NONE`).

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

display.getlastkey()

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

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

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

Key codes

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

NOTE

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

display.waitkey()

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

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

After sending the `display.waitkey()` function, the script pauses and waits for the operator to press a front-panel key. For example, if the MENU key is pressed, the function returns the value `68`, which is the key code for that key. The key code values are the same as listed in display.getlastkey() (on page 7-75).

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

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

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

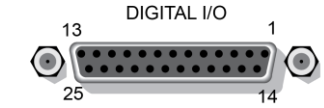
Digital I/O

The Model 2651A has a digital input/output port that can be used to control external digital circuitry. For example, you can use a handler that is used to perform binning operations with a digital I/O port.

Port configuration

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

Figure 70: Digital I/O port



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

Connecting cables for Trigger Link

Use a cable equipped with a male DB-25 connector (L-com part number CSMN25MF-5) to connect the digital I/O port to other Keithley Instruments models equipped with a Trigger Link (TLINK) interface.

Digital I/O lines

The port provides 14 digital I/O lines. Each output is set high (+5 V) or low (0 V) and can read high or low logic levels. Each digital I/O line is an open-drain signal.

+5 V output

The digital I/O port provides three +5 V dc output lines that you can use to drive external logic circuitry. Maximum combined current output for all lines is 250 mA. These lines are protected by a self-resetting fuse with a one-hour recovery time.

Output enable line

You can use the Model 2651A output enable (OE) line of the digital I/O with a switch in the test fixture or component handler. With proper use, power is removed from the device under test (DUT) when the lid of the fixture is opened. See Using output enable for more details.

WARNING

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

Interlock line

WARNING

At no time should you bypass the interlock feature of the Model 2651A. Safe operation requires a separate interlock circuit that meets the requirements of the application to reliably protect the operator from exposed voltages. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

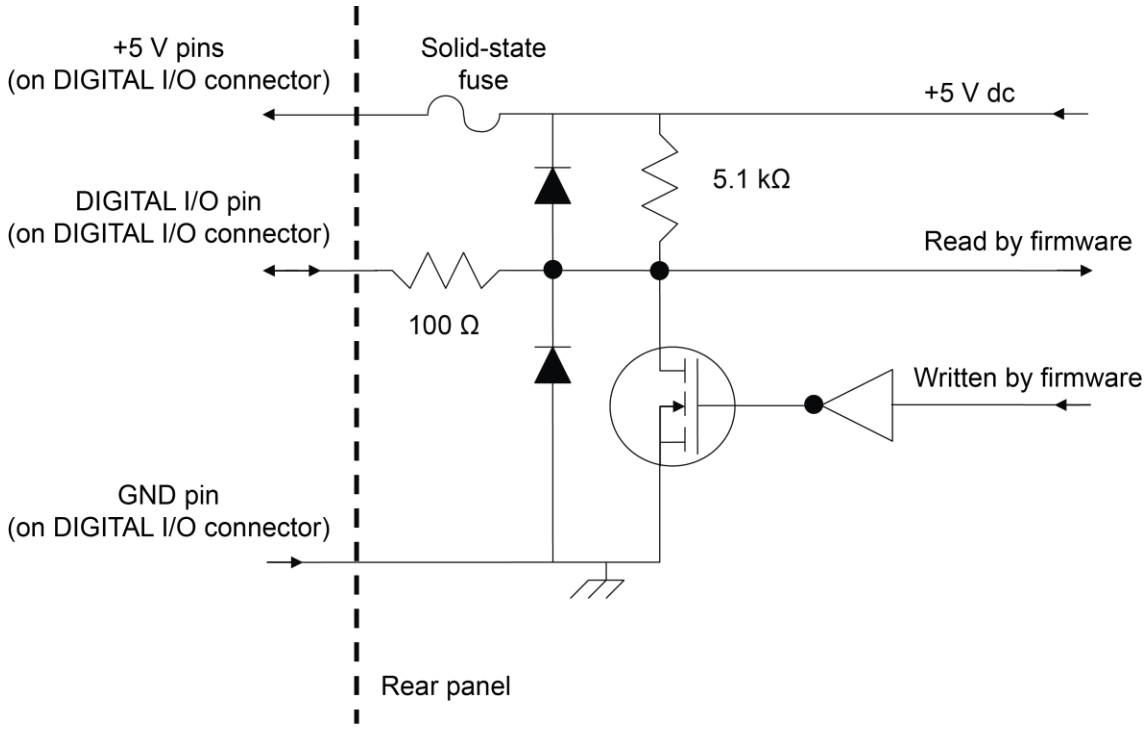
The interlock (INT) line of the digital I/O can be used with a switch in the test fixture or component handler. With proper use, power is removed from the DUT when the lid of the fixture is opened. See Interlock operation (on page 3-97, on page 3-98) and Interlock (on page 3-98) for more details.

Use interlock cable assembly CA-558 to connect the Model 2651A interlock to either a Model 8010 High Power Device Test Fixture or to the Model 2657A-LIM-3 LO Interconnect Module (refer to the connection information supplied with the device).

Digital I/O configuration

The following figure shows the basic configuration of the digital I/O port. Writing a 1 to a line sets that line high (~ +5 V). Writing a 0 to a line sets that line low (~0 V). Note that an external device pulls an I/O line low by shorting it to ground, so that a device must be able to sink at least 960 µA per I/O line.

Figure 71: Digital I/O interface schematic



Controlling digital I/O lines

Although the digital I/O lines are primarily intended for use with a device handler for limit testing, they can also be used for other purposes, such as controlling external logic circuits. You can control lines either from the front panel or over a remote interface.

NOTE

The trigger mode for the line must be set to `digio.TRIG_BYPASS` in order to use the line for digital I/O. See Triggering (on page 3-36) for more information.

To set digital I/O values from the front panel:

1. Press the **MENU** key, select **DIGOUT**, and then press the **ENTER** key or press the navigation wheel.
2. Select **DIG-IO-OUTPUT**, and then press the **ENTER** key or the navigation wheel.
3. Set the decimal value as required to set digital I/O lines in the range of 0 to 16,383 (see the table in Digital I/O bit weighting), and then press the ENTER key or the navigation wheel.
4. For example, to set digital I/O lines 3 and 8, set the value to 132.
   Press the **EXIT (LOCAL)** key as needed to return to the main menu.

To write-protect specific digital I/O lines to prevent their values from being changed:

1. Press the **MENU** key, then select **DIGOUT** and then press the **ENTER** key or the navigation wheel.
2. Select **WRITE-PROTECT**, and then press the **ENTER** key or the navigation wheel.
3. Set the decimal value as required to write-protect digital I/O lines within the range of 0 to 16,383 (see Digital I/O bit weighting (on page 3-95)), and then press the **ENTER** key or the navigation wheel.
   For example, to write-protect digital I/O lines 4 and 10, set the value to 520.
4. Press the **EXIT (LOCAL)** key as needed to return to the main menu.

To remove write protection, reset the decimal value to include only the lines that you want to write protect. To remove write protection from all lines, set the value to 0.

Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in the following table.

Digital bit weight

Line #	Bit	Decimal weighting	Hexadecimal weighting
1	B1	1	0x0001
2	B2	2	0x0002
3	B3	4	0x0004
4	B4	8	0x0008
5	B5	16	0x0010
6	B6	32	0x0020
7	B7	64	0x0040
8	B8	128	0x0080
9	B9	256	0x0100
10	B10	512	0x0200
11	B11	1,024	0x0400
12	B12	2,048	0x0800
13	B13	4,096	0x1000
14	B14	8,192	0x2000

Remote digital I/O commands

Commands that control and access the digital I/O port are summarized in the following table. See the TSP command reference (on page 7-1) for complete details on these commands. See the following table for decimal and hexadecimal values used to control and access the digital I/O port and individual lines. Use these commands to trigger the Model 2651A using external trigger pulses applied to the digital I/O port, or to provide trigger pulses to external devices.

Use these commands to perform basic steady-state digital I/O operations such as reading and writing to individual I/O lines or reading and writing to the entire port.

NOTE

You can use the digital I/O lines for both input and output. You must write a 1 to all digital I/O lines that are to be used as inputs.

Remote digital I/O commands

Command	Description
`digio.readbit(bit)`	Read one digital I/O input line
`digio.readport()`	Read digital I/O port
`digio.writebit(bit, data)`	Write data to one digital I/O output line
`digio.writeport(data)`	Write data to digital I/O port
`digio.writeprotect = mask`	Write protect mask to digital I/O port

Digital I/O programming example

The programming commands below illustrate how to set bit B1 of the digital I/O port high, and then read the entire port value.

```
digio.trigger[1].mode = digio.TRIG_BYPASS
-- Set bit B1 high.
digio.writebit(1,1)
-- Read digital I/O port.
data = digio.readport()
```

Using output enable

The Model 2651A digital I/O port provides an output enable line for use with a test fixture switch. When properly used, the output of the System SourceMeter® instrument turns OFF when the lid of the test fixture is opened. See DUT Test Connections (on page 2-45) for important safety information when using a test fixture.

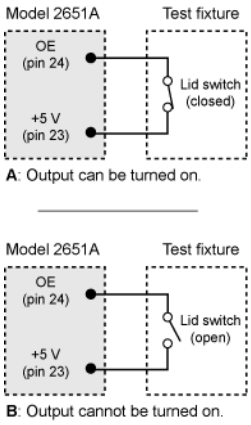
⚠ WARNING

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

Operation

When enabled, the output of the Model 2651A can only be turned on when the output enable line is pulled high through a switch to +5 V (as shown). If the lid of the test fixture opens, the switch opens, and the output enable line goes low, turning the output of the System SourceMeter® instrument off. The output will not automatically turn on when output enable is set high. The output cannot be turned back on until +5 V is applied to the output enable line.

Figure 72: Using the output enable line



Front-panel control of output enable

To activate the output enable line from the front panel:

1. Press the **CONFIG** key followed by the **OUTPUT ON/OFF** control.
2. Choose **DIO-CONTROL**, then press the **ENTER** key or the navigation wheel.
3. To activate the output enable signal, select **OE_OUTPUT_OFF**. This causes the source-measure unit (SMU) output to be blocked if the output enable is not asserted (connected to +5 V).
   To deactivate the output enable signal, select **NONE**. The state of the output enable signal has no effect on the SMU output.
4. Press the **EXIT (LOCAL)** key as needed to return to the normal display.

Remote control of output enable

Use one of these commands to control output enable action:

```
smua.source.outputenableaction = smua.OE_NONE
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

When set to smua.OE_NONE, the Model 2651A does not take action when the output enable line is low. When set to smua.OE_OUTPUT_OFF, the instrument turns the output off as if the smua.source.output = smua.OUTPUT_OFF command was received. The instrument does not automatically turn its output on when the output enable line returns to the high state. For example, the following command activates the output enable for SMU A:

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

Interlock

WARNING

The Model 2651A is provided with an interlock circuit that must be positively activated 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.

Overview

The digital I/O port provides an interlock line for use with a test fixture switch. When properly used, the output of the SourceMeter instrument will turn off when the lid of the test fixture is opened. See DUT Test Connections (on page 2-45) for important safety information when using a test fixture. Follow standard safety and electrical practices by verifying the correct operation of all components related to system safety, including the interlock.

Operation

can only be turned on when the interlock line is driven high through a switch to +5 V (as shown). If the lid of the test fixture opens, the switch opens, and the interlock line goes low, turning the output of the High Power System SourceMeter® Instrument off. The output is not automatically turned on when the interlock line is set high. The output cannot be turned back on until the interlock line is set high.

TSP-Link trigger lines

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

Connecting to the TSP-Link system

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

Using TSP-Link trigger lines for digital I/O

Each trigger line is an open-drain signal. When using the TSP-Link® trigger lines for digital I/O, any node that sets the programmed line state to zero (0) causes all nodes to read 0 from the line state. This occurs regardless of the programmed line state of any other node. Refer to the table in Digital I/O bit weighting (on page 3-95) for digital bit-weight values.

Remote TSP-Link trigger line commands

Commands that control and access the TSP-Link® trigger line port are summarized in the following table. See the TSP command reference (on page 7-1) for complete details on these commands. See the table in Digital I/O bit weighting (on page 3-95) for the decimal and hexadecimal values used to control and access the digital I/O port and individual lines.

Use the commands in following table to perform basic steady-state digital I/O operations; for example, you can program the Model 2651A to read and write to a specific TSP-Link trigger line or to the entire port.

NOTE

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

NOTE

The trigger mode for the line must be set to `tsplink.TRIG_BYPASS` in order to use the line for digital I/O. See Triggering (on page 3-36) for more information.

Remote trigger line commands

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

Programming example

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

```
-- Set the TSP-Link trigger line to the trigger bypass mode.
tsplink.trigger[1].mode = tsplink.TRIG_BYPASS
-- Set bit B1 high.
tsplink.writebit(1, 1)
-- Read I/O port.
data = tsplink.readport()
```

Section 4

Theory of operation

In this section:

Analog-to-digital converter

The Model 2651A has two analog-to-digital converters (ADC): An integrating ADC and a fast ADC.

The integrating ADC uses a ratiometric analog-to-digital conversion technique. Depending on the configuration of the integrating ADC, periodic fresh reference measurements are required to minimize drift. The measurement aperture is used to determine the time interval between these measurement updates. For additional information, see Autozero (on page 2-25). To help optimize operation of this ADC, the instrument caches the reference and zero values for up to ten of the most recent number of power line cycles. For additional information, see NPLC caching (on page 2-27).

The fast ADC can acquire measurements at speeds up to 1 million samples per second. The fast ADC does not take reference measurements. A reading measurement acquisition buffer allows up to 5,000 readings to be made at the maximum acquisition rate of the fast ADCs. If this buffer is filled, the instrument slows its acquisition rate to the rate at which the instrument can process the data.

Data acquisition takes priority over both source operation and display operation. Sustained high data acquisition rates will cause the display to stop updating. A sustained high data acquisition rate during a sweep or pulse train may slow source operations. This can cause erratic sweep or pulse timing and may lead to triggers being missed (trigger overruns). Use the status model to monitor for trigger overruns. If a high sustained data acquisition rate causes undesirable sweep or pulse timing, reduce the data acquisition rate or reduce the total number of measurements until the optimal sweep or pulse timing is achieved.

Source-measure concepts

This section provides detailed information about source-measure concepts, including:

Limit principles

A limit acts as a clamp. If the output reaches the limit value, the High Power System SourceMeter® Instrument attempts to prevent the output from exceeding that value. This action implies that the source will switch from a V-source to an I-source (or from an I-source to a V-source) when a limit is reached.

As an example, assume the following:

System SourceMeter® instrument: V_{SRC} = 10 V; I_{LIMIT} = 10 mA

Device under test (DUT) resistance: 10 Ω

With a source voltage of 10 V and a DUT resistance of 10 Ω, the current through the DUT should be: 10 V / 10 Ω = 1 A. However, because the limit is set to 10 mA, the current will not exceed that value, and the voltage across the resistance is limited to 100 mV. In effect, the 10 V voltage source is transformed into a 10 mA current source.

In steady-state conditions, the set limit will restrict the Model 2651A output. This holds true except for the limit conditions, as described in Limits (on page 2-20) or for fast transient load conditions.

The Model 2651A can also be set to limit power. This limit can be set in addition to any voltage or current limits specified. The power limit restricts power by lowering the present limit in effect (voltage or current) as needed to restrict the SMU from exceeding the specified power limit. The limit operation of the instrument changes dependent on the source mode (current or voltage), load, and the configured limits (current, voltage, and power). For additional details on using limits, including load considerations when specifying both a current (or a voltage) limit and a power limit, see the Operating boundaries (on page 4-5) topic.

For information on implementing limits, see Setting the limit (on page 2-22).

Overheating protection

Proper ventilation is required to keep the High Power System SourceMeter® Instrument from overheating. Even with proper ventilation, the instrument can overheat if the ambient temperature is too high or the High Power System SourceMeter® Instrument is operated in sink mode for long periods. The instrument has an overtemperature protection circuit that turns the output off if the instrument overheats. When the overtemperature protection circuit turns the output off, a message indicating this condition is displayed. You cannot turn the output on until the instrument cools down.

Power equations to avoid overheating

To avoid overheating, do not operate the instrument in a manner that forces the instrument to exceed the maximum duty cycle (DC_{MAX}), which is computed using the General power equation (on page 4-3) below. Factors such as the ambient temperature, quadrant of operation, and high-power pulse levels (if applicable) affect the maximum duty cycle. Exceeding the calculated maximum duty cycle may cause the temperature protection mechanism to engage. When this happens, an error message displays and the instrument output is disabled until the internal temperature of the instrument is reduced to an acceptable level.

You do not have to be concerned about overheating if the following conditions are true:

- The instrument is used as a power source and not a power sink.
- The ambient temperature is ≤ 30 °C.
- Extended operating area pulsing is not being performed.

However, if any one of these is false, the instrument may overheat if operated in a manner that exceeds the calculated maximum duty cycle, DC_{MAX}.

The maximum duty cycle equation is derived from the power equation below by solving for DC_{MAX}. The general power equation describes how much power an instrument channel can source and sink before the total power cannot be fully dissipated by the cooling system of the instrument. This equation incorporates all the factors that can influence the power dissipated by the instrument.

General power equation

$$|(V_{OA} - V_P)(I_P)|\sqrt{DC_{MAX}} + |(V_{OA} - V_B)(I_B)| \leq (P_{CS} - P_{DER})$$

V_{OA} The instrument output amplifier voltage. This constant can be found in the table in Maximum duty cycle equation (on page 4-4).

V_P The voltage level the instrument is attempting to force while at the pulse level.
When operating in quadrants 1 or 3 (sourcing power), the sign of this voltage must be positive when used in the power equations.
When operating in quadrants 2 or 4 (sinking power), the sign of this voltage must be negative when used in the power equations.

I_P The current flowing through the instrument channel while at the pulse level.

V_B The voltage level the instrument is attempting to force while at the bias level.
When operating in quadrants 1 or 3 (sourcing power), the sign of this voltage must be positive when used in the power equations.
When operating in quadrants 2 or 4 (sinking power), the sign of this voltage must be negative when used in the power equations.

I_B The current flowing through the instrument channel while at the bias level.

P_{CS} The maximum power generated in an instrument channel that can be properly dissipated by the instrument cooling system, measured in watts. For the Model 2651A, this constant equals 370.

P_{DER} = $3(T_{AMB} - 30)$
This factor represents the number of watts the instrument is derated when operating in environments above 30 °C. The maximum output power is reduced by 3 W per degree C above 30 °C.
P_{DER} is 0 when the ambient temperature is below 30 °C.

T_{AMB} The ambient temperature of the instrument operating environment.

Maximum duty cycle equation

The following equation applies to both channels, sinking or sourcing power simultaneously. If a duty cycle less than 100% is required to avoid overheating, the maximum on-time must be less than 10 seconds.

$$DC_{MAX} \leq \left[\frac{(P_{CS} - P_{DER}) - |(V_{OA} - V_B)(I_B)|}{|(V_{OA} - V_P)(I_P)|}\right]^2 \times 100$$

NOTE

When attempting to determine the maximum duty cycle, where the off state will be 0 V or 0 A:

- I_B is 0
- I_P and V_P are the voltage and current levels when the instrument is on

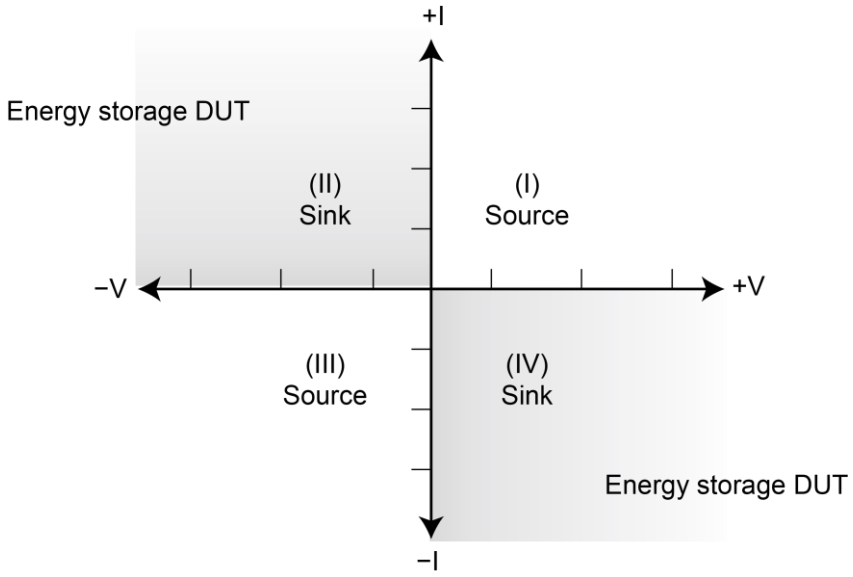
Model 2651A maximum duty cycle equation constants

Constant	100 mV range	1 V range	10 V range	20 V range	40 V range
V_{OA}	18.5	18.5	18.5	27	64

Operating boundaries

Depending on how the instrument is programmed and what is connected to the output (load or source), the instrument can operate in any of the four quadrants. The four quadrants of operation are shown in the following figure. When operating in the first (I) or third (III) quadrant, the instrument operates as a source (voltage and current have the same polarity). As a source, the instrument delivers power to a load.

Figure 73: Four quadrants of operation

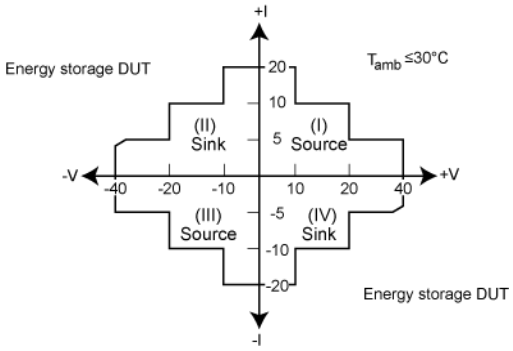


When operating in the second (II) or fourth (IV) quadrant, the instrument is operating as a sink (voltage and current have opposite polarity). As a sink, it is dissipating power rather than sourcing it. An external source or an energy storage device, such as a capacitor or battery, can force operation in the sink region.

Continuous power operating boundaries

The general operating boundaries for Model 2651A continuous power output are shown in the following figure. For derating factors, see the General power equation (on page 4-3). In this drawing, the illustrated voltage and current magnitudes are nominal values. Also note that the boundaries are not drawn to scale.

Figure 74: Model 2651A continuous power operating boundaries



Operation as a sink

When operating the Model 2651A in the second or fourth quadrant, the SMU operates as a load that sinks and dissipates the power internally. The ability of the SMU to dissipate power is defined by the boundaries shown in the previous figure. When operating the Model 2651A in the second or fourth quadrant, the DUT would be a power source (such as a battery, solar cell, or a power supply).

CAUTION

Use care when connecting a source to the Model 2651A that is capable of exceeding 120 mA. Using the Model 2651A to sink more than 120 mA can damage the instrument and invalidate your warranty.

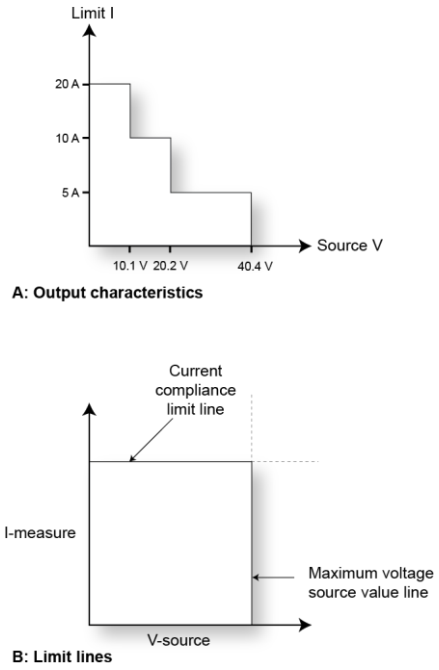
When operating as a sink within the continuous operating boundaries, the SMU operates as programmed. Programming source or compliance values will not cause the SMU to operate outside of these boundaries. The SMU treats these boundaries as a compliance limit known as the sink limit; the sink limit cannot be programmed by the user. The SMU deters operation outside of its sink limit by reducing its voltage. If the sink limit is reached, the source field on the display flashes and the sink limit bit is set in measurement event register of the status model (see Measurement event registers (on page 15-7)). For examples of instrument configuration in sink mode, see I-source sink operating boundaries (on page 4-18) and V-source sink operating boundaries (on page 4-11).

V-source operating boundaries

Model 2651A V-source operating boundaries

The following figure shows the operating boundaries for the V-source. Only the first quadrant of operation is shown; operation in the other three quadrants is similar with respect to the Continuous power operating boundaries (on page 4-6).

Figure 75: Model 2651A V-source boundaries



The first graph in the figure (labeled "A: Output characteristics"), shows the output characteristics for the V-source. As shown, the Model 2651A can continuously output up to 40.4 V at 5 A, up to 20.2 V at 10 A, or up to 10.1 V at 20 A. Note that when continuously sourcing more than 10.1 V, current is limited to 10 A and when continuously sourcing more than 20.2 V, current is limited to 5 A.

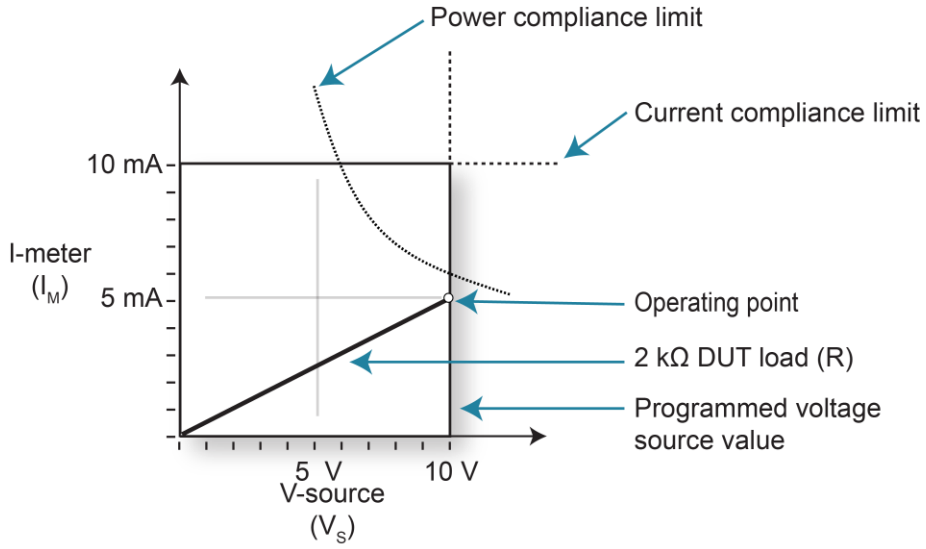
The second graph in the figure (labeled "B: Limit lines"), shows the operating area of the V source on a given range. The maximum voltage source value line shows the maximum possible source voltage for the selected voltage range. For example, if you are using the 20 V source range, the maximum voltage source value line is at 20.2 V. The current compliance limit line represents the actual compliance limit in effect (see Limit principles (on page 4-2)). The lines drawn are boundaries that represent the operating limits of the High Power System SourceMeter® Instrument for this quadrant of operation. The operating point can be anywhere inside (or on) these lines. The boundaries for the other quadrants are similar with respect to the Continuous power operating boundaries (on page 4-6).

Load considerations (V-source)

The boundaries within which the Model 2651A operates depend on the load (device-under-test, or DUT) that is connected to the output. The following figures show operation examples for resistive loads that are 2 kΩ and 800 Ω, respectively. For these examples, the SMU is programmed to source 10 V and limit current (10 mA).

In the following figure, the SMU is sourcing 10 V to the 2 kΩ load and subsequently measures 5 mA. The SMU is programmed to limit power (60 mW). As shown, the load line for 2 kΩ intersects the 10 V voltage source line at 5 mA. The current compliance limit and the power compliance limit are not reached (the SMU is not limited through its compliance settings).

Figure 76: Normal voltage source operation

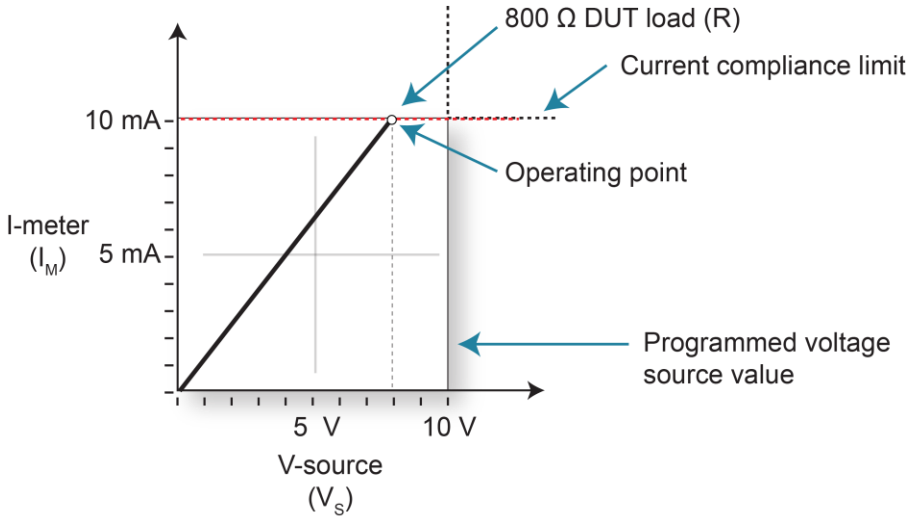


I_M = V_S / R
= 10 V / 2 kΩ)
= 5 mA

The following figure shows what happens if the resistance of the load is decreased to 800 Ω. The DUT load line for 800 Ω intersects the current compliance limit line, which places the SMU in compliance. When in compliance, the SMU cannot source its programmed voltage (10 V). For the 800 Ω DUT, the SMU only outputs 8 V (at the 10 mA limit).

Notice that as resistance decreases, the slope of the DUT load line increases. As resistance approaches infinity (open output), the SMU sources virtually 10 V at 0 mA. Conversely, as resistance increases, the slope of the DUT load line decreases. At zero resistance (shorted output), the SMU sources virtually 0 V at 10 mA.

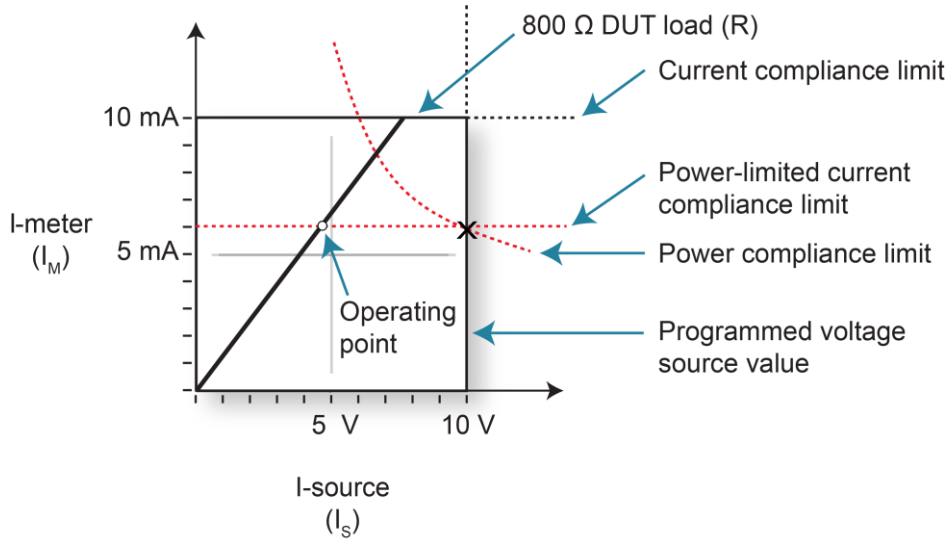
Figure 77: Voltage source operation in current compliance



$V_S = I_M \times R$
$= (10\ mA)(800\ \Omega)$
$= 8\ V$

The following figure shows what happens if a power limit of 60 mW is applied. As the SMU attempts to output the programmed source value of 10 V, the power compliance limit line is reached, which places the SMU in power compliance. The SMU enforces the power compliance limit by setting the current compliance limit line to the new power-limited current compliance limit line setting, which in this example is 6 mA. In compliance, the SMU cannot source its programmed voltage (10 V). For the 800 Ω DUT, the SMU outputs 4.8 V at the 5 mA limit. In this example, current never exceeds the programmed compliance of 10 mA or the programmed power compliance of 60 mW under any load.

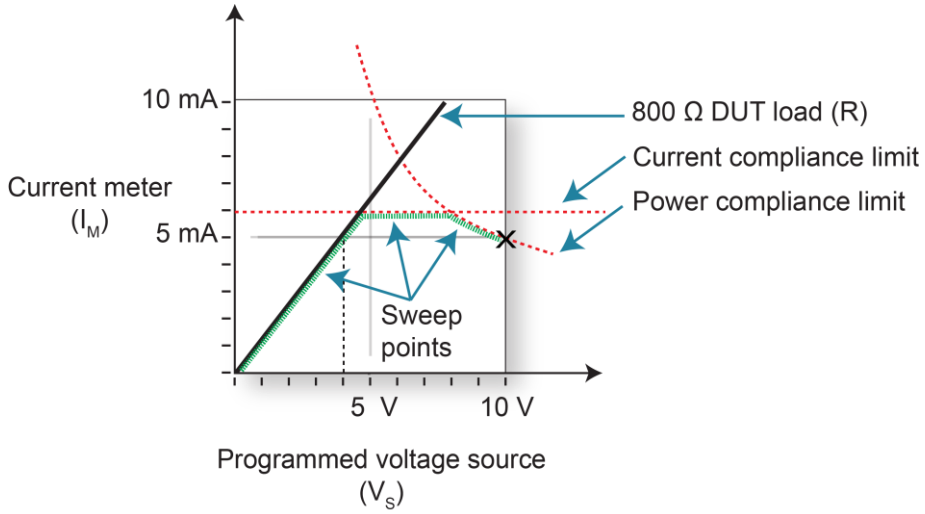
Figure 78: Voltage source operation in power compliance



V_S = $I_M \times R$
= (6 mA)(800 Ω)
= 4.8 V

The following figure shows a voltage sweep on a resistive load of 800 Ω. For this example, the SMU is programmed to sweep voltage to 10 V, limit current (6 mA), and limit power (50 mW). When sweeping, the actual source output varies according to the programmed source value until the current limit is reached. As the figure shows, the output sources the programmed value until placed in current compliance at the 6 mA limit. The sweep then continues (programmed current source values increase along the green sweep points line), but the output remains at the same value as when the SMU went into voltage compliance. This continues until the programmed source value sweeps to a high enough level that the power limit line is reached (50 mW). At this point, the current and voltage start to decrease, lowering the current and voltage values along the DUT load line. When the last point is swept (10 V), the actual output is 4 V at 5 mA.

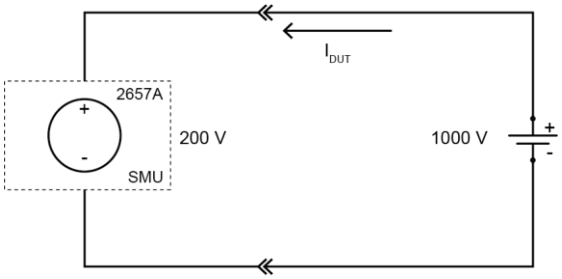
Figure 79: Programmed voltage source sweep operation in current and power compliance



V-source sink operating boundaries

The quadrant within which the Model 2651A operates depends on the device-under-test (DUT) connected to the Model 2651A output. The following example illustrates this operation by using the Model 2651A configured as a voltage source to discharge a 1000 V power source (a battery).

Figure 80: Sourcing voltage while sinking current



NOTE

The current compliance limit applies both to positive and negative currents. For example, if you set the current compliance limit to 50 mA, the current limit applies to ±50 mA.

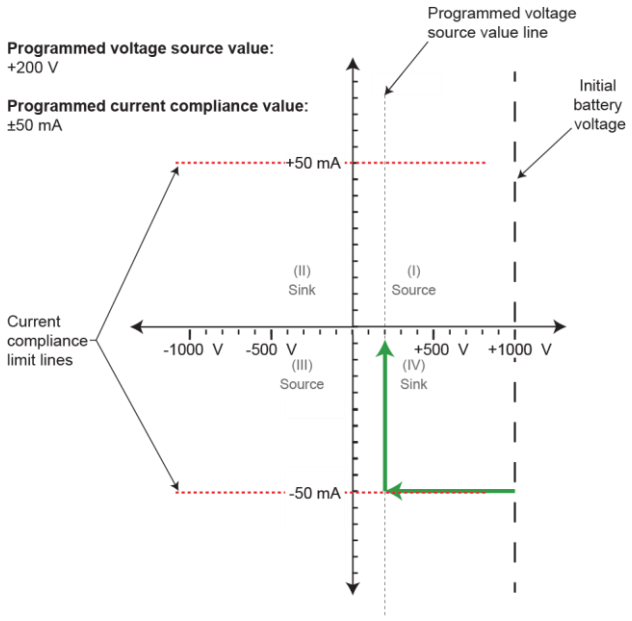
For this example, the Model 2651A is programmed to source 200 V and to limit current to 50 mA. When the SMU turns on, the battery voltage is higher than the programmed voltage source value. Since the SMU is unable to deliver the programmed voltage, the SMU is placed in current compliance and begins to sink current. Sink operation continues until the battery voltage equals the programmed voltage source level and the current in the circuit drops to nearly 0 A.

In the following figure, as the battery drains, the battery voltage is lowered (shown by the green arrow in the figure). Operation will continue in this direction until the SMU is able to deliver the programmed voltage source value.

NOTE

Since the battery is a power source, initial operation can occur anywhere along the initial battery voltage line. This voltage is only limited by the capability of the battery (see the following figure).

Figure 81: Considerations when sourcing voltage and sinking power

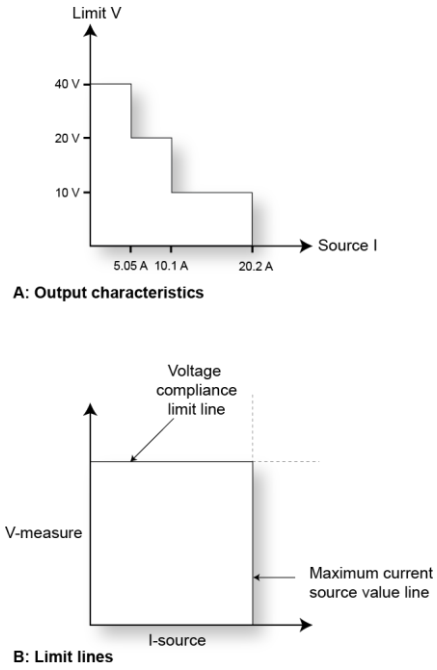


I-source operating boundaries

Model 2651A I-source operating boundaries

The following figure shows the operating boundaries for the I-source. Only the first quadrant of operation is shown; operation in the other three quadrants is similar with respect to the Continuous power operating boundaries (on page 4-6).

Figure 82: Model 2651A I-source boundaries



The first graph in the figure, labeled "A: Output characteristics," shows the output characteristics for the I-source. As shown, Model 2651A instruments can continuously output up to 20.2 A at 10 V, up to 10.1 A at 20 V, or up to 5.05 A at 40 V. Note that when continuously sourcing more than 5.05 A, voltage is limited to 20 V, and when sourcing more than 10.1 A, voltage is limited to 10 V.

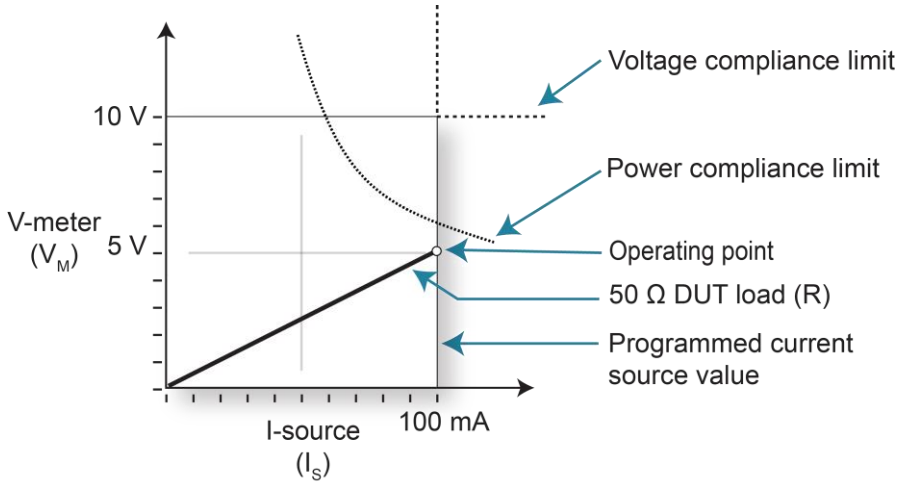
The second graph in the figure, labeled "B: Limit lines," shows the operating area of the I source on a given range. The maximum current source value line shows the maximum possible source current for the selected current range. The voltage compliance limit line represents the actual compliance limit that is in effect (see Compliance limit principles (on page 4-2)). These lines are boundaries that represent the operating limits of the System SourceMeter instrument for this quadrant of operation. The operating point can be anywhere inside (or on) these lines. The boundaries for the other quadrants are similar with respect to the Continuous power operating boundaries (on page 4-6).

Load considerations (I-source)

The boundaries within which the SMU operates depend on the load (device-under-test, or DUT) that is connected to its output. The following figures shows operation examples for resistive loads that are 50 Ω and 200 Ω. For these examples, the SMU is programmed to source 100 mA and limit voltage (10 V).

In the following figure, the SMU is sourcing 100 mA to the 50 Ω load and subsequently measures 5 V. The SMU is also programmed to limit power to 600 mW. As shown, the load line for 50 Ω intersects the 100 mA current source line at 5 V. The voltage compliance limit and the power compliance limit are not reached (the SMU is not limited through its compliance settings).

Figure 83: Normal current source operation

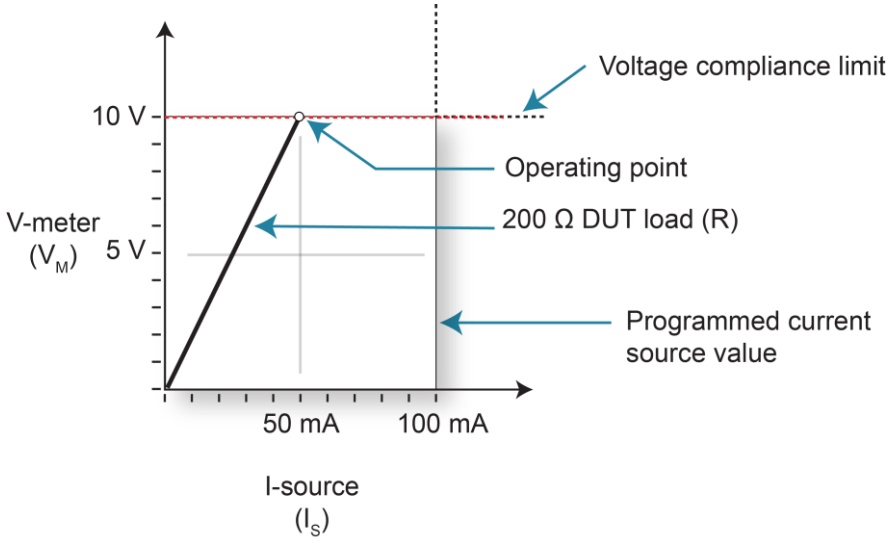


$V_M = I_S \times R$
$= (100\ mA)(50\ \Omega)$
$= 5\ V$

The following figure shows what happens if the resistance of the load is increased to 200 Ω. The DUT load line for 200 Ω intersects the voltage compliance limit line, which places the SMU in voltage compliance. In compliance, the SMU cannot source the programmed current of 100 mA. For the 200 Ω DUT, the SMU only outputs 50 mA at the 10 V limit.

As resistance increases, the slope of the DUT load line increases. As resistance increases and approaches infinity (open output), the SMU sources virtually 0 mA at 10 V. Conversely, as resistance decreases, the slope of the DUT load line decreases. At zero resistance (shorted output), the SMU sources 100 mA at virtually 0 V.

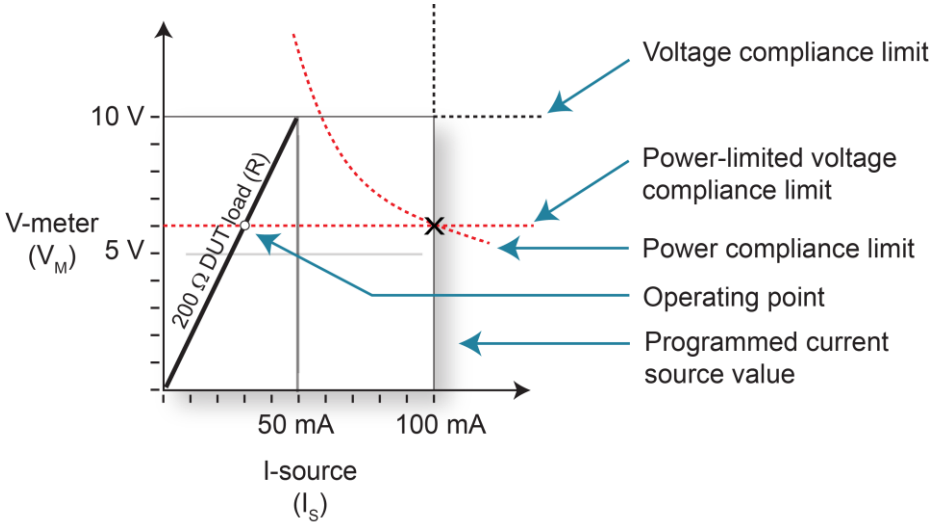
Figure 84: Current source operation in voltage compliance



I_S = V_M / R
= 10 V / 200 Ω)
= 50 mA

The following figure shows what happens if a power limit of 600 mW is applied. As the SMU attempts to output the programmed source value of 100 mA, the power-limited voltage compliance limit line is reached, which places the SMU in power compliance. The SMU enforces the power compliance limit by setting the voltage compliance limit to the new power-limited voltage compliance limit setting, which in this case is 6 V. In compliance, the SMU cannot source its programmed current of 100 mA. For the 200 Ω DUT, the SMU only outputs 30 mA at the 6 V limit. In this example, voltage never exceeds the programmed compliance of 10 V or the programmed power compliance of 600 mW under any load.

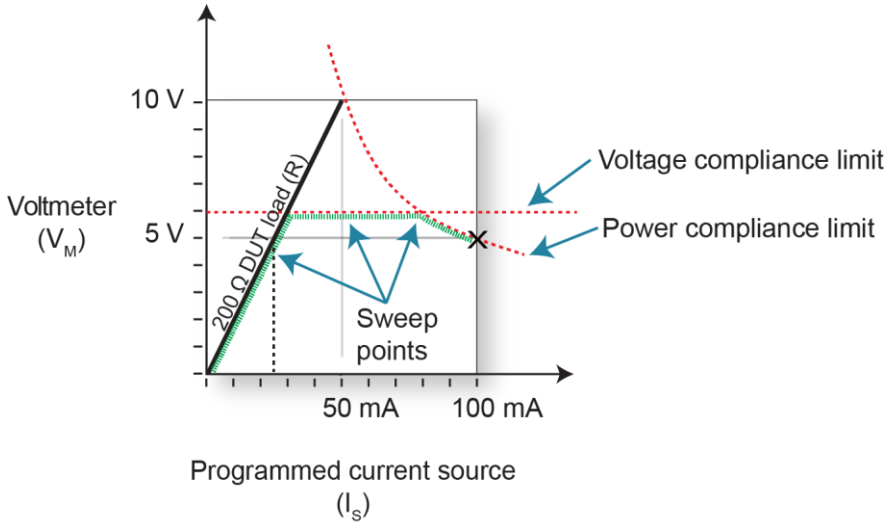
Figure 85: Current source operation in power compliance



I_S = V_M / R
= 6 V / 200 Ω)
= 30 mA

The following figure shows a current sweep on a resistive load of 200 Ω. For this example, the SMU is programmed to sweep current to 100 mA, limit voltage (6 V), and limit power (500 mW). When sweeping, the actual source output varies according to the programmed source value until the voltage limit is reached. As the figure shows, the output sources the programmed value until placed in voltage compliance at the 6 V limit. The sweep then continues (programmed current source values increase along the green sweep points line), but the output remains at the same value as when the SMU went into voltage compliance. This continues until the programmed source value sweeps to a high enough level that the power limit line is reached (in this example, 500 mW). At this point, the voltage and the current start to decrease, lowering the current and voltage values along the DUT load line. When the last point is swept (100 mA), the actual output is 25 mA (at 5 V).

Figure 86: Programmed current source sweep operation in voltage and power compliance



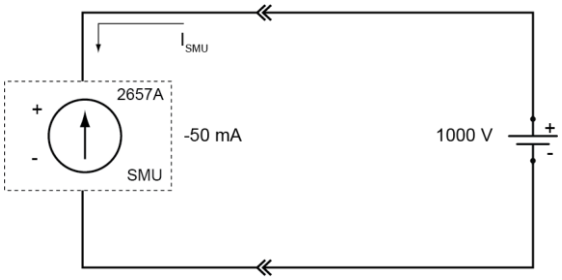
I-source sink operating boundaries

The quadrant within which the Model 2651A operates depends on the device-under-test (DUT) connected to the Model 2651A output. The following example illustrates this operation by using the Model 2651A configured to provide a constant current to discharge a 1000 V power source (a battery).

CAUTION

When using the I-Source as a sink, always set the voltage compliance limit to levels that are higher than the external voltage level. Failure to do so could result in excessive current flow into the Model 2651A (<102 mA) and incorrect measurements.

Figure 87: Sourcing current sink operation example



NOTE

The voltage compliance limit applies both to positive and negative voltages. For example, if you set the voltage compliance limit to 1500 V, the voltage limit applies to ±1500 V.

For this example, the Model 2651A is programmed to source –50 mA (the constant current) and to limit voltage to 1500 V. When the SMU turns on, it begins sinking current as determined by the programmed I-source level (-50 mA), causing a decrease in the battery voltage. If the battery were ideal and could be charged negatively, its voltage would then continue to decrease until it is negatively charged at –1500 V (shown by the green arrow in the following figure), at which point the SMU would be in voltage compliance.

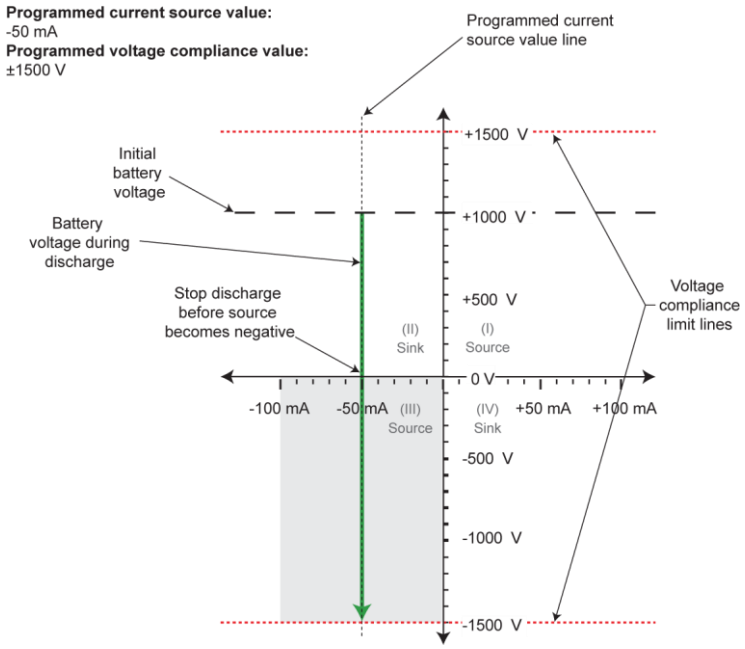
Make sure to take into account that reversing the polarity may destroy some power sources. To prevent a negative charge, monitor the SMU's measurement of the battery voltage and stop the discharge before the Model 2651A starts to operate in quadrant III (negative voltage). You can stop the discharge by changing the programmed current source level or by disconnecting the SMU from the device.

In the following figure, as the battery drains, the battery voltage is lowered as shown by the green arrow. Operation will continue in this direction until the user stops operation or the voltage reaches the voltage compliance limit line.

NOTE

Since the battery is a power source, operation in this example is limited by the capability of the battery to deliver 50 mA (see the following figure).

Figure 88: Considerations when sourcing current and sinking power



Basic circuit configurations

Source V

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

Sense circuitry continuously monitors the output voltage and makes adjustments to the V-source as needed. The V-meter senses the voltage at the HI and LO terminals (2-wire local sense) or at the device under test (DUT) (4-wire remote sense using the sense terminals) and compares it to the programmed voltage level. If the sensed level and the programmed value are not the same, the source voltage is adjusted accordingly. Remote sense eliminates the effect of voltage drops in the test leads, ensuring that the exact programmed voltage appears at the DUT.

With 4-wire sensing enabled, both remote sense leads must be connected or incorrect operation will occur. Use contact check to verify that the sense leads are connected (see Contact check measurements (on page 2-40)).

Source I

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

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

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

Source I measure I, source V measure V

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

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

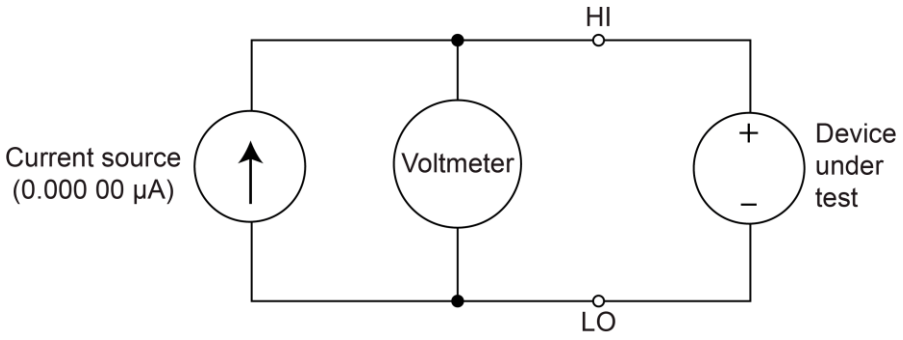
You can also use the fast analog-to-digital converter (ADC) to measure the transient behavior of the source. See Speed for more information.

Measure only (voltage or current)

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

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

Figure 89: Model 2651A measure voltage only



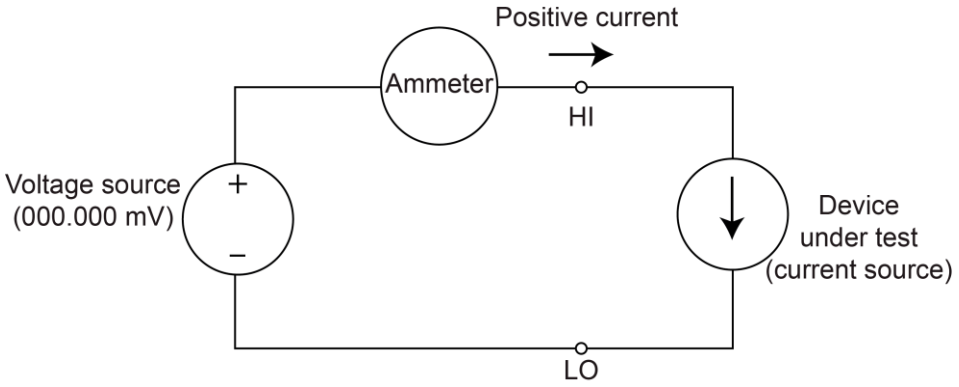
1	Current source (0.000 00 µA)
2	Voltmeter
3	HI
4	LO
5	DUT (voltage source)

CAUTION

Set the voltage limit to a level that is higher than the measured voltage. If the voltage limit is set to a level that is lower than the measured voltage, excessive current will flow into the instrument, resulting in operation as a sink. For more information, see Sink operation (on page 2-23). This current could damage the instrument. Also, when connecting an external energy source to the instrument when it is configured as a current source, set the output-off state to the high-impedance mode. See Output-off states (on page 2-65) for more information on the output-off states.

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

Figure 90: Model 2651A measure current only

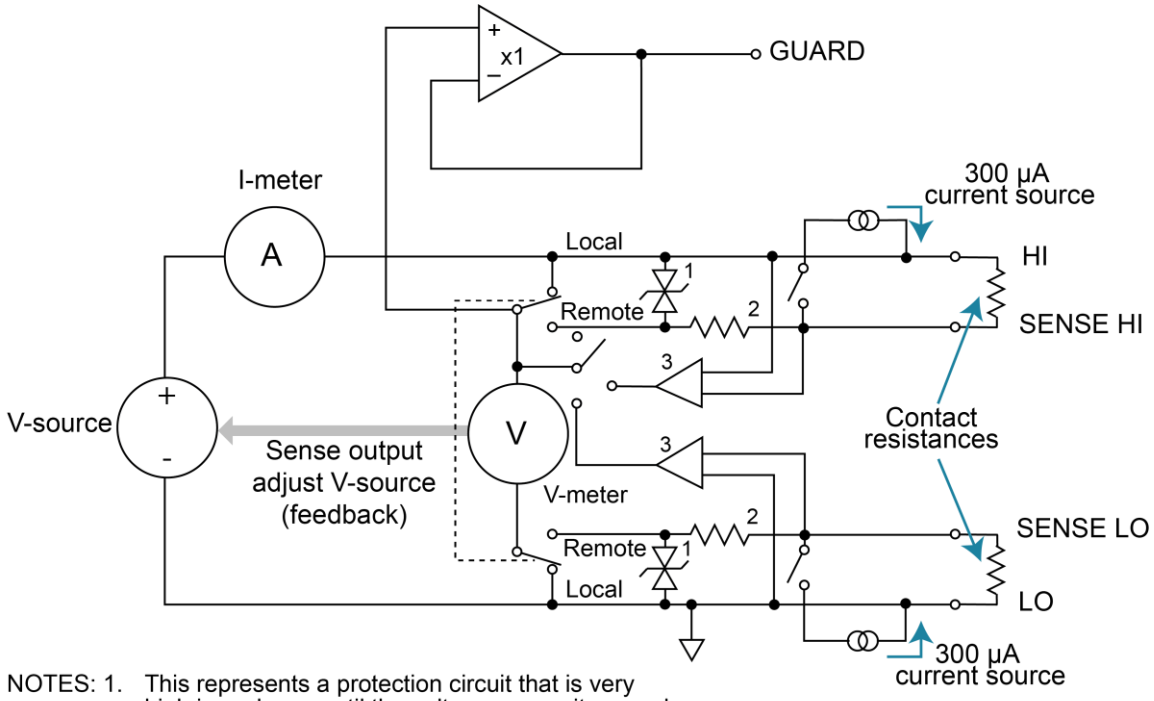


1	Voltage source (000.000 mV)
2	Ammeter
3	Positive current; positive current flowing out of HI results in positive measurements
4	HI
5	LO
6	DUT (current source)

Contact check

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

Figure 91: Contact check circuit



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

Guard

WARNING

GUARD is at the same potential as output HI. If hazardous voltages are present at output HI, they are also present at the GUARD terminal.

The rear-panel GUARD terminals are always enabled and provide a buffered voltage that is at the same level as the HI (or SENSE HI for remote sense) voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between HI and LO. In the absence of a driven guard, leakage in the external test circuit could be high enough to adversely affect the performance of the SMU.

Leakage current can occur through parasitic or nonparasitic leakage paths. An example of parasitic resistance is the leakage path across the insulator in a coaxial or triaxial cable. An example of nonparasitic resistance is the leakage path through a resistor that is connected in parallel to the device-under-test (DUT).

Guard connections

Guard is typically used to drive the guard shields of cables and test fixtures. Guard is extended to a test fixture from the cable guard shield. Inside the test fixture, the guard can be connected to a guard plate or shield that surrounds the device under test (DUT).

WARNING

To prevent injury or death, a safety shield must be used to prevent physical contact with a guard plate or guard shield that is at a hazardous potential (>30 V RMS or 42.4 V_{PEAK}). This safety shield must completely enclose the guard plate or shield and must be connected to safety earth ground. The figure in this topic shows the metal case of a test fixture being used as a safety shield.

NOTE

See Guarding and shielding (on page 2-57) for details about guarded test connections.

Inside the test fixture, a triaxial cable can be used to extend guard to the device under test (DUT). The center conductor of the cable is used for HI, and the inner shield is used for guard.

The figures below show how cable guard can eliminate leakage current through the insulators in a test fixture. In this figure, leakage current (I_L) flows through the insulators (R_{L1} and R_{L2}) to LO, adversely affecting the low-current (or high-resistance) measurement of the DUT.

Also in the figures below, the driven guard is connected to the cable shield and extended to the metal guard plate for the insulators. Since the voltage on either end of R_{L1} is the same (0 V drop), no current can flow through the leakage resistance path. Thus, the SourceMeter instrument only measures the current through the DUT.

Figure 92: Unguarded measurements

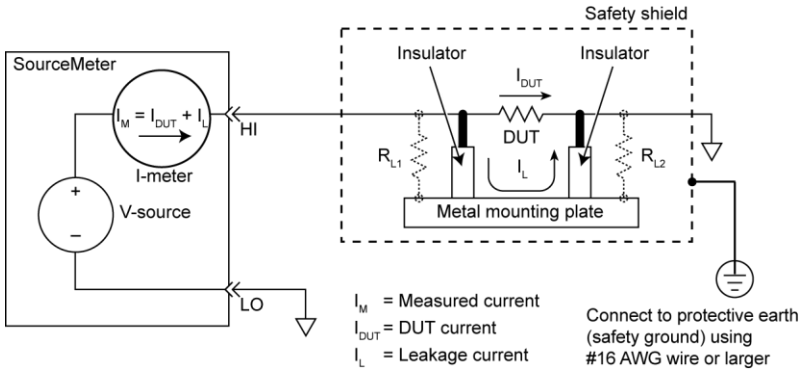
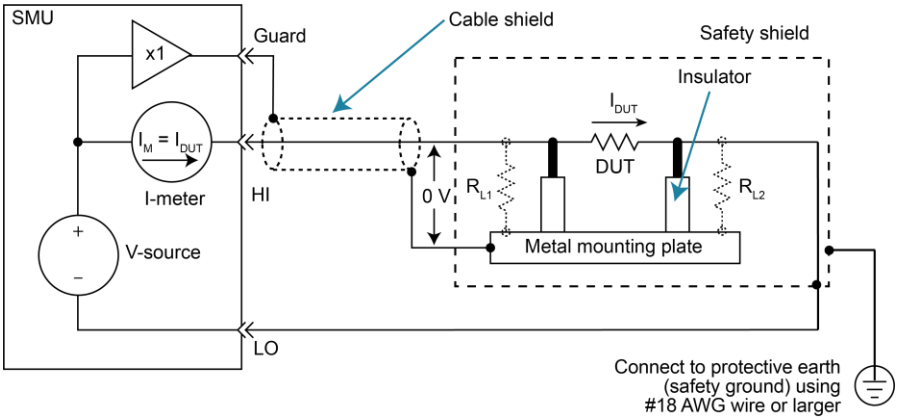


Figure 93: Guarded measurements



Cable considerations

The Model 2651A is supplied with a 1 m (3 foot) low inductance, 6 Ω cable. The supplied cable addresses several important considerations:

- Inductance
- Power dissipation
- Voltage drop at high dc current

This cable decreases the inductance as seen from the Model 2651A terminals to the load. This reduction assures specified slew rates and minimal overshoot, which is especially true when operating the Model 2651A in pulse mode at high levels of output current and voltage. The induction equation: V = Ldi/dt illustrates the need for controlling inductance. In this equation, when operating the Model 2651A in source I mode for any given inductance L, the voltage (V) represents the overhead voltage required to be applied across the inductance of the cable and load connections in order to maintain the specified di/dt. With a cable inductance of 5 µH, and a 50 A step in current in 30 µs:

V = Ldi/dt

= (5 µH)(50 A)/30 µs

= 8 V

Therefore 8 V of overhead (4 V for the Hi lead and 4 V for the Lo lead) is required in addition to the device under test (DUT) dc load voltage at 50 A. This 8 V is too high for the remote sensing circuits (3 V max/lead). In addition, the 10 V range does not have the power supply overhead to provide this 8 V (though the 40 V range does). In this example, the load/cable inductance is therefore too high and must be reduced. Using the supplied Model 2651A 6 Ω cable helps mitigate these problems since it contributes less than 200 nA/meter.

The low resistance cable supplied will dissipate only 3.6 W when operating at 20 A dc; 3.6 W is not excessive for a cable of this size.

The low resistance of the supplied cable will mitigate voltage drop due to operation at high dc currents. This characteristic will leave more overhead voltage available for the actual connection to the DUT.

Be careful to consider each issue outlined above when you make your own connections to your device.

Measurement settling time considerations

Several outside factors can influence measurement settling times. Effects such as dielectric absorption, cable leakages, and noise can all extend the times required to make stable measurements. Be sure to use appropriate shielding, guarding, and aperture selections when making low-current measurements.

Each current measurement range has a combination of a range resistor and a compensating capacitor that must settle out to allow a stable measurement. By default (when power is turned on or after a `smua.reset() command`), delays are enforced to account for approximately 6t or 6 time constants of a given range (to reach 0.1 percent of the final value, assuming 2.3t per decade). The table below lists the current ranges and associated default delays. In addition, a 1 Hz analog filter is used by default on the 1 nA and 100 pA ranges.

Current measure settling time[1, 2]

Time required to reach 0.1% of final value after source level command is processed on a fixed range. Values below for V_{OUT} = 2 V unless otherwise noted.	
Current range	**Settling time**
1.5 A to 1 A	<120 µs (typical) (Rload >6 Ω)
100 mA to 10 mA	<80 µs (typical)
1 mA	<100 µs (typical)
100 µA	<150 µs (typical)
10 µA	<500 µs (typical)
1 µA	<2.5 ms (typical)
100 nA	<15 ms (typical)
10 nA	<90 ms (typical)
1 nA[1]	<360 ms (typical)
100 pA[3]	<360 ms (typical)
1. Delay factor set to 1. Compliance equal to 100 mA. 2. Time for measurement to settle after a Vstep. 3. With default analog filter setting <450 ms.	

You can manipulate both the analog filter and the default delays to produce faster response times. Turn off the analog filter to yield faster settling times. Control the default delays by using the delay factor multiplier. The default value for delay factor multiplier is 1.0, but adjusting it to other values result in either a faster or slower response. For example, increasing the delay factor to 1.3 will account for settling to 0.01 percent of the final value. The commands to manipulate the delay factor and analog filter are shown below.

Programming example for controlling settling time delay

The following code provides measure delay examples for controlling settling time delay. You can use the delay factor to apply a multiplier when `smua.measure.delay` is set to `smua.DELAY_AUTO`. Setting the delay factor above 1.0 increases the delay; a value below 1.0 decreases the delay. Setting this value to 0.0 disables delays when autodelay is on.

```
-- Turn off measure delay (default setting is smua.DELAY_AUTO).
smua.measure.delay = 0

-- Set measure delay for all ranges to Y in seconds.
smua.measure.delay = Y

-- Adjust the delay factor.
smua.measure.delayfactor = 1.0
```

Effects of load on current source settling time

The settling time of the source-measure unit (SMU) can be influenced by the impedance of the device-under-test (DUT) in several ways. One influence is caused by an interaction between the impedances of the SMU current source feedback element and the DUT. This interaction can cause a reduction in the bandwidth of the SMU. This reduction results in an increase in the settling time of the current source.

There is a maximum DUT impedance for each current source range for which the specified current settling times are maintained. The following table lists the DUT impedances for each of these current source ranges. For the latest specifications, go to tek.com/keithley. The settling time on a current source range can increase significantly when measuring DUTs that have an impedance that is higher than the maximum DUT impedance listed below.

Maximum DUT impedances for specified settling time performance

Range	SMU feedback impedance	60 kHz ratio (DUT / SMU impedance)	Maximum DUT impedance
100 nA	33 MΩ	0.5	33 MΩ
1 µA	396 kΩ	0.5	1.2 MΩ
10 µA	396 kΩ	0.5	396 kΩ
100 µA	3687 Ω	0.5	11 kΩ
1 mA	3687 Ω	0.5	3687 kΩ
10 mA	27 Ω	0.5	81 Ω
100 mA	27 Ω	0.5	27 Ω
1 A	0.045 Ω	0.5	1.1 Ω
5 A	0.045 Ω	0.5	0.23 Ω
10 A	0.005 Ω	0.5	0.065 Ω
20 A	0.005 Ω	0.5	0.065 Ω
50 A	0.005 Ω	0.5	0.065 Ω

Creating pulses with the Model 2651A SMU

Although the Model 2651A is not a pulse generator, you can create pulses by programming the SMU to output a dc value and then return to an idle level. For information on how to create pulses, refer to Sweep operation (on page 3-21) and Using the remote trigger model (on page 3-38).

Pulse rise and fall times

NOTE

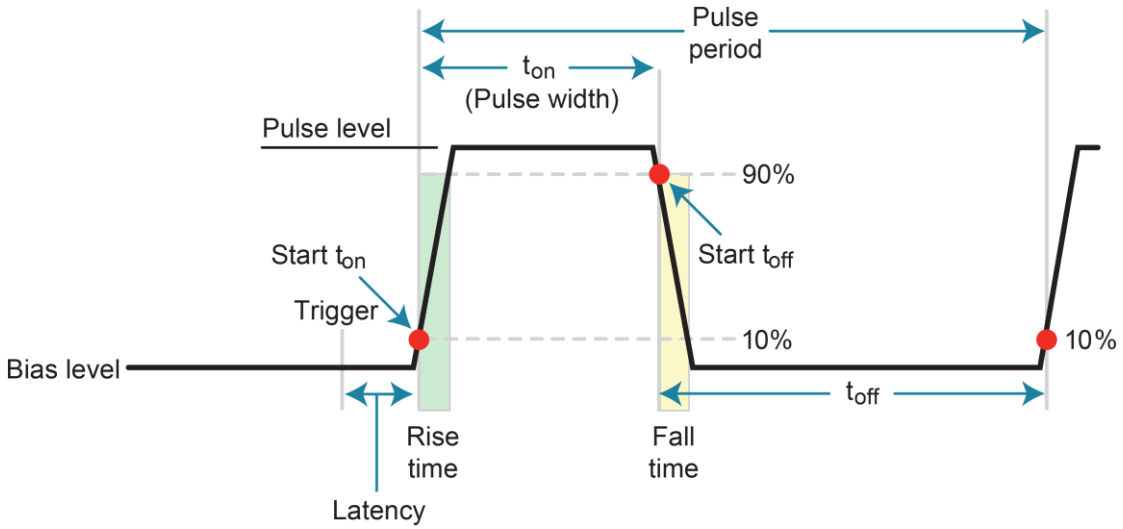
Although the Model 2651A can create pulses, it is not a pulse generator (pulse rise times are not programmable).

The pulse rise time is the time it takes a pulse to go from 10% to 90% of the maximum value of the pulse. Pulse fall time is similar but on the trailing edge of the pulse. For the Model 2651A, pulse rise and fall times can vary depending on the following factors:

- Cables (on page 4-26)
- Range and pulse settling (on page 4-30)
- Load and operating mode (on page 4-30)
- Compliance limit settings (for details, see Limit principles (on page 4-2))

Refer to the Model 2651A specifications for details on source settling times. For the latest specifications, go to tek.com/keithley.

Figure 94: Pulse rise and fall times



Cable specifications and connection configuration

Cable length, as well as capacitance and inductance in both the cabling and the test fixture, can affect pulse performance. See Cable considerations (on page 4-26) for pulse related considerations that may affect your connection scheme.

Range and pulse settling

Each SMU range has different specifications for source settling times. This causes different rise and fall time characteristics depending on the set range.

In addition, pulse performance is dependent on the pulse setting as a percent of full scale. For example, a 1 A pulse on the 10 A range (which is 10% of full scale) performs differently than a 10 A pulse on the 10 A range (which is full scale). Refer to the Model 2651A specifications for details. For the latest specifications, go to tek.com/keithley.

SMU load and operating mode

Settling times for the current source vary with the resistive load applied. In addition to the load, the times vary depending on whether the source-measure unit (SMU) is configured as a voltage source or a current source, and also if the voltage source range is selected.

Pulse width

The pulse width is the interval between 10% on the rising (leading) edge to 90% on the falling (trailing) edge. Exceeding the specified pulse width limits can result in short pulses. In addition, the jitter of the pulse width can change the pulse width (this is especially important for short pulse widths). Jitter in respect to pulse width is the short-term instability of the trailing edge relative to the leading edge.

Refer to the Model 2651A specifications for details on minimum pulse width limits, pulse width programming resolution, accuracy, and jitter. The usable pulse width is largely affected by the source settling time and measurement speed, as well as by the measure count. For the integrating analog-to-digital converter (ADC), the integration period (nplc) affects the measurement speed. For the fast ADC, the measurement interval affects the measurement speed. Review the Model 2651A specifications for information on source settling time. For the latest specifications, go to tek.com/keithley.

Section 5

Remote commands

In this section:

Introduction to TSP operation

Instruments that are enabled for Test Script Processor (TSP®) operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you do when using any other instrument.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without an external controller. You can load a series of TSP commands into the instrument. You can store these commands as a script that can be run later by sending a single command message to the instrument.

You do not have to choose between using conventional control or script control. You can combine these forms of instrument control in the way that works best for your test application.

Controlling the instrument by sending individual command messages

The simplest method of controlling an instrument through the communication interface is to send it a message that contains remote commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

TSP commands can be function-based or attribute-based. Function-based commands are commands that control actions or activities. Attribute-based commands define characteristics of an instrument feature or operation.

Constants represent fixed values.

Functions

Function-based commands control actions or activities. A function-based command performs an immediate action on the instrument.

Each function consists of a function name followed by a set of parentheses ( ). Only include information in the parentheses if the function takes a parameter. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Example 1

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

Example 2

You can use the results of a function-based command directly or assign the results to variables for later access. The following code defines `x` and prints it.

`x = math.abs(-100)` `print(x)`	Output: `100`

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, a characteristic of TSP-enabled instruments is the model number (`localnode.model`).

Attributes can be read-only, read-write, or write-only. They can be used as a parameter of a function or assigned to another variable.

To set the characteristics, attribute-based commands define a value. For many attributes, the value is in the form of a number or a predefined constant.

Example 1: Set an attribute using a number

```
beeper.enable = 0
```

This attribute controls the beeps that occur when front-panel controls are selected. Setting this attribute to `0` turns off the beeper.

Example 2: Set an attribute using a constant

```
format.data = format.REAL64
```

Using the constant `REAL64` sets the print format to double-precision floating-point format.

To read an attribute, you can use the attribute as the parameter of a function or assign it to another variable.

Example 3: Read an attribute using a function

```
print(format.data)
```

Reads the data format by passing the attribute to the print function. If the data format is set to 3, the output is:

```
3.00000e+00
```

This shows that the data format is set to double precision floating point.

Example 4: Read an attribute using a variable

```
fd = format.data
```

This reads the data format by assigning the attribute to a variable named `fd`.

Queries

Test Script Processor (TSP®) enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` and `printnumber()` commands generate output in the form of response messages. Each `print()` command creates one response message.

Example

```
x = 10
print(x)
```

Example of an output response message:

```
10
```

Note that your output may be different if you set your ASCII precision setting to a different value.

Information on scripting and programming

If you need information about using scripts with your TSP-enabled instrument, see Fundamentals of scripting for TSP (on page 6-1).

If you need information about using the Lua programming language with the instrument, see Fundamentals of programming for TSP.

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 7-7)).

Beeper control

The beeper commands allow you to enable or disable and sound the instrument beeper.

beeper.beep() (on page 7-7)
beeper.enable (on page 7-8)

Bit manipulation and logic operations

The bit functions perform bitwise logic operations on two given numbers, and bit operations on one given number. Logic and bit operations truncate the fractional part of given numbers to make them integers.

Logic operations

The `bit.bitand()`, `bit.bitor()`, and `bit.bitxor()` functions in this group perform bitwise logic operations on two numbers. The Test Script Processor (TSP®) scripting engine performs the indicated logic operation on the binary equivalents of the two integers. This bitwise logic operation is performed on all corresponding bits of the two numbers. The result of a logic operation is returned as an integer.

Bit operations

The rest of the functions in this group are used for operations on the bits of a given number. You can use these functions to:

- Clear a bit
- Toggle a bit
- Test a bit
- Set a bit or bit field
- Retrieve the weighted value of a bit or field value

All these functions use an index parameter to specify the bit position of the given number. The least significant bit of a given number has an index of 1, and the most significant bit has an index of 32.

NOTE

The Test Script Processor (TSP) scripting engine stores all numbers internally as IEEE Std 754 double-precision floating-point values. The logical operations work on 32-bit integers. Any fractional bits are truncated. For numbers larger than 4294967295, only the lower 32 bits are used.

bit.bitand() (on page 7-8)
bit.bitor() (on page 7-9)
bit.bitxor() (on page 7-10)
bit.clear() (on page 7-11)
bit.get() (on page 7-12)
bit.getfield() (on page 7-13)
bit.set() (on page 7-14)
bit.setfield() (on page 7-15)
bit.test() (on page 7-16)
bit.toggle() (on page 7-17)

Data queue

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link® network at any time

The data queue in the Test Script Processor (TSP®) scripting engine is first-in, first-out (FIFO).

You can access data from the data queue even if a remote group or a node has overlapped operations in process.

dataqueue.add() (on page 7-51)
dataqueue.CAPACITY (on page 7-52)
dataqueue.clear() (on page 7-53)
dataqueue.count (on page 7-54)
dataqueue.next() (on page 7-55)

Digital I/O

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

The I/O port has 14 lines. Each line can be at TTL logic state 1 (high) or 0 (low). See the pinout diagram in Digital I/O port (on page 3-92) for additional information.

There are commands to read and write to each individual bit, and commands to read and write to the entire port.

Display

Error queue

When errors and events occur, the error and status messages are placed in the error queue. Use the error queue commands to request error and status message information.

errorqueue.clear() (on page 7-98)
errorqueue.count (on page 7-98)
errorqueue.next() (on page 7-99)

Event log

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

eventlog.all() (on page 7-100)
eventlog.clear() (on page 7-101)
eventlog.count (on page 7-101)
eventlog.enable (on page 7-102)
eventlog.next() (on page 7-102)
eventlog.overwritemethod (on page 7-103)

File I/O

You can use the file I/O commands to open and close directories and files, write data, or to read a file on an installed USB flash drive. File I/O commands are organized into two groups:

- Commands that reside in the `fs` and `io` table, for example: `io.open()`, `io.close()`, `io.input()`, and `io.output()`. Use these commands to manage file system directories; open and close file descriptors; and perform basic I/O operations on a pair of default files (one input and one output).
- Commands that reside in the file descriptors (for example: *`fileVar`*`:seek()`, *`fileVar`*`:write()`, and *`fileVar`*`:read()`) operate exclusively on the file with which they are associated.

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

```
"/usb1/"
```

NOTE

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

For basic information about navigation and directory listing of files on a flash drive, see File system navigation (on page 2-70).

File descriptor commands for file I/O use a colon (`:`) to separate the command parts rather than a period (`.`), like the `io` commands.

File descriptors cannot be passed between nodes in a TSP-Link® system, so the `io.open()`, `fileVar::read()`, and `fileVar::write` commands are not accessible to the TSP-Link system. However, the default input and output files mentioned above allow for the execution of many file I/O operations without any reference to a file descriptor.

fileVar:close() (on page 7-104)
fileVar:flush() (on page 7-105)
fileVar:read() (on page 7-107)
fileVar:seek() (on page 7-108)
fileVar:write() (on page 7-110)
fs.chdir() (on page 7-114)
fs.cwd() (on page 7-115)
fs.is_dir() (on page 7-116)
fs.is_file() (on page 7-117)
fs.mkdir() (on page 7-118)
fs.readdir() (on page 7-119)
fs.rmdir() (on page 7-120)
io.close() (on page 7-129)
io.flush() (on page 7-130)
io.input() (on page 7-131)
io.open() (on page 7-132)
io.output() (on page 7-133)
io.read() (on page 7-134)
io.type() (on page 7-135)
io.write() (on page 7-136)
os.remove() (on page 7-188)
os.rename() (on page 7-189)

The following standard I/O commands are not supported:

File	I/O
■ *fileVar*:lines() ■ *fileVar*:setvbuf()	■ io.lines() ■ io.popen()

GPIB

This attribute stores the GPIB address.

gpib.address (on page 7-124)

Instrument identification

These commands store strings that describe the instrument.

LAN and LXI

The LAN commands have options that allow you to review and configure network settings.

The `lan.config.*` commands allow you to configure LAN settings over the remote interface.

NOTE

You must send `lan.applysettings()` for the configuration settings to take effect.

The `lan.status.*` commands help you determine the status of the LAN.

The `lan.trigger[N].*` commands allow you to set up and assert trigger events that are sent over the LAN.

Other LAN commands allow you to reset the LAN, restore defaults, check LXI domain information, and enable or disable the Nagle algorithm.

Miscellaneous

Parallel script execution

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.

Reading buffer

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

Reset

Resets settings to their default settings.

RS-232

Saved setups

Use the saved setups commands to save or restore the configurations to or from the nonvolatile memory of the instrument or an installed USB flash drive. You can use the `setup.poweron` attribute to specify which setup is recalled when the instrument is turned on.

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 6-1).

SMU

SMU calibration

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.

Time

Triggering

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

TSP-Link

These functions and attributes allow you to set up and work with a system that is connected by a TSP-Link® network.

TSP-Net

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

Userstrings

Use the functions in this group to store and retrieve user-defined strings in nonvolatile memory. These strings are stored as key-value pairs.

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

Factory scripts

The Keithley Instruments Model 2651A High Power System SourceMeter® Instrument is shipped with one or more factory scripts saved in its flash firmware memory. A factory script is made up of a number of functions. Some of them can be called from the front-panel LOAD TEST menu. All of them can be called using remote programming.

A factory script is similar to a user script, except a factory script is created by Keithley Instruments at the factory and is permanently stored in nonvolatile memory. The differences between a user script and a factory script include the following:

- A factory script cannot be deleted from nonvolatile memory.
- The script listing for a factory script can be retrieved and modified, but it is then treated as a user script. A user script cannot be saved as a factory script.
- Factory scripts are not stored in global variables. The only references to factory scripts are in the `script.factory.scripts` attribute.
- The `script.factory.catalog()` function returns an iterator that can be used in a `for` loop to iterate over all the factory scripts.

Example

To retrieve the catalog listing for factory scripts, send:

```
for name in script.factory.catalog() do print(name) end
```

Running a factory script

Use either of the following commands to run a factory script:

```
script.factory.scripts.name()
script.factory.scripts.name.run()
```

Where: *`name`* is the name of the factory script.

Example:

Run the factory script named `KISweep`.

```
script.factory.scripts.KISweep()
```

Running a factory script function from the front-panel controls

1. Press the **LOAD** key.
2. Select **FACTORY**.
3. Select the function to run and press the **ENTER** key or navigation wheel.
4. Press the **RUN** key.
5. Follow the prompts on the front panel to run the script.

Retrieving and modifying a factory script listing

The script listing for a factory script can be retrieved and modified. However, it cannot be saved as a factory script. The modified script can be saved as a user script using the same name or a new name.

An imported factory script can only be reloaded into the Model 2651A as a user script.

The following function retrieves a script listing. The script code is output with the shell keywords (`loadscript` or `loadandrunscript` and `endscript`):

```
script.factory.scripts.name.list()
```

Where: *`name`* is the name of the factory script.

An example that retrieves the script listing for a factory script named `KISweep`:

```
script.factory.scripts.KISweep.list()
```

KISweep factory script

The KISweep factory script provides simple sweep test programming and shows how to use the sweeping function.

This script is made up of the following functions. Access these functions from the front panel or the remote interfaces. The following functions make up the KISweep factory script:

SweepILinMeasureV() (on page 7-394)
SweepIListMeasureV() (on page 7-395)
SweepILogMeasureV() (on page 7-396)
SweepVLinMeasureI() (on page 7-398)
SweepVListMeasureI() (on page 7-399)
SweepVLogMeasureI() (on page 7-400)

KIPulse factory script

The KIPulse factory script provides examples of how to generate pulses and to provide a simple pulsing interface. Pulses can be generated using the functions listed below.

NOTE

Please note the following information about the KIPulse factory script:

- This factory script only operates on the channels present in the instrument executing the pulse functions. These functions do not operate correctly if you attempt to access instrument channels over the TSP-Link® interface.
- The KIPulse factory scripts are general purpose examples that may not be suitable for all use cases. Very short pulses (less than 1 ms pulse width) may require optimization of the examples provided by the factory script in order to achieve settled measurements.
- The `PulseIMeasureV()` and `PulseVMeasureI()` functions may be accessed from the front panel. The remaining functions may only be accessed remotely.

Use the configuration KIPulse tag parameter pulse functions (on page 5-24) to configure a pulse train and assign the configuration to the *tag* parameter (use `QueryPulseConfig()` to inspect configured pulse trains). Use the initiation `InitiatePulseTest()` function to execute the pulse trains assigned to its *tag* arguments. The conditions listed in the table below must be true for these functions to execute successfully.

Conditions that must be true for successful function execution:

Config functions:
Source autorange (I and V) off
Measure autorange (I and V) off
Measure NPLC < ton
Measure autozero OFF or ONCE

InitiatePulseTest functions:
Output on
There is enough free space in the buffer
Buffer append mode is on when pulse train is >1 point

Use the KIPulse simple pulse functions (on page 5-24) to specify and perform a specified number of pulse-measure cycles.

KIPulse tag parameter pulse functions

KIPulse simple pulse functions

Advanced features for KIPulse tag parameter pulse functions

Variable off time between pulses in a pulse train

The KIPulse "Configure" functions accept the *toff* parameter as a table or as a number. The table allows you to define different off times to be used after each pulse. Note the following:

- If *toff* is passed as a number or only a single value is used in the table, it is used for all points in a multiple point pulse.
- The number of times specified in the table must match the number of points called for in the sweep.
- The times used in tables must match for dual channel pulsing.
- Each specified off time must adhere to the duty cycle limits for the specified pulsing region.

Simultaneous IV measurement during pulse

The KIPulse "Configure" functions optionally accept an extra reading buffer to activate simultaneous IV measurements during pulsing. Previous usage of passing in a reading buffer or a nil (for no measurement) is still supported.

KIHighC factory script

The KIHighC factory script is made up of two functions: `i_leakage_measure()` and `i_leakage_threshold()`. These functions are intended to be used when HighC mode is active. Output is generally at a nonzero voltage before calling these functions. These functions can also be used to step the voltage to zero volts in order to measure the leakage current.

KIParlib factory script

The KIParlib factory script is made up of two functions: `gm_vsweep()` and `gm_isweep()`.

gm_vsweep() (on page 7-123)
gm_isweep() (on page 7-122)

KISavebuffer factory script

The KISavebuffer script has one function: `savebuffer()`.

savebuffer() (on page 7-198)

Section 6

Instrument programming

In this section:

Fundamentals of scripting for TSP

NOTE

Though it can improve your process to use scripts, you do not have to create scripts to use the instrument. Most of the examples in the documentation can be run by sending individual command messages. The next few sections of the documentation describe scripting and programming features of the instrument. You only need to review this information if you are using scripting and programming.

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument more efficiently.

Scripts offer several advantages compared to sending individual commands from the host controller (computer):

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs more quickly and efficiently when it processes scripts than it does when it processes individual commands.
- You can incorporate features such as looping and branching into scripts.
- Scripts allow the controller to perform other tasks while the instrument is running a script, enabling some parallel operation.
- Scripts eliminate repeated data transfer times from the controller.

In the instrument, the Test Script Processor (TSP®) scripting engine processes and runs scripts.

This section describes how to create, load, modify, and run scripts.

What is a script?

A script is a collection of instrument control commands and programming statements. Scripts that you create are referred to as **user scripts**.

Your scripts can be interactive. Interactive scripts display messages on the front panel of the instrument that prompt the operator to enter parameters.

Runtime and nonvolatile memory storage of scripts

Scripts are loaded into the runtime environment of the instrument. From there, they can be stored in nonvolatile memory in the instrument.

The runtime environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to store a value while the instrument is turned on. When you create a script, the instrument creates a global variable with the same name so that you can reference the script more conveniently. After scripts are loaded into the runtime environment, you can run and manage them from the front panel of the instrument or from a computer. Information in the runtime environment is lost when the instrument is turned off.

Nonvolatile memory is where information is stored even when the instrument is turned off. Save scripts to nonvolatile memory to save them even if the power is cycled. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

Scripts are placed in the runtime environment when:

- The instrument is turned on. All scripts that are saved to nonvolatile memory are copied to the runtime environment when the instrument is turned on.
- Loaded over a remote command interface.

For detail on the amount of memory available in the runtime environment, see Memory considerations for the runtime environment (on page 6-53).

NOTE

If you make changes to a script in the runtime environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See Working with scripts in nonvolatile memory (on page 6-11).

What can be included in scripts?

Scripts can include combinations of Test Script Processor (TSP®) commands and Lua code. TSP commands instruct the instrument to do one thing and are described in the command reference (see TSP commands (on page 7-7)). Lua is a scripting language that is described in Fundamentals of programming for TSP.

Commands that cannot be used in scripts

Though the instrument accepts the following commands, you cannot use these commands in scripts.

Commands that cannot be used in scripts

General commands	IEEE Std 488.2 common commands	
abort	*CLS	*RST
endflash	*ESE	*SRE
endscript	*ESE?	*SRE?
flash	*ESR?	*STB?
loadscript	*IDN?	*TRG
loadandrunscript	*OPC	*TST?
password	*OPC?	*WAI

Manage scripts

This section describes how to create scripts by sending commands over the remote interface.

Tools for managing scripts

To manage scripts, you can send messages to the instrument, use your own development tool or program, use Keithley Instruments Test Script Builder (TSB) software, or use TSB Embedded on the web interface of the instrument.

TSB software is a programming tool that you can download from the Product Support and Downloads web page (tek.com/product-support). You can use it to create, modify, debug, and store Test Script Processor (TSP®) scripting engine scripts. For more information about using the TSB software, see Using Test Script Builder (TSB) (on page 6-35).

TSB Embedded is a tool with a reduced set of features than the complete Keithley TSB software. TSB Embedded has both script-building functionality and console functionality (single-line commands). It is accessed from a web browser. Refer to Working with TSB Embedded (on page 6-39) for additional information.

NOTE

If you are using TSB or TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

Create and load a script

You create scripts by loading them into the runtime environment of the instrument. You can load a script as a named script or as the anonymous script.

Once a script is loaded into the instrument, you can execute it remotely or from the front panel.

Anonymous scripts

If a script is created with the `loadscript` or `loadandrunscript` command with no name defined, it is called the anonymous script. There can only be one anonymous script in the runtime environment. If another anonymous script is loaded into the runtime environment, it replaces the existing anonymous script.

Named scripts

A named script is a script with a unique name. You can have as many named scripts as needed in the instrument (within the limits of the memory available to the runtime environment). When a named script is loaded into the runtime environment with the `loadscript` or `loadandrunscript` commands, a global variable with the same name is created to reference the script.

Key points regarding named scripts:

- If you load a new script with the same name as an existing script, the existing script becomes an unnamed script, which in effect removes the existing script if there are no variables that reference it.
- Sending revised scripts with different names does not remove previously loaded scripts.
- Named scripts can be saved to internal nonvolatile memory. Saving a named script to nonvolatile memory allows the instrument to be turned off without losing the script. See Working with scripts in nonvolatile memory (on page 6-11).

Load a script by sending commands over the remote interface

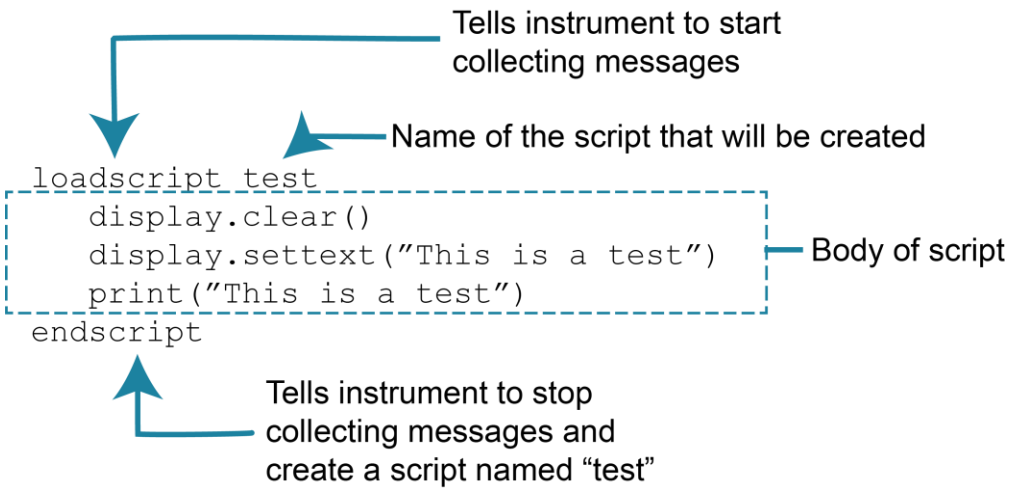
To load a script over the remote interface, you can use the `loadscript`, `loadandrunscript`, and `endscript` commands.

The `loadscript` and `loadandrunscript` commands start the collection of messages that make up the script. When the instrument receives either of these commands, it starts collecting all subsequent messages. Without these commands, the instrument runs them immediately as individual commands.

The `endscript` command tells the instrument to compile the collection of messages. It compiles the messages into one group of commands. This group of commands is loaded into the runtime environment.

The following figure shows an example of how to load a script named “test.” The first command tells the instrument to start collecting the messages for the script named “test.” The last command marks the end of the script. When this script is run, the message `This is a test` is displayed on the instrument and sent to the computer.

Figure 95: Loadscript and endscript example



To load a named script by sending commands:

1. Send the command `loadscript` *`scriptName`*, where *`scriptName`* is the name of the script. The name must be a legal Lua variable name.
2. Send the commands that need to be included in the script.
3. Send the command `endscript`.
4. You can now run the script. See Run scripts (on page 6-7).

NOTE

To run the script immediately, use `loadandrunscript` *`scriptName`* instead of `loadscript`.

Load a script from the instrument front panel

You can also load scripts from a USB flash drive to the runtime environment of the instrument. Depending on the content of the TSP file on the drive, the script can be loaded either as an anonymous script without a designated name, or as a named script with a user-defined name. Only named scripts can be saved to internal nonvolatile memory. Only one anonymous script can exist in the runtime environment.

To load a script into the instrument with a specific name, the TSP file must include the shell keywords `loadscript` and endscript, along with the specified script name, as shown in the example file `MyScript1.tsp`, which contains the script:

```
loadscript Beeper
reset()
beeper.enable = beeper.ON
beeper.beep(2, 2400)
endscript
```

When you load the file `MyScript1.tsp` from the flash drive, a script named `Beeper` is created in the runtime environment. Note that the script is named using the name that follows the `loadscript` keyword, not the name of the TSP file on the flash drive. After the script is loaded, you can choose to save it to nonvolatile memory.

If the loaded file does not contain `loadscript` and `endscript` keywords, or if no name is included after the loadscript keyword, the code is loaded as the anonymous script. Loading an unnamed script overwrites the existing anonymous script. For example, if a file named `MyScript2.tsp` contains only the following code, the script is loaded as the anonymous script:

```
reset()
beeper.enable = beeper.ON
beeper.beep(2, 2400)
```

The file must be a valid script file. If not, an error message is posted and no further action is taken. You can view the errors on the front panel of the instrument.

To load a script from a USB flash drive:

1. Insert the flash drive into the USB port on the instrument.
2. Select the **MENU** key.
3. Select the **SCRIPT** option.
4. Select the **LOAD** option.
5. Select the **USB1** option. A menu is displayed that lists the TSP files and directories on the flash drive.
6. If the files are in a directory, use the navigation wheel to select the directory. A new menu is displayed that lists the TSP files and directories in that directory.
7. Use the navigation wheel to select the TSP file you want to load.

8. If the script has the same name as a script that is already in memory, you are prompted to overwrite the script.
   - Select **Yes** to continue.
   - Select **No** to return to the list of files. You must select a file to continue.
9. The SCRIPT ACTION menu is displayed. You can select:
   - **SAVE-INTERNAL:** Save the file to nonvolatile memory. This is the same as sending `scriptVar.save()` with no parameters.
   - **ACTIVE-FOR-RUN:** Set the script to run from the RUN button.
10. Loading is complete. To return to the MAIN menu, press **EXIT (LOCAL)** until the MAIN menu is displayed.
11. If you selected **ACTIVE-FOR-RUN**, you can select **RUN** to run the script.

NOTE

The entries in the SCRIPT ACTION menu depend on whether the script that was loaded is a named script or the anonymous script. If it is a named script, both SAVE-INTERNAL and ACTIVE-FOR-RUN appear in the menu. If it is the anonymous script, then only ACTIVE-FOR-RUN appears in the menu.

Create a script using TSB Embedded

NOTE

If you are using TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

You can create a script from the instrument web interface with TSB Embedded. When you save the script, it is loaded into the runtime environment and saved in the nonvolatile memory of the instrument.

To create a script using TSB Embedded:

1. In the **TSP Script** box, enter a name for the script.
2. In the input area, enter the sequence of commands to be included in the script.
3. Click **Save Script**. The name is added to the User Scripts list on the left.

Run scripts

This section describes how to run the anonymous and named scripts.

On the front panel, items are available through the USER menu if you explicitly add them to the menu. The items the menu selections represent can be scripts, function calls, or instrument commands. Items in the menus are referred to as scripts in this section.

The SCRIPTS menu lists the names of scripts in nonvolatile memory or scripts that have been added to the runtime environment. The anonymous script also appears in this menu.

NOTE

If the instrument is in local control when the script is started, it switches to remote control (`REM` is displayed) while the script is running. The instrument is returned to local control when the script completes. If you press the front-panel **EXIT (LOCAL)** key while the script is running, the script is stopped.

Run the anonymous script

The anonymous script can be run many times without reloading it. It remains in the runtime environment until a new anonymous script is created or until the instrument is turned off.

To run the anonymous script, use any one of these commands:

- `run()`
- `script.run()`
- `script.anonymous()`
- `script.anonymous.run()`

Run a named script

You can run any named script that is in the runtime environment using one of the following commands:

- *`scriptVar`*`()`
- *`scriptVar`*`.run()`

Where *`scriptVar`* is the user-defined name of the script.

When a script is named, it can be accessed using the global variable *`scriptVar`*.

To run a named script from TSB Embedded, select the script from the User Scripts list and click Run.

Example: Run a named script

`test3()`	If the script `test3` is loaded into the runtime environment, the instrument executes `test3`.

Run a user script from the instrument front panel

From the front panel, you can load and run a script that was previously added to the USER menu.

To run the code from the front panel and add it to the USER menu:

1. Select the **LOAD** key.
2. Select **USER**.
3. Select the script from list and press the **ENTER** key. The script is loaded into the runtime environment.

NOTE

If you are used to using `print` in Test Script Builder, note that the output of the print commands using this procedure do not function the same as when you are in Test Script Builder. You may find that it makes more sense to use Test Script Builder to get the output you need.

4. Press the **RUN** key to execute.

To run a script directly without adding it to the USER menu:

1. Select the **LOAD** key.
2. Select **SCRIPTS** and select the **ENTER** key. There may be a short pause before a menu is displayed that represents the scripts in the instrument.
3. Select the script from the list and select the **ENTER** key. The script is now loaded for front-panel execution.
4. Press the **RUN** key to execute.

NOTE

If you are used to using `print` in Test Script Builder, note that the output of the prints using this procedure do not function the same as when you are in Test Script Builder. You may find that it makes more sense to use Test Script Builder to get the output you need.

Scripts that run automatically

You can set up scripts to run automatically when you power on the instrument. To do this, either set the `autorun` attribute for the script to `yes` (see Autorun scripts (on page 6-10)) or create a script with the script name `autoexec` (see Autoexec script (on page 6-10)).

Autorun scripts

Autorun scripts run automatically when the instrument is turned on. You can set any number of scripts to autorun. The run order for autorun scripts is arbitrary, so make sure the run order is not important.

As shown in the example below, you can set a script to run automatically by setting the `.autorun` attribute of the script to `"yes"` and then saving the script.

Example:

```
scriptVar.autorun = "yes"
scriptVar.save()
```

Where: *`scriptVar`* is the user-defined name of the script.

To disable autorun, set the `autorun` attribute of the script to `no` and then save the script.

NOTE

The *`scriptVar`*`.save()` command saves the script to nonvolatile memory, which makes the change persistent through a power cycle. See Save a user script to nonvolatile memory (on page 6-11) for more detail.

Example: Set a script to run automatically

```
test5.autorun = "yes"
test5.save()
```

Assume a script named `test5` is in the runtime environment.
The next time the instrument is turned on, `test5` script automatically loads and runs.

Autoexec script

The autoexec script runs automatically when the instrument is turned on. It runs after all the scripts have loaded and any scripts defined as autorun have run.

To create a script that executes automatically, create and load a new script and name it `autoexec`. See Create and load a script (on page 6-4).

NOTE

You must save the `autoexec` script to nonvolatile memory if you want to use it after instrument power has been turned off and then turned on again. See Save a user script to nonvolatile memory (on page 6-11) for more detail.

Example: Creating an autoexec script with the loadscript command

```
loadscript autoexec
display.clear()
display.settext("Hello from autoexec")
endscript
autoexec.save()
```

Creates the script `autoexec`.
Saves the `autoexec` script to nonvolatile memory. The next time the instrument is turned on, `Hello from autoexec` is displayed.

Example: Creating an autoexec script using TSB Embedded

```
display.clear()
display.settext("Hello from autoexec")
```

In the TSP Script box, enter `autoexec`.
Enter the code in the entry box.
Click **Save Script**.
Creates a new script that clears the display when the instrument is turned on and displays `Hello from autoexec`.

Working with scripts in nonvolatile memory

The Fundamentals of scripting for TSP (on page 6-1) section in this manual describes working with scripts, primarily in the runtime environment. You can also work with scripts in nonvolatile memory.

The runtime environment and nonvolatile memory are separate storage areas in the instrument. The information in the runtime environment is lost when the instrument is turned off. The nonvolatile memory remains intact when the instrument is turned off. When the instrument is turned on, information in nonvolatile memory is loaded into the runtime environment.

Save a user script

You can save scripts to nonvolatile memory using commands or TSB Embedded.

Only named scripts can be saved to nonvolatile memory. The anonymous script must be named before it can be saved to nonvolatile memory.

NOTE

If a script is not saved to nonvolatile memory, the script is lost when the instrument is turned off.

To save a script to nonvolatile memory from a remote interface:

1. Create and load a named script (see Create and load a script (on page 6-4)).
2. Send the command *`scriptVar`*`.save()`, where *`scriptVar`* is the name of the script.

To save a script to nonvolatile memory using TSB Embedded:

Select **Save Script**.

Example: Save a user script to nonvolatile memory

```
test1.save()
```

Assume a script named `test1` has been loaded. `test1` is saved into nonvolatile memory.

To save a script to an external USB flash drive using a remote interface:

NOTE

When you save a script to a USB flash drive, you do not need to specify a file extension. The extension `.tsp` is automatically added. If you do specify a file extension, it must be `.tsp`. An error occurs if you use any other file extension.

1. Load a script (see Create and load a script (on page 6-4)).
2. Send the command *`scriptVar`*`.save("/usb1/`*`filename`*`.tsp")`, where *`scriptVar`* is the variable referencing the script and *`filename`*`.tsp` is the name of the file.

To save a script to an external USB flash drive or other accessible drive using TSB Embedded:

Load the script and select **Export to PC**.

Save the anonymous script as a named script

To save the anonymous script to nonvolatile memory, you must name it first.

To save the anonymous script as a named script:

1. To name the script, send the command `script.anonymous.name = "`*`myTest`*`"` (where *`myTest`* is the name of the script).
2. Send the `script.anonymous.save()` command to save *`myTest`* to nonvolatile memory.

Save a script from the instrument front panel

You can save scripts from the runtime environment to nonvolatile memory on the instrument front panel.

NOTE

If you want to save the anonymous script to nonvolatile memory, you must name it first. See Save the anonymous script as a named script (on page 6-12).

To save a script to nonvolatile memory from the front panel:

1. Select the **MENU** key.
2. Select the **SCRIPT** option.
3. Select the **SAVE** option.

   A list of the scripts available to save is displayed. It may take a few seconds to display. The displayed list is from the `script.user.scripts` table in the instrument.
4. Turn the navigation wheel to select the script that you want to save.
5. Select **INTERNAL**. Press the navigation wheel. The script is saved to nonvolatile memory using the name attribute of the script.
6. Press **EXIT (LOCAL)** several times to return to the Main Menu.

Delete user scripts

NOTE

These steps remove a script from nonvolatile memory. To completely remove a script from the instrument, there are additional steps you must take. See Delete user scripts from the instrument (on page 6-52).

To delete a script from nonvolatile memory using a remote interface:

You can delete the script from nonvolatile memory by sending either of the following commands:

- `script.delete("name")`
- `script.user.delete("name")`

Where: *`name`* is the user-defined name of the script.

Example: Delete a user script from nonvolatile memory

`script.delete("test8")`	Delete a user script named `test8` from nonvolatile memory.

To delete a script from nonvolatile memory using TSB Embedded:

1. In TSB Embedded, select the script from the **User Scripts** list.
2. Click **Delete**. There is no confirmation message.

Programming example: Interactive script

An interactive script prompts the operator to input values using the instrument front panel. The following example script uses display messages to prompt the operator to:

- Enter the digital I/O line on which to output a trigger
- Enter the output trigger pulsewidth

After the output trigger occurs, the front display displays a message to the operator.

When an input prompt is displayed, the script waits until the operator inputs the parameter or presses the **ENTER** key.

The example shown here assumes that you are using TSB. If you are using a remote interface, you need to add the `loadscript` and `endscript` commands to the example code. See Load a script by sending commands over the remote interface (on page 6-4) for details.

```
-- Clear the display.
display.clear()

-- Prompt user for digital I/O line on which to output trigger.
myDigioLine = display.menu("Select digio line", "1 2 3 4 5 6 7 8 9")

-- Convert user input to a number.
intMyDigioLine = tonumber(myDigioLine)

-- Prompt user for digital output trigger mode.
myDigioEdge = display.menu("Select digio mode", "Rising Falling")
if myDigioEdge == "Rising" then
   edgeMode = digio.TRIG_RISING
else
   edgeMode = digio.TRIG_FALLING
end

-- Prompt user for output trigger pulsewidth.
myPulseWidth = display.prompt(
    "000.0", "us", "Enter trigger pulsewidth", 10, 10, 100)

-- Scale the entered pulsewidth.
myPulseWidth = myPulseWidth * 1e-6

-- Generate the pulse.
digio.trigger[intMyDigioLine].mode = edgeMode
digio.trigger[intMyDigioLine].pulsewidth = myPulseWidth
digio.trigger[intMyDigioLine].assert()

-- Alert the user through the display that the
-- output trigger has occurred.
display.setcursor(1, 1)
display.settext("Trigger asserted $Non digital I/O line " .. intMyDigioLine)

-- Wait five seconds and then return to main screen.
delay(5)
display.screen = display.SMUA
```

Fundamentals of programming for TSP

To conduct a test, a computer (controller) is programmed to send sequences of commands to an instrument. The controller orchestrates the actions of the instrumentation. The controller is typically programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

To use the advanced features of the instrument, you can add programming commands to your scripts. Programming commands control script execution and provide tools such as variables, functions, branching, and loop control.

The Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands.

What is Lua?

Lua is a programming language that can be used with TSP-enabled instruments. Lua is an efficient language with simple syntax that is easy to learn.

Lua is also a scripting language, which means that scripts are compiled and run when they are sent to the instrument. You do not compile them before sending them to the instrument.

Lua basics

This section contains the basics about the Lua programming language to allow you to start adding Lua programming commands to your scripts quickly.

For more information about Lua, see the Lua website (lua.org). Another source of useful information is the Lua users group (lua-users.org), created for and by users of Lua programming language.

Comments

You can start a comment anywhere outside a string by typing a double hyphen (`--`). If the text immediately after `--` is anything other than double left brackets (`[[`), the comment is a short comment, which continues until the end of the line.

If `--` is followed by `[[`, the following characters are a long comment, which continues until double right brackets (`]]`) close the comment. Long comments may continue for several lines and may contain nested `[[ ... ]]` pairs. The example below shows how to use code comments.

An example of a short comment is:

```
-- Turn off the front-panel display.
```

An example of a long comment is:

```
--[[Display a menu with three menu items. If the second menu item is selected,
the selection is given the value Test2.]]
```

Function and variable name restrictions

You cannot use factory script names, functions created by factory scripts, Lua reserved words and top-level command names for function or variable names.

For information on factory script names, see Factory scripts (on page 5-21).

You cannot use the following Lua reserved words for function or variable names.

Lua reserved words

and	for	or
break	function	repeat
do	if	return
else	in	then
elseif	local	true
end	nil	until
false	not	while

You also cannot use top-level command names as variable names. If you use these names, it results in the loss of use of the commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until you turn the instrument power off and then turn it on again. These names include:

Top level command names			
beeper	gcinfo	os	status
bit	gettimezone	print	string
collectgarbage	gpib	printbuffer	timer
dataqueue	io	printnumber	tonumber
delay	lan	reset	tostring
digio	localnode	savebuffer	trigger
display	makegetter	script	tsplink
errorqueue	makesetter	serial	tspnet
eventlog	math	settime	type
exit	meminfo	settimezone	userstring
format	node	setup	waitcomplete
fs	opc	smua	

Values and variable types

In Lua, you use variables to store values in the runtime environment for later use.

Lua is a dynamically-typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been assigned a value.

Variable types

Variables can be one of the following types.

Variable types and values

Variable type returned	Value	Notes
`"nil"`	not declared	The type of the value `nil`, whose main property is to be different from any other value; usually it represents the absence of a useful value.
`"boolean"`	true or false	Boolean is the type of the values `false` and `true`. In Lua, both `nil` and `false` make a condition `false`; any other value makes it `true`.
`"number"`	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers.
`"string"`	sequence of words or characters	
`"function"`	a block of code	Functions perform a task or compute and return values.
`"table"`	an array	New tables are created with `{ }` braces. For example: `{1, 2, 3.00e0}`
`"userdata"`	variables	Allows arbitrary program data to be stored in Lua variables.
`"thread"`	line of execution	

To determine the type of a variable, you can call the `type()` function, as shown in the examples below.

NOTE

The output you get from these examples may vary depending on the data format that is set.

Example: Nil

```
x = nil
print(x, type(x))
```

```
nil
```

Example: Boolean

```
y = false
print(y, type(y))
```

```
false      boolean
```

Example: String and number

```
x = "123"
print(x, type(x))

x = x + 7
print(x, type(x))
```

```
123      string


Adding a number to x forces its type to number.
130     number
```

Example: Function

```
function add_two(first_value,
   second_value)
   return first_value + second_value
end
print(add_two(3, 4), type(add_two))
```

```
7     function
```

Example: Table

```
atable = {1, 2, 3, 4}
print(atable, type(atable))
print(atable[1])
print(atable[4])
```

Defines a table with four numeric elements.
Note that the *table* value (shown here as a096cd30) varies.

```
table: a096cd30      table
1
4
```

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the runtime environment.

Functions

With Lua, you can group commands and statements using the `function` keyword. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple values.

You can use functions to form expressions that calculate and return a value. Functions can also act as statements that execute specific tasks.

Functions are first-class values in Lua. That means that functions can be stored in variables, passed as arguments to other functions, and returned as results. They can also be stored in tables.

Note that when a function is defined, it is stored in the runtime environment. Like all data that is stored in the runtime environment, the function persists until it is removed from the runtime environment, is overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or in Lua code in either of the following forms:

```
function myFunction(parameterX) functionBody end
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function.
- *parameterX*: Parameter names. To use multiple parameters, separate the names with commas.
- *functionBody*: The code that is executed when the function is called.

To execute a function, substitute appropriate values for *parameterX* and insert them into a message formatted as:

```
myFunction(valueForParameterX, valueForParameterY)
```

Where *valueForParameterX* and *valueForParameterY* represent the values to be passed to the function call for the given parameters.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example 1

```
function add_two(first_value, second_value)
   return first_value + second_value
end
print(add_two(3, 4))
```

Creates a variable named `add_two` that has a variable type of function.
Output:
`7`

Example 2

```
add_three = function(first_value,
   second_value, third_value)
   return first_value + second_value +
      third_value
end
print(add_three(3, 4, 5))
```

Creates a variable named `add_three` that has a variable type of function.
Output:
`12`

Example 3

`function sum_diff_ratio(first_value,`
`  second_value)`
`  psum = first_value + second_value`
`  pdif = first_value - second_value`
`  prat = first_value / second_value`
`  return psum, pdif, prat`
`end`
`sum, diff, ratio = sum_diff_ratio(2, 3)`
`print(sum)`
`print(diff)`
`print(ratio)`
	Returns multiple parameters (sum, difference, and ratio of the two numbers passed to it).
	Output:
	`5`
	`-1`
	`0.66666666666667`

Create functions using scripts

You can use scripts to define functions. Scripts that define a function are like any other script: They do not cause any action to be performed on the instrument until they are executed. The global variable of the function does not exist until the script that created the function is executed.

A script can consist of one or more functions. Once a script has been run, the computer can call functions that are in the script directly.

NOTE

The following steps use TSB Embedded. You can also use the `loadscript` and `endscript` commands to create the script over the remote interface. See Load a script by sending commands over the remote interface (on page 6-4).

Steps to create a function using a script:

1. In TSB Embedded, enter a name into the TSP Script box. For example, type `MakeMyFunction`.
2. Enter the function as the body of the script. This example concatenates two strings:

   ```
   MyFunction = function (who)
       print ("Hello".. who)
   end
   ```

3. Select **Save Script**.
4. `MakeMyFunction` is now on the instrument in a global variable with the same name as the script (`MakeMyFunction`). However, the function defined in the script does not yet exist because the script has not been executed.
5. Run the script as a function. For this example, send:

   ```
   MakeMyFunction()
   ```

   This instructs the instrument to run the script, which creates the `MyFunction` global variable. This variable is of the type "`function`" (see Variable types (on page 6-17)).

6. Run the new function with a value.

   ```
   MyFunction("world")
   ```

   The response message is:

   ```
   Hello world.
   ```

Group commands using the function keyword

The following script contains instrument commands that display the name of the person that is using the script on the front panel of the instrument. It takes one parameter to represent this name. When this script is run, the function is loaded in memory. Once loaded into memory, you can call the function outside of the script to execute it.

When calling the function, you must specify a string for the *name* argument of the function. For example, to set the name to **John**, call the function as follows:

```
myDisplay("John")
```

Example: User script

User script created in Test Script Builder or TSB Embedded	User script created in a different program

`function myDisplay(name)`
`  display.clear()`
`  display.settext(`
`  name .. "$N is here!")`
`end`
	`loadscript`
	`function myDisplay(name)`
	`  display.clear()`
	`  display.settext(`
	`  name .. " $N is here!")`
	`end`
	`endscript`

Operators

You can compare and manipulate Lua variables and constants using operators.

Arithmetic operators

Operator	Description
+	addition
−	subtraction
*	multiplication
/	division
−	negation (for example, c = −a)
^	exponentiation

Relational operators

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~=	not equal
==	equal

Logical operators

The logical operators in Lua are `and`, `or`, and `not`. All logical operators consider both `false` and `nil` as false and anything else as true.

The operator `not` always returns `false` or `true`.

The conjunction operator `and` returns its first argument if the first argument is `false` or `nil`; otherwise, `and` returns its second argument. The disjunction operator `or` returns its first argument if this value is different from `nil` and `false`; otherwise, `or` returns its second argument. Both `and` and `or` use shortcut evaluation, that is, the second operand is evaluated only if necessary.

NOTE

The example output you get may vary depending on the data format settings of the instrument.

Example

```
print(10 or errorqueue.next())          1.00000e+01
print(nil or "a")                       a
print(nil and 10)                       nil
print(false and errorqueue.next())      false
print(false and nil)                    false
print(false or nil)                     nil
print(10 and 20)                        2.00000e+01
```

String concatenation

String operators

Operator	Description
..	Concatenates two strings. If either argument is a number, it is coerced to a string (in a reasonable format) before concatenation.

Example: Concatenation

```
print(2 .. 3)
print("Hello " .. "World")
```

Output:

```
23
Hello World
```

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority):

- `^` (exponentiation)
- `not, -` (unary)
- `*, /`
- `+, –`
- `..` (concatenation)
- `<, >, <=, >=, ~=, !=, ==`
- `and`
- `or`

You can use parentheses to change the precedences in an expression. The concatenation ("`..`") and exponentiation ("`^`") operators are right associative. All other binary operators are left associative. The examples below show equivalent expressions.

Equivalent expressions

`reading + offset < testValue/2+0.5`	=	`(reading + offset) < ((testValue/2)+0.5)`
`3+reading^2*4`	=	`3+((reading^2)*4)`
`Rdg < maxRdg and lastRdg <= expectedRdg`	=	`(Rdg < maxRdg) and (lastRdg <= expectedRdg)`
`-reading^2`	=	`-(reading^2)`
`reading^testAdjustment^2`	=	`reading^(testAdjustment^2)`

Conditional branching

Lua uses the `if`, `else`, `elseif`, `then`, and `end` keywords to do conditional branching.

Note that in Lua, `nil` and `false` are `false` and everything else is `true`. Zero (0) is `true` in Lua.

The syntax of a conditional block is as follows:

```
if expression then
   block
elseif expression then
   block
else
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

Example: If

```
if 0 then
   print("Zero is true!")
else
   print("Zero is false.")
end
```

Output:
```
Zero is true!
```

Example: Comparison

```
x = 1
y = 2
if x and y then
   print("Both x and y are true")
end
```

Output:
```
Both x and y are true
```

Example: If and else

```
x = 2
if not x then
   print("This is from the if block")
else
   print("This is from the else block")
end
```

Output:
```
This is from the else block
```

Example: Else and elseif

```
x = 1
y = 2
if x and y then
   print("'if' expression 2 was not false.")
end

if x or y then
   print("'if' expression 3 was not false.")
end

if not x then
   print("'if' expression 4 was not false.")
else
   print("'if' expression 4 was false.")
end

if x == 10 then
   print("x = 10")
elseif y > 2 then
   print("y > 2")
else
   print("x is not equal to 10, and y is not greater than 2.")
end
```

Output:

```
'if' expression 2 was not false.
'if' expression 3 was not false.
'if' expression 4 was false.
x is not equal to 10, and y is not greater than 2.
```

Loop control

If you need to repeat code execution, you can use the Lua `while`, `repeat`, and `for` control structures. To exit a loop, you can use the `break` keyword.

While loops

To use conditional expressions to determine whether to execute or end a loop, you use `while` loops. These loops are similar to Conditional branching (on page 6-24) statements.

```
while expression do
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: While

```
list = {
   "One", "Two", "Three", "Four", "Five", "Six"}
print("Count list elements on numeric index:")
element = 1
while list[element] do
   print(element, list[element])
   element = element + 1
end
```

This loop exits when `list[element] = nil`.

Output:

```
Count list elements on
   numeric index:
1    One
2    Two
3    Three
4    Four
5    Five
6    Six
```

Repeat until loops

To repeat a command, you use the `repeat` ... `until` statement. The body of a `repeat` statement always executes at least once. It stops repeating when the conditions of the `until` clause are met.

```
repeat
   block
until expression
```

Where:

- *`block`* consists of one or more Lua statements
- *`expression`* is Lua code that evaluates to either `true` or `false`

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Repeat until

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
print("Count elements in list using repeat:")
element = 1
repeat
   print(element, list[element])
   element = element + 1
until not list[element]
```

Output:

```
Count elements in list
   using repeat:
1  One
2  Two
3  Three
4  Four
5  Five
6  Six
```

For loops

There are two variations of `for` statements supported in Lua: Numeric and generic.

NOTE

In a `for` loop, the loop expressions are evaluated once, before the loop starts.

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Numeric for

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
---------- For loop -----------
print("Counting from one to three:")
for element = 1, 3 do
   print(element, list[element])
end
print("Counting from one to four, in steps of two:")
for element = 1, 4, 2 do
   print(element, list[element])
end
```

The numeric `for` loop repeats a block of code while a control variable runs through an arithmetic progression.

Output:

```
Counting from one to three:
1   One
2   Two
3   Three
Counting from one to four, in steps of two:
1   One
3   Three
```

Example: Generic for

```
days = {"Sunday",
   "Monday",     "Tuesday",
   "Wednesday",  "Thursday",
   "Friday",     "Saturday"}

for i, v in ipairs(days) do
   print(days[i], i, v)
end
```

The generic `for` statement works by using functions called iterators. On each iteration, the iterator function is called to produce a new value, stopping when this new value is nil.

Output:

```
Sunday      1    Sunday
Monday      2    Monday
Tuesday     3    Tuesday
Wednesday   4    Wednesday
Thursday    5    Thursday
Friday      6    Friday
Saturday    7    Saturday
```

Break

The `break` statement terminates the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A break ends the innermost enclosing loop.

Return and break statements can only be written as the last statement of a block. If it is necessary to return or break in the middle of a block, an explicit inner block can be used.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Break with while statement

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a `break` value (`breakValue`) so that the `break` statement is used to exit the `while` loop before the value of k reaches `0`.

Output:

```
Going to break and k = 3
```

Example: Break with while statement enclosed by comment delimiters

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
-- local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a break value (`breakValue`), but the break value line is preceded by comment delimiters so that the break value is not assigned, and the code reaches the value `0` to exit the `while` loop.

Output:

```
Break value not found
```

Example: Break with infinite loop

```
a, b = 0, 1
while true do
   print(a, b)
   a, b = b, a + b
   if a > 500 then
      break
   end
end
```

This example uses a `break` statement that causes the while loop to exit if the value of `a` becomes greater than 500.

Output:

```
0      1
1      1
1      2
2      3
3      5
5      8
8      13
13     21
21     34
34     55
55     89
89     144
144    233
233    377
377    610
```

Tables and arrays

Lua makes extensive use of the data type table, which is a flexible array-like data type. Table indices start with 1. Tables can be indexed not only with numbers, but with any value except `nil`. Tables can be heterogeneous, which means that they can contain values of all types except `nil`.

Tables are the sole data structuring mechanism in Lua. They may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, and so on. To represent records, Lua uses the field `name` as an index. The language supports this representation by providing `a.name` as an easier way to express `a["name"]`.

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Loop array

```
atable = {1, 2, 3, 4}
i = 1
while atable[i] do
   print(atable[i])
   i = i + 1
end
```

Defines a table with four numeric elements.
Loops through the array and prints each element.
The Boolean value of `atable[index]` evaluates to `true` if there is an element at that index. If there is no element at that index, `nil` is returned (`nil` is considered to be `false`).

Output:

```
1
2
3
4
```

Standard libraries

In addition to the standard programming constructs described in this document, Lua includes standard libraries that contain useful functions for string manipulation, mathematics, and related functions. Test Script Processor (TSP®) scripting engine instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP scripting engine. These libraries are automatically loaded when the TSP scripting engine starts and do not need to be managed by the programmer.

The following topics provide information on some of the basic Lua standard libraries. For additional information, see the Lua website (lua.org).

NOTE

When referring to the Lua website, please be aware that the TSP scripting engine uses Lua 5.0.2.

Base library functions

Base library functions

Function	Description
`collectgarbage()` `collectgarbage(`*`limit`*`)`	Sets the garbage-collection threshold to the given limit (in kilobytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, Lua immediately runs the garbage collector. If there is no limit parameter, it defaults to zero (0), which forces a garbage-collection cycle. See Lua memory management (on page 6-31) for more information.
`gcinfo()`	Returns the number of kilobytes of dynamic memory that the Test Script Processor (TSP®) scripting engine is using and returns the present garbage collector threshold (also in kilobytes). See Lua memory management (on page 6-31) for more information.
`tonumber(`*`x`*`)` `tonumber(`*`x, base`*`)`	Returns *x* converted to a number. If *x* is already a number, or a convertible string, the number is returned; otherwise, it returns `nil`. An optional argument specifies the base to use when interpreting the numeral. The base may be any integer from 2 to 36, inclusive. In bases above 10, the letter `A` (in either upper or lower case) represents 10, `B` represents 11, and so forth, with `Z` representing 35. In base 10, the default, the number may have a decimal part and an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(`*`x`*`)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(`*`v`*`)`	Returns (as a string) the type of its only argument. The possible results of this function are `"nil"` (a string, not the value `nil`), `"number"`, `"string"`, `"boolean"`, `"table"`, `"function"`, `"thread"`, and `"userdata"`.

Lua memory management

Lua automatically manages memory, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua occasionally runs a garbage collector to collect all objects that are no longer accessible from Lua. All objects in Lua are subject to automatic management, including tables, variables, functions, threads, and strings.

Lua uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory Lua is using; the other is a threshold. When the number of bytes crosses the threshold, Lua runs the garbage collector, which reclaims the memory of all inaccessible objects. The byte counter is adjusted, and the threshold is reset to twice the new value of the byte counter.

String library functions

This library provides generic functions for string manipulation, such as finding and extracting substrings. When indexing a string in Lua, the first character is at position 1 (not 0, as in ANSI C). Indices may be negative and are interpreted as indexing backward from the end of the string. Thus, the last character is at position –1, and so on.

String library functions

Function	Description

`string.byte(s)`
`string.byte(s, i)`
`string.byte(s, i, j)`
	Returns the internal numeric codes of the characters `s[i], s[i+1], ···, s[j]`. The default value for `i` is `1`; the default value for `j` is `i`.

`string.char(···)`	Receives zero or more integers separated by commas. Returns a string with length equal to the number of arguments, in which each character has the internal numeric code equal to its corresponding argument.

`string.format(`
`  formatstring, ...)`
	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the modifiers `*`, `l`, `L`, `n`, `p`, and `h` are not supported and there is an extra option, `q`. The `q` option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written.
	For example, the call:
	`string.format('%q', 'a string with "quotes" and \n new line')`
	produces the string:
	`"a string with \"quotes\" and \`
	`  new line"`
	The options `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `X`, and `x` all expect a number as argument. `q` and `s` expect a string. This function does not accept string values containing embedded zeros, except as arguments to the `q` option.

`string.len(s)`	Receives a string and returns its length. The empty string `""` has length `0`. Embedded zeros are counted, so `"a\000bc\000"` has length `5`.
`string.lower(s)`	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
`string.rep(s, n)`	Returns a string that is the concatenation of `n` copies of the string `s`.
`string.sub(s, i)` `string.sub(s, i, j)`	Returns the substring of `s` that starts at `i` and continues until `j`; `i` and `j` can be negative. If `j` is absent, it is assumed to be equal to `-1` (which is the same as the string length). In particular, the call `string.sub(s, 1, j)` returns a prefix of `s` with length `j`, and `string.sub(s, -i)` returns a suffix of `s` with length `i`.
`string.upper(s)`	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions

Function	Description
`math.abs(x)`	Returns the absolute value of *x*.
`math.acos(x)`	Returns the arc cosine of *x*.
`math.asin(x)`	Returns the arc sine of *x*.
`math.atan(x)`	Returns the arc tangent of *x*.
`math.atan2(y, x)`	Returns the arc tangent of *y*/*x* but uses the signs of both parameters to find the quadrant of the result (it also correctly handles the case of *x* being zero).
`math.ceil(x)`	Returns the smallest integer larger than or equal to *x*.
`math.cos(x)`	Returns the cosine of *x*.
`math.deg(x)`	Returns the angle *x* (given in radians) in degrees.
`math.exp(x)`	Returns the value e^x.
`math.floor(x)`	Returns the largest integer smaller than or equal to *x*.
`math.frexp(x)`	Returns `m` and `e` such that $x = m2^e$, where `e` is an integer and the absolute value of `m` is in the range `[0.5, 1]` (or zero when *x* is zero).
`math.ldexp(m, e)`	Returns $m2^e$ (`e` should be an integer).
`math.log(x)`	Returns the natural logarithm of *x*.
`math.log10(x)`	Returns the base-10 logarithm of *x*.
`math.max(x, ...)`	Returns the maximum value among its arguments.
`math.min(x, ...)`	Returns the minimum value among its arguments.
`math.pi`	The value of π (3.141592654).
`math.pow(x, y)`	Returns x^y (you can also use the expression `x^y` to compute this value).
`math.rad(x)`	Returns the angle *x* (given in degrees) in radians.

`math.random()`
`math.random(m)`
`math.random(m, n)`
	This function is an interface to the simple pseudorandom generator function `rand` provided by ANSI C.
	When called without arguments, returns a uniform pseudorandom real number in the range `[0,1]`. When called with an integer number *m*, `math.random()` returns a uniform pseudorandom integer in the range `[1, m]`. When called with two integer numbers *m* and *n*, `math.random()` returns a uniform pseudorandom integer in the range `[m, n]`.

`math.randomseed(x)`	Sets *x* as the seed for the pseudorandom generator; equal seeds produce equal sequences of numbers.
`math.sin(x)`	Returns the sine of *x*.
`math.sqrt(x)`	Returns the square root of *x*. You can also use the expression `x^0.5` to compute this value.
`math.tan(x)`	Returns the tangent of *x*.

Script with a for loop

The following script puts a message on the front panel display slowly — one character at a time. The intent of this example is to demonstrate:

- The use of a `for` loop
- Simple display remote commands
- Simple Lua string manipulation

NOTE

When creating a script using the TSB Embedded, you do not need the shell commands `loadscript` and `endscript`, as shown in the examples below.

Example: User script

User script created in TSB Embedded	User script created in user's own program

`display.clear()`
`myMessage = "Hello World!"`
`for k = 1, string.len(myMessage) do`
`  x = string.sub(myMessage, k, k)`
`  display.settext(x)`
`  print(x)`
`  delay(1)`
`end`
	`loadscript`
	
	`display.clear()`
	`myMessage = "Hello World!"`
	`for k = 1, string.len(myMessage) do`
	`  x = string.sub(myMessage, k, k)`
	`  display.settext(x)`
	`  print(x)`
	`  delay(1)`
	`end`
	`endscript`

Test Script Builder

Keithley Instruments Test Script Builder (TSB) is a software tool you can use to develop scripts for TSP-enabled instruments.

Installing the TSB software

The installation files for the TSB software are available at tek.com/keithley.

To install the TSB software:

1. Close all programs.
2. Download the installer to your computer and double-click the `.exe` file to start the installation.
3. Follow the on-screen instructions.

Using Test Script Builder (TSB)

Keithley Instruments Test Script Builder (TSB) is a software tool that simplifies building test scripts. You can use TSB to perform the following operations:

- Send remote commands and Lua statements
- Receive responses (data) from commands and scripts
- Upgrade instrument firmware
- Create, manage, and run user scripts
- Debug scripts
- Import factory scripts to view or edit and convert to user scripts

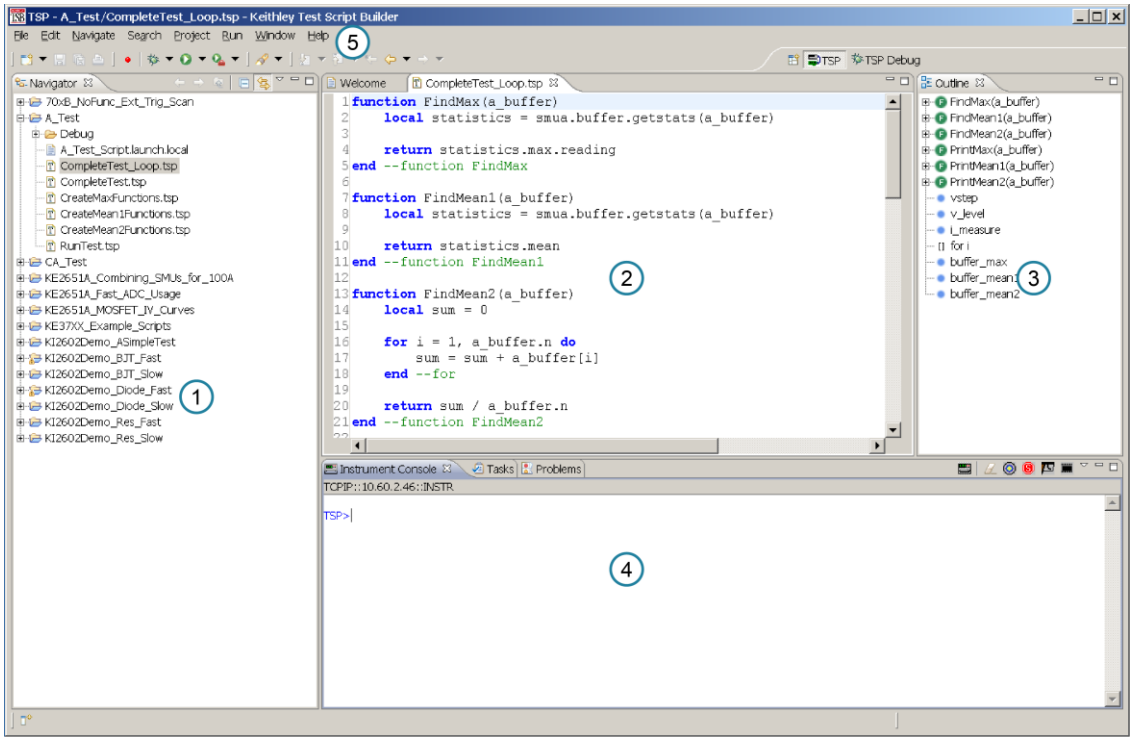
The Keithley Instruments Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands. For more information about using the Lua scripting language with Keithley TSP-enabled instruments, refer to the Fundamentals of programming for TSP.

Keithley has created a collection of remote commands specifically for use with Keithley TSP-enabled instruments; for detailed information about those commands, refer to the "Command reference" section of the documentation for your specific instrument. You can build scripts from a combination of these commands and Lua programming statements. Scripts that you create are referred to as "user scripts." Also, some TSP-enabled instruments include built-in factory scripts.

The following figure shows an example of the Test Script Builder. As shown, the workspace is divided into these areas:

- Project navigator
- Script editor
- Outline view
- Programming interaction
- Help files

Figure 96: Example of the Test Script Builder workspace



Item	Description
1	Project navigator
2	Script editor; right-click to run the script that is displayed
3	Outline view
4	Programming interaction
5	Help; includes detailed information on using Test Script Builder

Project navigator

The project navigator consists of project folders and the script files (.tsp) created for each project. Each project folder can have one or more script files.

To view the script files in a project folder, select the plus (+) symbol next to the project folder. To hide the folder contents, select the minus (−) symbol next to the project folder.

You can download a TSP project to the instrument and run it, or you can run it from the TSB interface.

Script editor

The script editor is where you write, modify, and debug scripts.

To open and display a script file, double-click the file name in the project navigator. You can have multiple script files open in the script editor at the same time. Each open script file is displayed on a separate tab.

To display another script file that is already open, select the tab that contains the script in the script editor area.

Outline view

The outline view allows you to navigate through the structure of the active script in the script editor. Double-clicking a variable name or icon causes the first instance of the variable in the active script to be highlighted.

This view shows:

- Names of local and global variables
- Functions referenced by the active script in the script editor
- Parameters
- Loop control variables
- Table variables
- Simple assignments to table fields

The Outline tab is visible by default in the TSP perspective.

Icon	Name	Examples
	Global function variable	`function gFunction()` `end`
	Local function variable	`local function lFunction()` `end`
	Anonymous function	`myTest(function() return 1 end)`
	Global table variable	`gTable = { }`
	Local table variable	`local lTable = { }`
	Other table field	`testTable.unit1 = "This is unit 1"` `testTable.unit2 = "This is unit 2"`
	Global variable	`gVariable = 3`
	Local variable	`local lVariable = 5`

	Table method	`gTable = { }`
		`function gTable:testmethod()`
		`end`

[ ]	Nonfunction block statement (example 1)	`if true == true then`
		`  local var`
		`end`

	Nonfunction block statement (example 2)	`for index = 1, 10 do` `end`

Programming interaction

This part of the workspace is where you interact with the scripts that you are building in Test Script Builder (TSB). The actual contents of the programming interaction area of the workspace can vary.

You can send commands from the Instrument Console command line, retrieve data, view variables and errors, and view and set breakpoints when using the debug feature. For additional information, refer to the online help that is accessible from Test Script Builder (TSB).

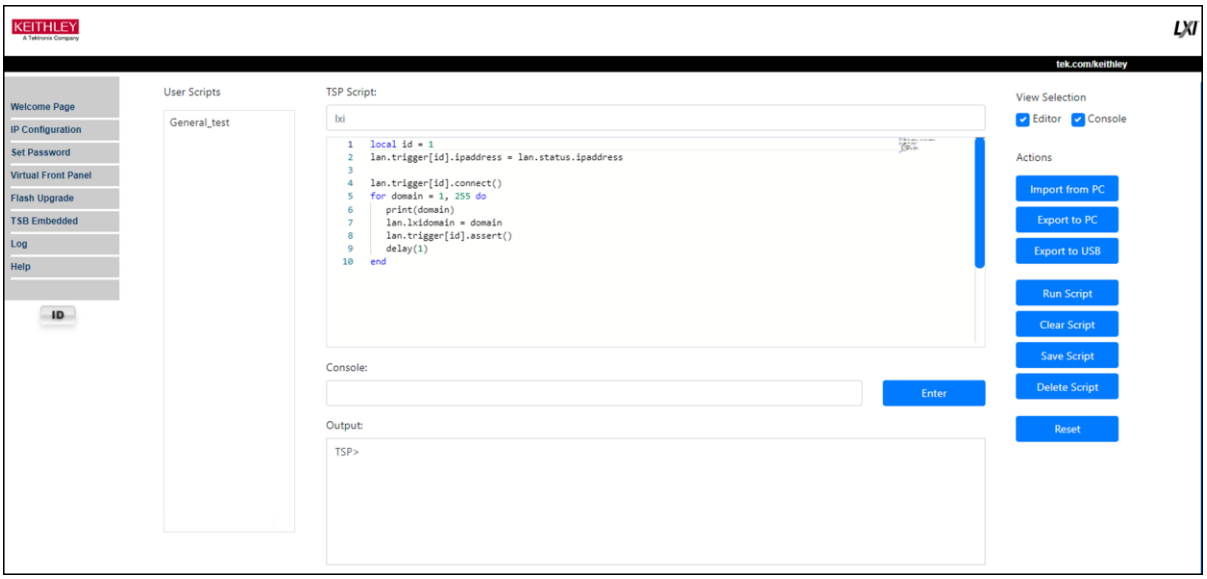
Working with TSB Embedded

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

NOTE

For more information on scripts, refer to Fundamentals of scripting for TSP (on page 6-1).

Figure 97: TSB Embedded interface



Send individual instrument commands with TSB Embedded

You can send individual commands to the instrument using TSB Embedded. The response from the instrument appears in the Output box.

To send commands from the console:

1. Type the command in **Console**.
2. Press the **Enter** key to send the command to the instrument. The command is displayed in the Output box. If there is a response to the command, it is displayed after the command.

To clear information from the Output box:

1. Right-click in the **Output** box.
2. Select **Clear**.

To copy information from the Output box:

1. Right-click in the **Output** box.
2. Select **Copy**. The information is copied to the clipboard.

The following procedure creates a script and saves it to nonvolatile memory.

To create a new script:

1. Click in the script editor area and then type the first line of your script.
2. Press the **Enter** key to advance to line 2.
3. In the TSP Script field, type the name of the script and then click **Save Script**.
   The instrument validates the syntax and then saves the script to the nonvolatile memory.

To remove the code from the script editor:

Click the **Clear** button that is above the editor.

Run a script

Running a script executes the script on the instrument.

To run a script:

1. Select a script from the User Scripts list.
2. Click **Run Script**.

To stop a running script, select **Abort**.

The Abort Script button is only displayed while a script is running.

Delete a script

NOTE

You cannot retrieve a deleted script. Be sure to back up your script to your computer before deleting.

To delete a script from TSB Embedded:

1. Select the script from the User Scripts list.
2. Select **Delete Script**.
3. Select **Delete** on the confirmation message.

Modify a script

You can modify the script in TSB Embedded.

To modify a script:

1. Select a script from the User Scripts list.
2. Modify the code in the editor.
3. Select **Save Script**.

Export a script to a computer

You can download a script from TSB Embedded to the host computer.

TSP scripts have the extension `.tsp`.

To export a script to a computer:

1. Select the script from the User Scripts list.
2. Select **Export to PC**. The file is saved as a download.
3. Use the procedure for your browser to work with file.

Password management

The Model 2651A has password capabilities that let you decide how to password protect the instrument. Password protection prevents unauthorized access to any remote interface and reserves the instrument exclusively for your use.

When password usage is enabled, you must supply a password to change the configuration or to control an instrument from a remote command interface.

Setting the password from a command or web interface

The attribute `localnode.passwordmode` enables passwords and sets the mode. The password mode identifies which interface to password protect.

Set this attribute to one of the values below to enable password checking:

- `localnode.PASSWORD_NONE` or `0`: Disable passwords everywhere
- `localnode.PASSWORD_WEB` or `1`: Use passwords on the web interface only
- `localnode.PASSWORD_LAN` or `2`: Use passwords on the web interface and all LAN interfaces
- `localnode.PASSWORD_ALL` or `3`: Use passwords on the web interface and all remote command interfaces

NOTE

When a password is set for the web interface, you cannot make changes using the web interface options Reading Buffers, Flash Upgrade, or TSB Embedded.

The password lock feature on Model 2651A is similar to the lock feature on your computer.

NOTE

You must assign a password to use this feature. Passwords can be up to 255 characters.

To set the password using the web interface:

1. From the web interface, click **Set Password**.
   The LXI - Keithley Instruments - 2651A - Administration page is displayed.
2. In **Current Password**, type the existing password. The default is `admin`.
3. In **New Password**, type the new password.
4. Retype the new password in **Confirm New Password**.
5. Click **Submit**.

The LXI Welcome page is displayed.

To enable the password from a command interface:

To lock the instrument when you are away from the testing area, send the following command:

```
password
```

The remote interface is locked. The Model 2651A does not respond to commands issued from the command interface until you unlock the interface. This reserves the instrument and protects the test script running on the instrument.

Unlocking the remote interface

If the remote interface is locked, you must enter the password before the Model 2651A responds to any command issued over a remote interface.

NOTE

The password for the example below is `Keithley`.

To unlock the remote interface, send the following command:

```
password Keithley
```

The Model 2651A is unlocked and communicates with any remote interface.

Resetting the password

If you forget the password, you can reset the password from the front panel. Once you enable the password feature, the Model 2651A stores this password until the LAN configuration is reset or until you reset the password.

To reset the password:

1. From the front panel, press the **MENU** key.
2. Select **RESET-PASSWORD**.

NOTE

Resetting the LAN settings also resets the password feature. If you reset the LAN settings, you must re-enable the password feature.

Advanced scripting for TSP

The following topics describe advanced information that can help you understand how the Test Script Processor (TSP®) scripting engine works.

Global variables and the script.user.scripts table

When working with script commands, it is helpful to understand how scripts are handled in the instrument.

Scripts are loaded into the runtime environment from nonvolatile memory when you turn the instrument on. They are also added to the runtime environment when you load them into the instrument.

A script in the runtime environment can be:

- A named script
- An unnamed script
- The anonymous script (which is a special unnamed script)

Script names can be assigned by using the loadscript command or by defining the *scriptVar* parameter of the `script.new()` function. When a named script is loaded into the runtime environment:

- A global variable with the same name is created so that you can reference the script more conveniently.
- An entry for the script is added to the `script.user.scripts` table.

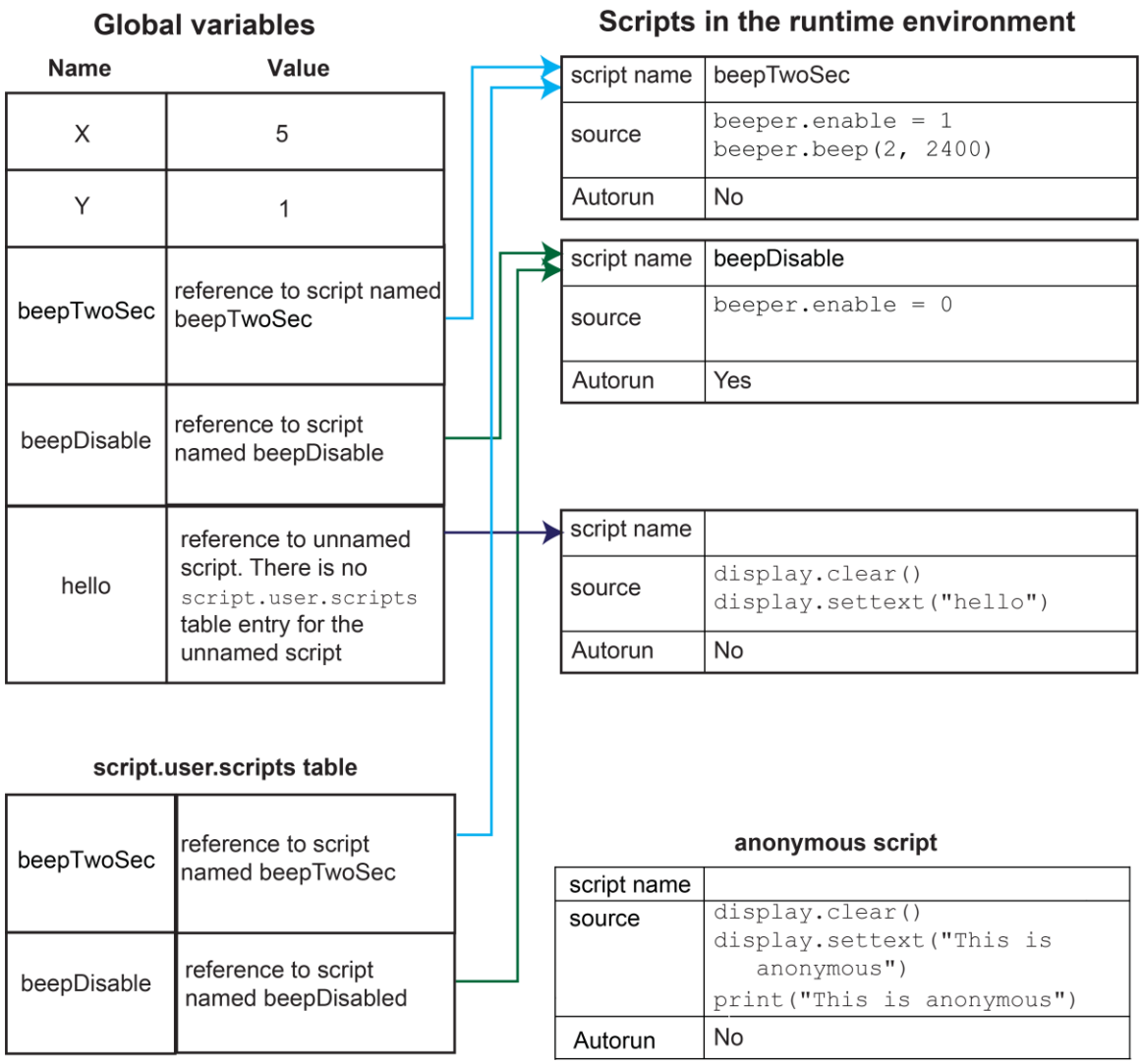
When you create a script using the `script.new()` function without providing a name, the script is added to the runtime environment as an unnamed script. The `script.new()` function returns the script, but the script is not added to the `script.user.scripts` table.

When the anonymous script is loaded, it does not have a global variable or an entry in the `script.user.scripts` table. If there is an existing anonymous script, it is replaced by the new one.

When the instrument is turned off, everything in the runtime environment is deleted, including the scripts and global variables.

See the figure below to see how the scripts, global variables, and `script.user.scripts` table interrelate.

Figure 98: Global variables and scripts in the runtime environment



Create a script using the script.new() command

Use the `script.new()` function to copy an existing script from the local node to a remote node. This enables parallel script execution.

You can create a script with the `script.new()` function using the command:

```
scriptVar = script.new(code, name)
```

Where:

scriptVar = Name of the variable created when the script is loaded into the runtime environment
code = Content of the script
name = Name that is added to the `script.user.scripts` table

For example, to set up a two-second beep, you can send the command:

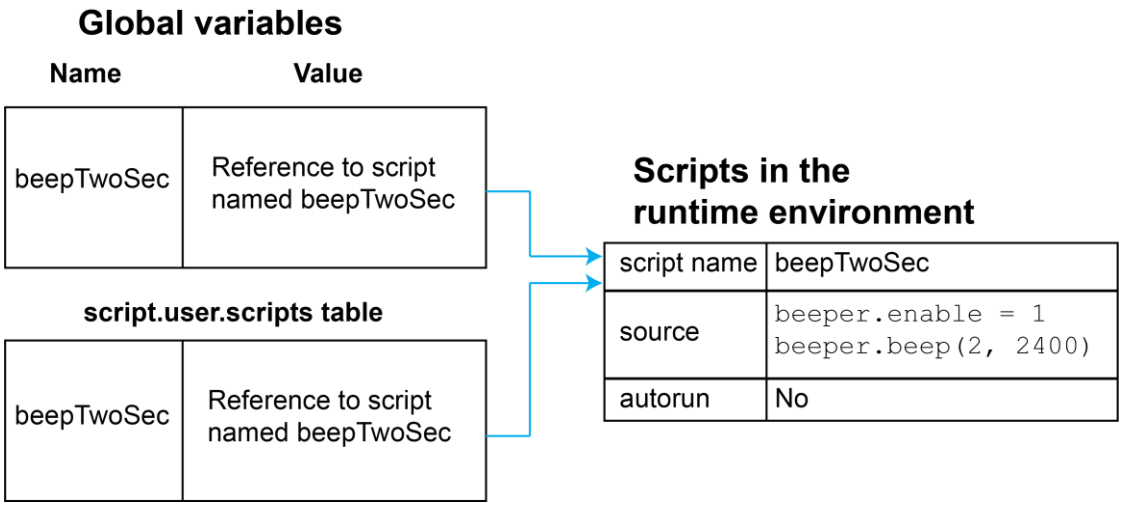
```
beepTwoSec = script.new("beeper.enable = 1 beeper.beep(2, 2400)", "beepTwoSec")
```

To run the new script, send the command:

```
beepTwoSec()
```

When you add `beepTwoSec`, the global variable and `script.user.scripts` table entries are made to the runtime environment, as shown in the following figure.

Figure 99: Runtime environment after creating a script



Create an unnamed script using script.new()

NOTE

Unnamed scripts are not available from the front-panel display of the instrument. Only the anonymous script and named scripts are available from the front-panel display.

When you create a script using `script.new()`, if you do not include *`name`*, the script is added to the runtime environment as an unnamed script. The `script.new()` function returns the script. You can assign it to a global variable, a local variable, or ignore the return value. A global variable is not automatically created.

For example, send the following command:

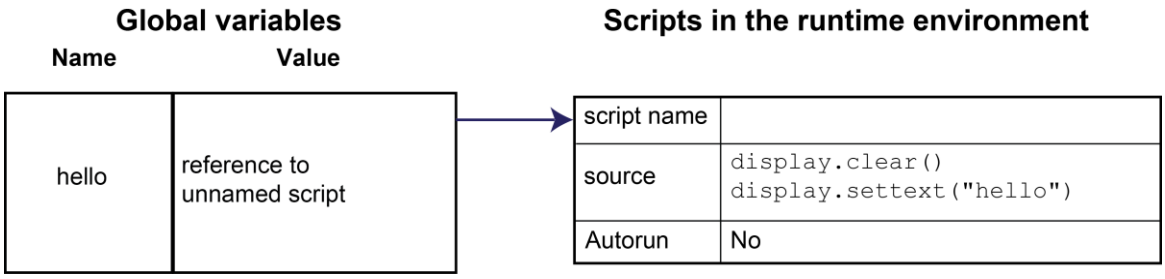
```
hello = script.new('display.clear() display.settext("hello")')
```

A script is created in the runtime environment and a global variable is created that references the script.

To run the script, send the command:

```
hello()
```

Figure 100: Create an unnamed script



A script becomes unnamed if you create a new script with the same name. In this circumstance, the name of the script in the `script.user.scripts` table is set to an empty string before it is replaced by the new script.

For example, if `beepTwoSec` already exists in the `script.user.scripts` table and you sent:

```
beepTwoSec1200 = script.new("beeper.enable = 1 beeper.beep(2, 1200)", "beepTwoSec")
```

The following actions occur:

- `beepTwoSec1200` is added as a global variable.
- The script that was in the runtime environment as `beepTwoSec` is changed to an unnamed script (the name attribute is set to an empty string).
- The global variable `beepTwoSec` remains in the runtime environment unchanged (it points to the now unnamed script).
- A new script named `beepTwoSec` is added to the runtime environment.

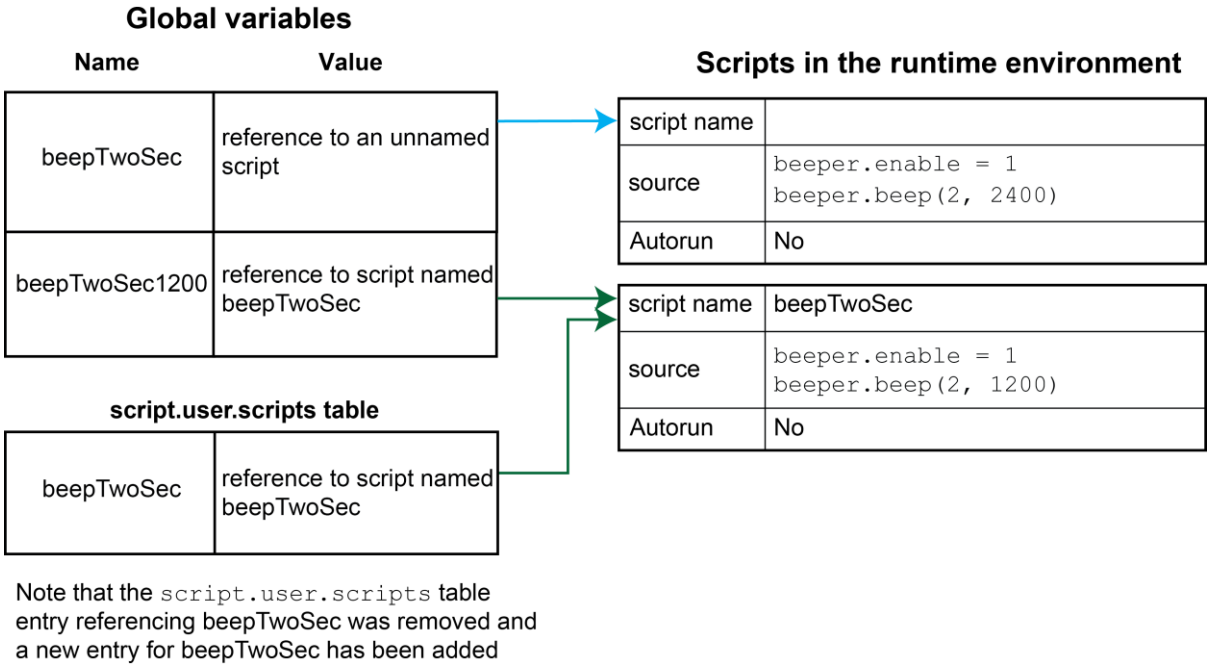
In this example, you can access the new script by sending either of the following commands:

```
beepTwoSec1200()
script.user.scripts.beepTwoSec()
```

To access the unnamed script, you can send the command:

```
beepTwoSec()
```

Figure 101: Create a new script with the name of an existing script



Rename a script

You can rename a script. You might want to rename a script if you need to name another script the same name as the existing script. You could also rename an existing script to be the autoexec script.

To change the name of a script, use the command:

```
scriptVar.name = "renamedScript"
```

Where:

scriptVar	=	The global variable name
"*renamedScript*"	=	The new name of the user script that was referenced by the *scriptVar* global variable

After changing the name, you need to save the original script to save the change to the name attribute.

For example:

```
beepTwoSec.name = "beep2sec"
beepTwoSec.save()
```

Run the `beep2sec` script using the following command:

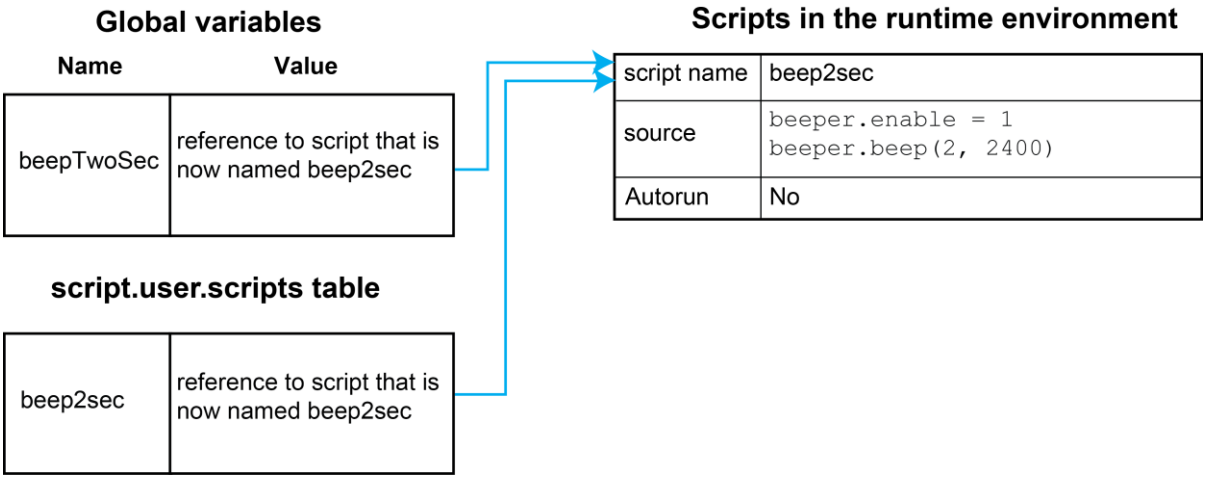
```
script.user.scripts.beep2sec()
```

NOTE

If the new name is the same as a name that is already used for a script, the name of the existing script is removed and that script becomes unnamed. This removes the existing script if there are no other variables that reference the previous script. If variables do reference the existing script, the references remain intact.

Changing the name of a script does not change the name of any variables that reference that script. After changing the name, the script is in the `script.user.scripts` table under its new name.

Figure 102: Rename script



For example, to change the name of the script named `test2` to be `autoexec`:

```
test2.name = "autoexec"
test2.save()
```

The `autoexec` script runs automatically when the instrument is turned on. It runs after all the scripts have loaded and any scripts marked as autorun have run.

NOTE

You can also use the `script.new()` and the *`scriptVar`*`.source` attribute commands to create a script with a new name. For example, if you had an existing script named `test1`, you could create a new script named `test2` by sending the command:

```
test2 = script.new(test1.source, "test2")
```

See script.new() (on page 7-202).

Retrieve a user script

There are several ways to retrieve the source code of a user script:

- One line at a time: Use *scriptVar*.list() to retrieve the source code one line at a time
- Entire script: Use the print(*scriptVar*.source) command to retrieve the script source code as a single string
- Use TSB Embedded; refer to Working with TSB Embedded (on page 6-39) for more information

See Create and load a script (on page 6-4) for information about recreating the script and loading it back into the instrument.

NOTE

To get a list of scripts that are in nonvolatile memory, use the script.user.catalog() (on page 7-205) function.

Retrieve source code one line at a time

To retrieve the source code one line at a time, send the *scriptVar*.list() command. When this command is received, the instrument sends the entire script. Each line of the script is sent as a separate response message. The output includes the loadscript or loadandrunscript and endscript keywords.

After retrieving the source code, you can modify and save the command lines as a user script under the same name or a new name.

To retrieve the source code of a script one line at a time, send the command:

```
scriptVar.list()
```

Where *scriptVar* is the name of the script.

NOTE

To retrieve the commands in the anonymous script, use script.anonymous.list().

Example: Retrieve source code one line at a time

`test.list()`	Retrieve the source of a script named "test".
	The output looks similar to:
	`loadscript test`
	`display.clear()`
	`display.settext("This is a test")`
	`print("This is a test")`
	`endscript`

Retrieve a script as a single string

To retrieve the entire user script source code as a single string, use the *scriptVar*.source attribute. The loadscript or loadandrunscript and endscript keywords are not included.

To retrieve the source code as a single string, send the command:

```
print(scriptVar.source)
```

Where *scriptVar* is the name of the script.

Example: Retrieve the source code as a single string

`print(test.source)`	Retrieve the source of a script named "test". Output looks similar to: `display.clear() display.settext("This is a test") print("This is a test")`

Retrieve a script using TSB Embedded

In TSB Embedded, from the User Scripts list, select the script you want to retrieve. The contents of the script are displayed. See Working with TSB Embedded (on page 6-39) for more information.

Script example: Retrieve the content of scripts

This set of examples:

- Retrieves the source of a script using *scriptVar*.list()
- Retrieves the source of a script using *scriptVar*.source

Example: Retrieve the content of a script with scriptVar.list()

`test.list()`	Request a listing of the source of test.
	An example of the possible instrument output is shown here (note that the loadscript and endscript commands are included).
	
	Output:
	`loadscript scriptVarTest`
	`listTones = {100, 400, 800}`
	`for index in listTones do`
	`  beeper.beep(.5, listTones[index])`
	`end`
	`endscript`

Example: Retrieve the content of a script with scriptVar.source

`print(test.source)`	Request a listing of the source of the script named `test`. The `loadscript` and `endscript` commands are not included.
	An example of the possible instrument output is:
	`listTones = {100, 400, 800}`
	`for index in listTones do`
	`  beeper.beep(.5, listTones[index])`
	`end`

Delete user scripts from the instrument

In most circumstances, you can delete a script using `script.delete()` (as described in Delete user scripts (on page 6-13)), and then turn the instrument off and back on again. However, if you cannot turn the instrument off, you can use the following steps to completely remove a script from the instrument.

When you completely remove a script, you delete all references to the script from the runtime environment, the `script.user.scripts` table, and nonvolatile memory.

To completely remove a script:

1. **Remove the script from the runtime environment.** Set any variables that refer to the script to `nil` or assign the variables a different value. For example, to remove the script `"beepTwoSec"` from the runtime environment, send the following code:
   `beepTwoSec = nil`
2. **Remove the script from the script.user.scripts table.** Set the name attribute to an empty string (`""`). This makes the script nameless, but does not make the script become the anonymous script. For example, to remove the script named `"beepTwoSec"`, send the following code:
   `script.user.scripts.beepTwoSec.name = ""`
3. **Remove the script from nonvolatile memory.** To delete the script from nonvolatile memory, send the command:
   `script.delete("`*`name`*`")`

   Where *`name`* is the name that the script was saved as. For example, to delete `"beepTwoSec"`, send:
   `script.delete("beepTwoSec")`

Restore a script to the runtime environment

You can retrieve a script that was removed from the runtime environment but is still saved in nonvolatile memory.

To restore a script from nonvolatile memory into the runtime environment, you can use `script.restore("scriptName")`, where *scriptName* is the user-defined name of the script to be restored.

For example, to restore a user script named `"test9"` from nonvolatile memory:

```
script.restore("test9")
```

Memory considerations for the runtime environment

The Model 2651A reserves 32 MB of memory for dynamic runtime use. Approximate allocation of this memory is shown below:

5 MB	Firmware general operation
1 MB	Reserve for instrument internal operation
2 MB	Reserve for future firmware updates
24 MB	Runtime environment, user-created reading buffers, and active sweep configuration

Note that the runtime environment, user-created reading buffers, and active sweep configuration must fit in the 24 MB of memory that is available. The amount of memory used by a reading buffer is approximately 15 bytes for each entry requested.

Reading buffers also use a small amount of memory for reading buffer management, which is not significant when making memory utilization calculations. For example, assume two reading buffers were created. One of them was created to store up to 1,000 readings and the other to store up to 2,500 readings. The memory reserved for the reading buffers is calculated as follows:

(1000 * 15) + (2500 * 15) = 52,500 bytes or 52.5 kilobytes

Note that the dedicated reading buffers do not consume memory that is needed by the runtime environment; do not include them in your memory consumption calculations. Also, reading buffers for remote nodes consume memory on the remote node, not the local node. Make sure the total reading buffer memory for any particular remote node does not exceed 24 MB, but do not include that amount in your local memory consumption calculations.

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

It is possible for the memory used for the runtime environment, sweep configuration and reading buffers to exceed 24 MB. When this occurs, there is a risk that memory allocation errors will occur and commands will not be executed as expected.

CAUTION

If the instrument encounters memory allocation errors when the memory used is above 95 percent, the state of the instrument cannot be guaranteed. After attempting to save any important data, turn off power to the instrument and turn it back on to reset the runtime environment and return the instrument to a known state. Unsaved scripts and data in reading buffers will be lost.

The amount of memory in use can be checked using the `meminfo()` function. The first value returned by `meminfo()` is the number of kilobytes of memory in use.

If the amount of memory used is over 95 percent or if you receive out-of-memory errors, you should reduce the amount of memory that is used.

Some suggestions for increasing the available memory:

- Turn the instrument off and on. This deletes scripts that have not been saved and reloads only scripts that have been stored in nonvolatile memory.
- Remove unneeded scripts from nonvolatile memory. Scripts are loaded from nonvolatile memory into the runtime environment when the instrument is turned on. See Delete user scripts from the instrument (on page 6-52).
- Reduce the number of TSP-Link® nodes.
- Delete unneeded global variables from the runtime environment by setting them to `nil`.
- Set the source attribute of all scripts to `nil`.
- Adjust the `collectgarbage()` settings in Lua. See Lua memory management (on page 6-31) for more information.
- Review scripts to optimize their memory usage. In particular, you can see memory gains by changing string concatenation lines into a Lua table of string entries. You can then use the `table.concat()` function to create the final string concatenation.

TSP-Link system expansion interface

The TSP-Link® expansion interface allows the Model 2651A instrument to communicate with other Test Script Processor (TSP®) enabled instruments. The test system can be expanded to include up to 32 TSP-Link enabled instruments.

CAUTION

Combining two Model 2651A instruments to achieve greater currents in both source voltage and source current applications requires specific precautions, including configuration settings. Make sure that you adequately understand the risks involved and the measures needed to accommodate the combination of two Model 2651A instruments. To prevent damage to the Model 2651A, connected instruments, and the device under test, make sure proper procedures are used. For further information, visit the Keithley website at tek.com/keithley for application notes on combining two Model 2651A channels.

Although the expanded system can control up to 32 TSP-Link enabled instruments, combining the output of a Model 2651A to increase either voltage or current is only permitted with another Model 2651A.

Master and subordinates

In a TSP-Link system, one of the nodes (instruments) is the master node and the other nodes are the subordinate nodes. The master node in a TSP-Link system can control the other nodes (subordinates) in the system.

When any node transitions from local operation to remote operation, it becomes the master of the system. All other nodes also transition to remote operation and become its subordinates. When any node transitions from remote operation to local, all other nodes also transition to local operation, and the master/subordinate relationship between nodes is dissolved.

The expanded system can be stand-alone or computer-based.

Stand-alone system: You can run a script from the front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation (REM indicators turn on). The node running the script becomes the master and can control all other nodes, which become its subordinates. When the script is finished running, all the nodes in the system return to local operation (REM indicators turn off), and the master/subordinate relationship between nodes is dissolved.

Computer-based system: You can use a computer and a remote communications interface to any single node in the system. This node becomes the interface to the entire system. When a command is sent through this node, all nodes go into remote operation (REM indicators turn on). The node that receives the command becomes the master and can control all other nodes, which become its subordinates. In a computer-based system, the master/subordinate relationship between nodes can only be dissolved by performing an abort operation.

TSP-Link system

You can use the TSP-Link® expansion interface to expand your test system to include up to 32 addressable TSP® enabled instruments that use the TSP-LINK®. The expanded system can be stand-alone or computer-based.

Stand-alone system: You can run a script from the front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation (REM indicators turn on). The node running the script becomes the master and can control all of the other nodes, which become its subordinates. When the script is finished running, all the nodes in the system return to local operation (REM indicators turn off), and the master/subordinate relationship between nodes is dissolved.

Computer-based system: You can use a computer and a LAN, GPIB, or RS-232 interface to any single node in the system. This node becomes the interface to the entire system. When a command is sent through this node, all nodes go into remote operation (REM indicators turn on). The node that receives the command becomes the master and can control all of the other nodes, which become its subordinates. In a computer-based system, the master/subordinate relationship between nodes can only be dissolved by performing an abort operation.

TSP-Link nodes

Each instrument (node) attached to the TSP-Link® network must be identified by assigning it a unique TSP-Link node number.

Commands for remote nodes are stored in the `node` table. An individual node is accessed as `node[`*N*`]`, where *N* is the node number assigned to the node.

All TSP-accessible remote commands can be accessed as elements of the specific node. The following attributes are examples of items you can access:

- `node[`*N*`].model`: The product model number string of the node.
- `node[`*N*`].revision`: The product revision string of the node.
- `node[`*N*`].serialno`: The product serial number string of the node.

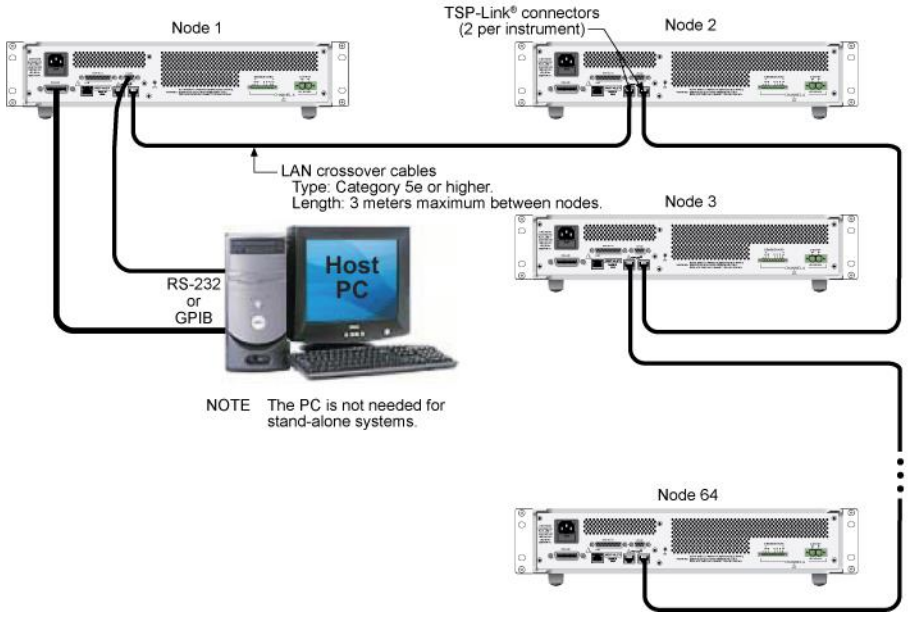
You do not need to know the node number of the node that is running a script. The variable `localnode` is an alias for the node entry of the node where the script is running. For example, if a script is running on node 5, you can use the global variable `localnode` as an alias for `node[5]`. To access the product model number for this example, use `localnode.model`.

Connections

Connections for an expanded system are shown in the following figure. As shown, one instrument is optionally connected to the computer using the GPIB, LAN, or RS-232 interface. Details about these computer communication connections are described in Remote communications interfaces (on page 2-80).

All the instruments in the system are connected in a sequence (daisy-chained) using LAN crossover cables.

Figure 103: Series 2650 TSP-Link connections



Initialization

Before you can use a TSP-Link® system, it must be initialized. For initialization to succeed, each instrument in a TSP-Link system must be assigned a different node number.

Assigning node numbers

At the factory, each Model 2651A instrument is assigned as node 1. The node number is stored in nonvolatile memory and remains in storage when the instrument is turned off. You can assign a node number to a Model 2651A using the front panel or by using a remote command. Note that there can only be 32 physical nodes, but you can assign node numbers from 1 to 64.

To assign a node number from the front panel of the instrument:

1. Press the **MENU** key, then select **TSPLINK > NODE**.
2. Press the navigation wheel and select the node number.
3. Press the **ENTER** key to save the number.

To assign a node number using a remote command:

Set the `tsplink.node` attribute of the instrument:

```
tsplink.node = N
```

Where *N* = `1` to `64` To determine the node number of an instrument, you can read the `tsplink.node` attribute by sending the following command:

```
print(tsplink.node)
```

The above `print` command outputs the node number. For example, if the node number is 1, a `1` is displayed.

Resetting the TSP-Link network

After all the node numbers are set, you must initialize the system by performing a TSP-Link® network reset.

NOTE

If you change the system configuration after initialization, you must reinitialize the system by performing a TSP-Link network reset. Changes that require that you reinitialize the TSP-Link network include turning off power or rebooting any instrument in the system, or rearranging or disconnecting the TSP-Link cable connections between instruments.

Front-panel operation

To reset the TSP-Link® network from the front panel:

1. Power on all instruments connected to the TSP-Link network.
2. Press the **MENU** key, select **TSPLINK**, and then press the **ENTER** key.
3. Turn the navigation wheel to select **RESET**, and then press the **ENTER** key.

Remote programming

The commands associated with the TSP-Link® system reset are listed in the following table.

TSP-Link reset commands

Command	Description
`tsplink.reset()`	Initializes the TSP-Link network
`tsplink.state`	Reads the state of the TSP-Link network: ■ "`online`" if the most recent TSP-Link reset was successful ■ "`offline`" if the reset operation failed

An attempted TSP-Link reset operation fails if any of the following conditions are true:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument performing the reset (only if the expected number of nodes was not provided in the reset call)
- One or more of the instruments in the system is turned off
- If the actual number of nodes is less than the expected number

The programming example below illustrates a TSP-Link reset operation and displays its state:

```
tsplink.reset()
print(tsplink.state)
```

If the reset operation is successful, `online` is output to indicate that communications with all nodes have been established.

Accessing nodes

You can access all the remote commands for a specific node by adding `node[N].` to the beginning of the remote command, where *N* is the node number. For example, to set the NPLC value for the source-measure unit (SMU) A on node 1 to 0.1, you could send this command:

```
node[1].smua.measure.nplc = 0.1
```

The variable `localnode` is an alias for `node[N]`, where *N* is the node number of the node on which the code is running. For example, if node 1 is running the code, `localnode` can be used instead of `node[1]`.

The following programming examples illustrate how to access instruments in the TSP-Link system (shown in TSP-Link connections):

- Any of the following three commands can be used to reset SMU A of node 1 (which, in this example, is the master). The other nodes in the system are not affected.

```
smua.reset()
localnode.smua.reset()
node[1].smua.reset()
```

- The following command will reset SMU A of node 4, which is a subordinate. The other nodes are not affected.

```
node[4].smua.reset()
```

Using the reset() command

Most TSP-Link® system operations target a single node in the system, but the `reset()` command affects the system as a whole by resetting all nodes to their default settings:

```
-- Reset all nodes in a TSP-Link system to their default state.
reset()
```

NOTE

Using the `reset()` command in a TSP-Link network differs from using the `tsplink.reset()` command. The `tsplink.reset()` command reinitializes the TSP-Link network and turns off the output of any TSP-linked instrument; it may change the state of individual nodes in the system.

Use `node[N].reset()` or `localnode.reset()` to reset only one of the nodes. The other nodes are not affected. The following programming example shows this type of reset operation with code that is run on node 1.

```
-- Reset node 1 only.
node[1].reset()
-- Reset the node you are connected to (in this case, node 1).
localnode.reset()
-- Reset node 4 only.
node[4].reset()
```

Using the abort command

An `abort` command terminates an executing script and returns all nodes to local operation (REM indicators turn off). This dissolves the master/subordinate relationships between nodes. To invoke an abort operation, either send an `abort` command to a specific node or press the EXIT (LOCAL) key on any node in the system.

You can also perform an abort operation by pressing the OUTPUT ON/OFF control on any node. The results are the same as above, with the addition that all source-measure unit (SMU) outputs in the system are turned off.

Triggering with TSP-Link

The TSP-Link® expansion interface has three trigger lines that function similarly to the digital I/O synchronization lines. See Digital I/O (on page 3-92) and Triggering (on page 3-36) for more information.

TSP advanced features

Use the Test Script Processor (TSP®) scripting engine's advanced features to:

- Run test scripts simultaneously
- Manage resources allocated to test scripts that are running simultaneously
- Use the data queue to facilitate real-time communication between nodes on the TSP-Link® network

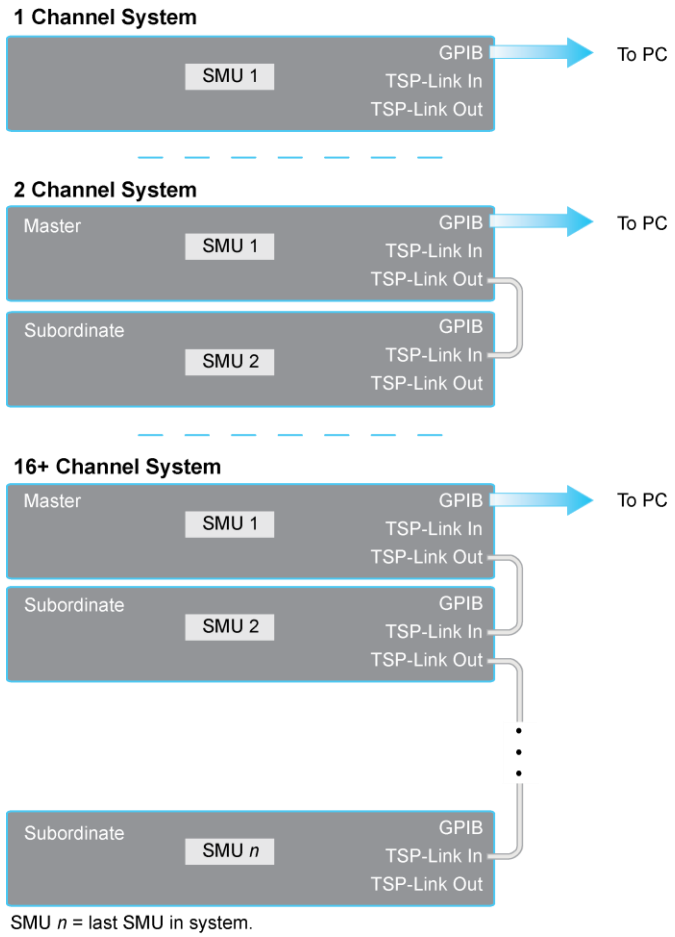
When test scripts are run simultaneously, it improves functional testing, provides higher throughput, and expands system flexibility.

There are two methods you can use to run test scripts simultaneously:

- Create multiple TSP-Link networks
- Use a single TSP-Link network with groups

The following figure displays the first method, which consists of multiple TSP-Link networks. Each TSP-Link network has a master node and a remote connection to the computer.

Figure 104: Model 2651A multiple TSP-Link networks

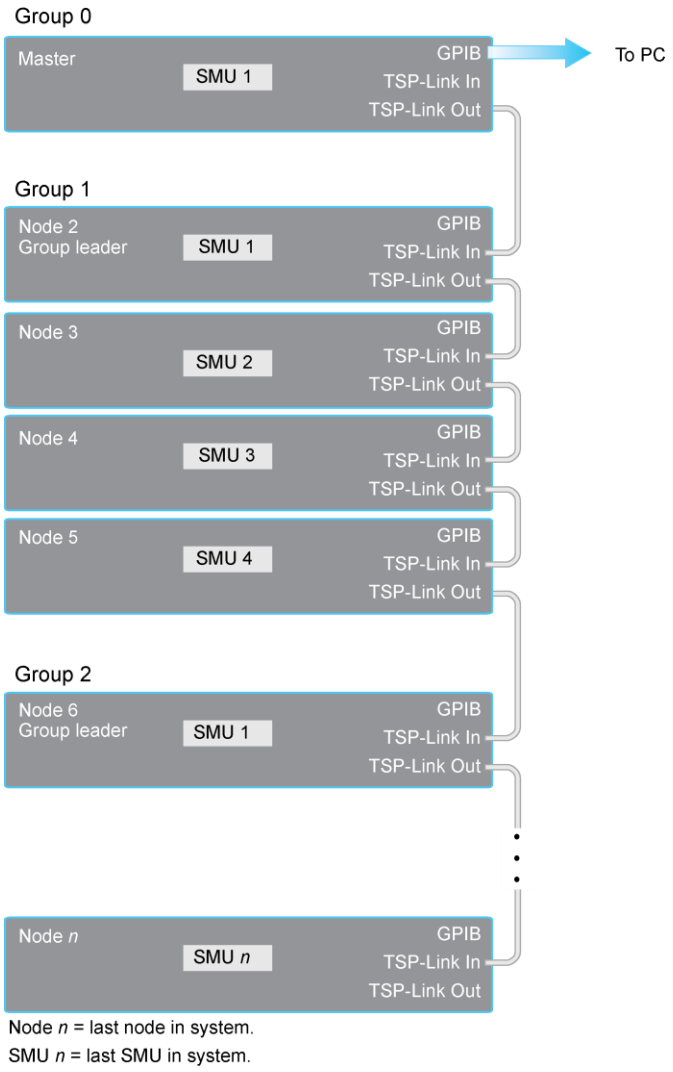


SMU *n* = last SMU in system.

Another method you can use to run simultaneous test scripts is to use groups with a single TSP-Link network. Each group on the TSP-Link network can run a test while other groups are running different tests.

A group consists of one or more nodes with the same group number. The following figure displays a single TSP-Link network with groups. This method requires one TSP-Link network and a single GPIB connection to the computer.

Figure 105: Model 2651A TSP-Link network with groups



The following table shows an example of the functions of a single TSP-Link network. Each group in this example runs a different test script than the other groups, which allows the system to run multiple tests simultaneously.

TSP-Link network group functions

Group number	Group members	Present function
0	Master node 1	Initiates and runs a test script on node 2 Initiates and runs a test script on node 6
1	Group leader node 2	Runs the test script initiated by the master node Initiates remote operations on node 3 through node 5
	Node 3 through node 5	Performs remote operations initiated by node 2
2	Group leader node 6	Runs the test script initiated by the master node Initiates remote operations through node *n*
	Node *n*	Performs remote operations initiated by node 6

Using groups to manage nodes on TSP-Link network

The primary purpose of groups is to allow each group to run a different test script simultaneously.

A group can consist of one or more nodes. You must assign group numbers to each node using remote commands. If you do not assign a node to a group, it defaults to group 0, which is always grouped with the master node (regardless of the group to which the master node is assigned).

Master node overview

You can assign the master node to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to group 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

The master node is always the node that coordinates activity on the TSP-Link network.

The master node:

- Is the only node that can use the `execute()` command on a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation (a command that continues to operate after the command that initiated it has finished running)
- Can execute the `waitcomplete()` command to wait for the group to which the master node belongs; to wait for another group; or to wait for all nodes on the TSP-Link network to complete overlapped operations (overlapped commands allow the execution of subsequent commands while device operations of the overlapped command are still in progress)

Group leader overview

Each group has a dynamic group leader. The last node in a group that performs any operation initiated by the master node is the group leader.

The group leader:

- Performs operations initiated by the master node
- Initiates remote operations on any node with the same group number
- Cannot initiate remote operations on any node with a different group number
- Can use the `waitcomplete()` command without a parameter to wait for all overlapped operations running on nodes in the same group

Assigning groups

Group numbers can range from zero (0) to 64. The default group number is 0. You can change the group number at any time. You can also add or remove a node to or from a group at any time.

Each time the power for a node is turned off, the group number for that node changes to 0.

The following example code dynamically assigns a node to a group:

```
-- Assign node 3 to group 1.
node[3].tsplink.group = 1
```

Running simultaneous test scripts

You can send the `execute()` command from the master node to initiate a test script and Lua code on a remote node. The `execute()` command places the remote node in the overlapped operation state. As a test script runs on the remote node, the master node continues to process other commands simultaneously.

Use the following code to send the `execute()` command for a remote node. The *N* parameter represents the node number that runs the test script (replace *N* with the node number).

To set the global variable "setpoint" on node N to 2.5:

```
node[N].execute("setpoint = 2.5")
```

The following code demonstrates how to run a test script that is defined on the local node. For this example, *scriptVar* is defined on the local node, which is the node that initiates the code to run on the remote node. The local node must be the master node.

To run scriptVar on node N:

```
node[N].execute(scriptVar.source)
```

The programming example below demonstrates how to run a test script that is defined on a remote node. For this example, *scriptVar* is defined on the remote node.

To run a script defined on the remote node:

```
node[N].execute("scriptVar()")
```

It is recommended that you copy large scripts to a remote node to improve system performance. See Copying test scripts across the TSP-Link network (on page 6-67) for more information.

Coordinating overlapped operations in remote groups

All overlapped operations on all nodes in a group must have completed before the master node can send a command to the group. If you send a command to a node in a remote group when an overlapped operation is running on any node in that group, errors occur.

You can execute the `waitcomplete()` command on the master node or group leader to wait for overlapped operations. The action of `waitcomplete()` depends on the parameters specified.

If you want to wait for completion of overlapped operations for:

- **All nodes in the local group:** Use `waitcomplete()` without a parameter from the master node or group leader.
- **A specific group:** Use `waitcomplete(N)` with a group number as the parameter from the master node. This option is not available for group leaders.
- **All nodes in the system:** Use `waitcomplete(0)` from the master node. This option is not available for group leaders.

For additional information, refer to waitcomplete() (on page 7-459).

The following code shows two examples of using the `waitcomplete()` command from the master node:

```
-- Wait for each node in group N to complete all overlapped operations.
waitcomplete(N)
-- Wait for all groups on the TSP-Link network to complete overlapped operations.
waitcomplete(0)
```

A group leader can issue the `waitcomplete()` command to wait for the local group to complete all overlapped operations.

The following code is an example of how to use the `waitcomplete()` command from a group leader:

```
-- Wait for all nodes in the local group to complete all overlapped operations.
waitcomplete()
```

Using the data queue for real-time communication

Nodes that are running test scripts at the same time can store data in the data queue for real-time communication. Each instrument has an internal data queue that uses the first-in, first-out (FIFO) structure to store data. You can use the data queue to post numeric values, strings, and tables.

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link network at any time

You cannot access the reading buffers or global variables from any node in a remote group while a node in that group is performing an overlapped operation. However, you can use the data queue to retrieve data from any node in a group that is performing an overlapped operation. In addition, the master node and the group leaders can use the data queue to coordinate activities.

Tables in the data queue consume one entry. When a node stores a table in the data queue, a copy of the data in the table is made. When the data is retrieved from the data queue, a new table is created on the node that is retrieving the data. The new table contains a separate copy of the data in the original table, with no references to the original table or any subtables.

You can access data from the data queue even if a remote group or a node has overlapped operations in process. See the `dataqueue` commands for more information.

Copying test scripts across the TSP-Link network

To run a large script on a remote node, copy the test script to the remote node to increase the speed of test script initiation.

The code in the example below copies a test script across the TSP-Link® network, creating a copy of the script on the remote node with the same name.

```
-- Add the source code from the script
-- testScript to the data queue.
node[2].dataqueue.add(testScript.source)
-- Create a new script on the remote node
-- using the source code from testScript.
node[2].execute(testScript.name ..
   "= script.new(dataqueue.next(), [[" .. testScript.name .. "]])")
```

Removing stale values from the reading buffer cache

The node that acquires the data also stores the data for the reading buffer. To optimize data access, all nodes can cache data from the node that stores the reading buffer data.

When you run Lua code remotely, it can cause reading buffer data that is held in the cache to become stale. If the values in the reading buffer change while the Lua code runs remotely, another node can hold stale values. Use the `clearcache()` command to clear the cache. For additional detail on the reading buffer cache commands, see bufferVar.cachemode (on page 7-20) and bufferVar.clearcache() (on page 7-22).

The following example code demonstrates how stale values occur and how to use the `clearcache()` command to clear the cache on node 2, which is part of group 7.

```
-- Create a reading buffer on a node in a remote group.
node[2].tsplink.group = 7
node[2].execute("rbremote = smua.makebuffer(20) " ..
                "smua.measure.count = 20 " ..
                "smua.measure.v(rbremote)")
-- Create a variable on the local node to
-- access the reading buffer.
rblocal = node[2].getglobal("rbremote")
-- Access data from the reading buffer.
print(rblocal[1])
-- Run code on the remote node that updates the reading buffer.
node[2].execute("smua.measure.v(rbremote)")
-- Use the clearcache command if the reading buffer contains cached data.
rblocal.clearcache()
-- If you do not use the clearcache command, the data buffer
-- values never update. Every time the print command is
-- issued after the first print command, the same data buffer
-- values print.
print(rblocal[1])
```

TSP-Net

The TSP-Net® library allows the Model 2651A to control LAN-enabled devices directly through its LAN port. This enables the Model 2651A to communicate directly with a device that is not TSP® enabled without the use of a controlling computer.

TSP-Net capabilities

The TSP-Net library permits the Model 2651A to control a remote instrument through the LAN port for both Test Script Processor (TSP®) and non-TSP instruments. Using TSP-Net library methods, you can transfer string data to and from a remote instrument, transfer and format data into Lua variables, and clear input buffers. The TSP-Net library is only accessible using commands from a remote command interface.

You can use TSP-Net commands to communicate with any ethernet-enabled instrument. However, specific TSP-Net commands exist for TSP-enabled instruments to allow for support of features unique to the TSP scripting engine. These features include script downloads, reading buffer access, wait completion, and handling of TSP scripting engine prompts.

Using TSP-Net commands with TSP-enabled instruments, a Model 2651A can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The Model 2651A can read the data from the remote instrument and either manipulate the data or send the data to a different remote instrument on the LAN. You can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the Model 2651A.

Using TSP-Net with any ethernet-enabled instrument

NOTE

Refer to TSP command reference (on page 7-1) for details about the commands presented in this section.

The Model 2651A LAN port is auto-sensing (Auto-MDIX), so you can use either a LAN crossover cable or a LAN straight-through cable to connect directly from the Model 2651A to an ethernet device or to a hub.

To set up communication to a remote ethernet-enabled instrument that is TSP® enabled:

1. Send the following command to configure TSP-Net to send an abort command when a connection to a TSP instrument is established:

```
tspnet.tsp.abortonconnect = 1
```

   If the scripts are allowed to run, the connection is made, but the remote instrument may be busy.

2. Send the command:

```
connectionID = tspnet.connect(ipAddress)
```

   Where:

   - *connectionID* is the connection ID that is used as a handle in all other TSP-Net function calls.
   - *ipAddress* is the IP address, entered as a string, of the remote instrument.

See tspnet.connect() (on page 7-443) for additional detail.

To set up communication to a remote ethernet-enabled device that is not TSP enabled:

Send the command:

```
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

Where:

- *connectionID* is the connection ID that is used as a handle in all other `tspnet` function calls.
- *ipAddress* is the IP address, entered as a string, of the remote device.
- *portNumber* is the port number of the remote device.
- *initString* is the initialization string that is to be sent to *ipAddress*.

See tspnet.connect() (on page 7-443) for additional detail.

To communicate to a remote ethernet device from the Model 2651A:

1. Connect to the remote device using one of the above procedures. If the Model 2651A cannot make a connection to the remote device, it generates a timeout event. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 s.
2. Use `tspnet.write()` or `tspnet.execute()` to send strings to a remote device. If you use:
   - `tspnet.write()`: Strings are sent to the device exactly as indicated, and you must supply any needed termination characters.
   - `tspnet.execute()`: The Model 2651A appends termination characters to all strings that are sent. Use `tspnet.termination()` to specify the termination character.
3. To retrieve responses from the remote instrument, use `tspnet.read()`. The Model 2651A suspends operation until the remote device responds or a timeout event is generated. To check if data is available from the remote instrument, use `tspnet.readavailable()`.
4. Disconnect from the remote device using the `tspnet.disconnect()` function. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote device that is not TSP® enabled, and send and receive data from this device:

```
-- Set tspnet timeout to 5 s.
tspnet.timeout = 5
--  Establish connection to another device with IP address 192.168.1.51
-- at port 1394.
id_instr = tspnet.connect("192.168.1.51", 1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set the termination character to CRLF. You must do this
-- for each connection after the connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device.
tspnet.write(id_instr, "login admin\r\n")
-- Read the data available, then print it.
tspnet.write(id_instr, "*idn?\r\n")
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

This example produces a return such as:

```
ID is:    1
instrument write/read returns:       SUCCESS: Logged in
instrument write/read returns:       KEITHLEY INSTRUMENTS,MODEL
   2651A,04089762,1.6.3d
```

TSP-Net compared to TSP-Link to communicate with TSP-enabled devices

The TSP-Link® network interface is the preferred communication method for most applications where communication occurs between the Model 2651A and another TSP-enabled instrument.

One of the advantages of using the TSP-Link network interface is that TSP-Link connections have three trigger lines that are available to each device on the TSP-Link network. You can use any one of the trigger lines to perform hardware triggering between devices on the TSP-Link network. Refer to Hardware trigger modes (on page 3-65) for details.

However, if the distance between the Model 2651A and the TSP-enabled device is longer than 15 feet, use TSP-Net commands.

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

Example: Using tspnet commands

```
 function telnetConnect(ipAddress, userName, password)
   -- Connect through Telnet to a computer.
   id = tspnet.connect(ipAddress, 23, "")
   -- Read the title and login prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the login name.
   tspnet.write(id, userName .. "\r\n")
   -- Read the login echo and password prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the password information.
   tspnet.write(id, password .. "\r\n")
   -- Read the telnet banner from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
end

function test_tspnet()
   tspnet.reset()
   -- Connect to a computer using Telnet.
   telnetConnect("192.0.2.1", "my_username", "my_password")
   -- Read the prompt back from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   -- Change directory and read the prompt back from the computer.
   tspnet.write(id, "cd c:\\\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Make a directory and read the prompt back from the computer.
   tspnet.write(id, "mkdir TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Change to the newly created directory.
   tspnet.write(id, "cd c:\\TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- if you have data print it to the file.
   -- 11.2 is an example of data collected.
   cmd = "echo " .. string.format("%g", 11.2) .. " >> datafile.dat\r\n"
   tspnet.write(id, cmd)
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   tspnet.disconnect(id)
end
test_tspnet()
```

Section 7

TSP command reference

In this section:

TSP command programming notes

This section contains general information about using TSP commands.

Placeholder text

This manual uses italicized text to represent the parts of remote commands that must be replaced by user specified values. The following examples show typical uses of italicized text.

Example 1:

`beeper.enable = `*`state`*

Where *state* can be a value (`beeper.ON` or `beeper.OFF`) or an integer (`1` or `0`) that you specify. For example, to set this attribute on, you send one of the following commands:

```
beeper.enable = beeper.ON
```

```
beeper.enable = 1
```

Example 2:

`digio.trigger[`*`N`*`].assert()`

Where *N* is an integer (`1` to `14`) that you specify. For example, to assert trigger line 7, you send:

```
digio.trigger[7].assert()
```

To assert a trigger line with a variable as the integer, you send:

```
triggerline = 7
```

```
digio.trigger[triggerline].assert()
```

Example 3:

smu*X*.trigger.measure.*Y*(*rbuffer*)

Where:

X refers to the source-measure unit (SMU) channel (use a for SMU A).

Y is the measurement type that you specify (v, i, r, or p).

rbuffer is the reading buffer object where the readings are stored.

For example, to use SMU A to make voltage measurements and store them in buffer vbuffername, you send:

```
smua.trigger.measure.v(vbuffername)
```

Syntax rules

Use these syntax requirements to build well-formed instrument control commands.

Instrument commands are case sensitive. Refer to the command reference descriptions for the correct case.

The white space in lists of parameters in functions is optional. For example, the following functions are equivalent:

```
digio.writebit(3,0)
digio.writebit (3, 0)
```

All functions must have a set of parentheses () immediately following the function, even if there are no parameters specified. For example:

```
waitcomplete(G)
timezone = localnode.gettimezone()
```

If there are multiple parameters, they must be separated by commas (,). For example:

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

Time and date values

Time and date values are represented as the number of seconds since some base. Representing time as a number of seconds is referred to as “standard time format.” There are three time bases:

- **UTC 12:00 am Jan 1, 1970.** Some examples of UTC time are reading buffer base timestamps, adjustment dates, and the value returned by os.time().
- **Instrument on.** References time to when the instrument was turned on. The value returned by os.clock() is referenced to the turn-on time.
- **Event.** Time referenced to an event, such as the first reading stored in a reading buffer.

Remote versus local state

The instrument can be in either the local state or the remote state. When in the local state (REM indicator off), the instrument is operated using the front panel controls. When in the remote state (REM indicator on), instrument operation is being controlled by the computer. When the instrument is powered on, it is in the local state.

Remote state

The following actions place the instrument in the remote state:

- Sending a command from the computer to the instrument.
- Running a script (FACTORY or USER test) from the front panel. After the test has completed, the instrument will return to the local state.
- Opening communications between the instrument and Test Script Builder.

While in the remote state, front panel controls are disabled. However, the LOCAL key is active if it has not been locked out. When an interactive script is running, the front panel controls are active, which allows the operator to input parameter values.

The OUTPUT ON/OFF control is always active. If it is pressed when the instrument is in the remote state, the instrument turns the output off (if it is on) and returns to the local state.

Local state

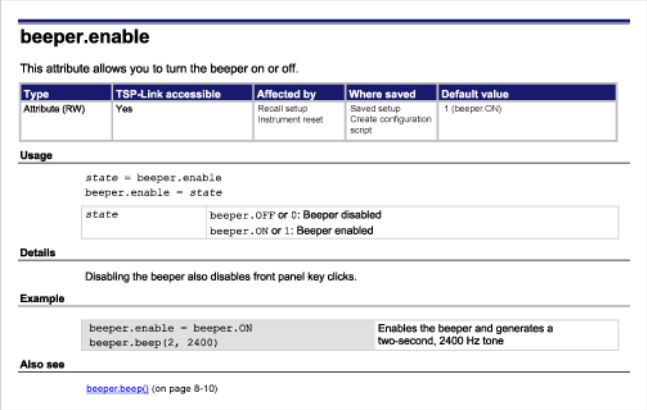
The following actions cancel the remote state and return the instrument to the local state:

- Turning the instrument off and on.
- Pressing the OUTPUT ON/OFF control.
- Pressing front panel LOCAL key (if it is not locked out).
- Sending the `abort` command from the computer.
- Clicking the Abort Execution icon on the toolbar of the Instrument Console for Test Script Builder.
- After a front panel script (FACTORY or USER test) has completed, the instrument will return to the local state.

Using the TSP command reference

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

Figure 106: Example instrument command description



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

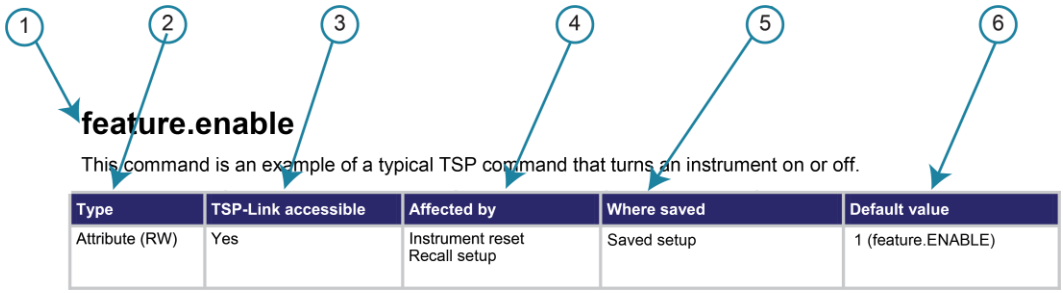
- Command name and summary table
- Usage
- Details
- Example
- Also see

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

Command name and summary table

Each instrument command description starts with the command name, followed by a brief description and a table with relevant information for each command. Definitions for the numbered items in the figure below are listed following the figure.

Figure 107: TSP command name and summary table

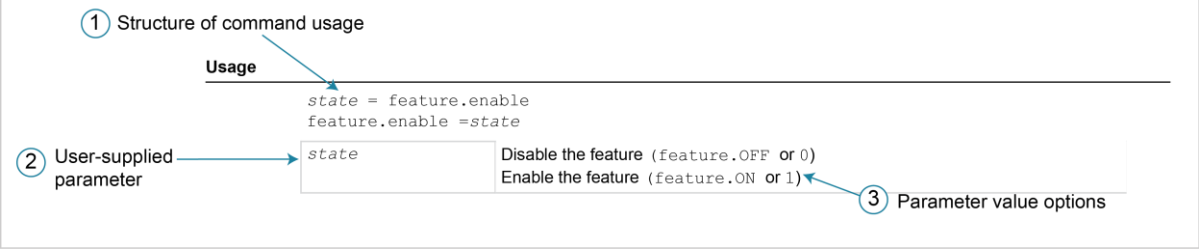


1. **Instrument command name**. Indicates the beginning of the command description. It is followed by a brief description of what the command does.
2. **Type of command**. Commands can be functions, attributes, or constants. If the command is an attribute, it can be read-only (R), read-write (RW), or write-only (W). For detail on commands, see Introduction to TSP operation (on page 5-1).
3. **TSP-Link accessible**. **Yes** or **No**; indicates whether or not the command can be accessed through a TSP-Link network.
4. **Affected by**. Commands or actions that may change the setting of this command.
   - **LAN restore defaults:** This command is reset to the default value when `lan.restoredefaults()` is sent.
   - **Digital I/O trigger N reset:** This command is reset to the default value when `digio.trigger[N].reset()` is sent.
   - **Recall setup:** This command is stored as part of the saved setup and is changed to the value stored in the saved setup when the setup is recalled.
   - **Instrument reset:** This command is reset to the default value when `reset()`, `localnode.reset()`, or `*RST` is sent.
   - **SMU reset:** This command is reset to the default value when `smuX.reset()` is sent.
   - **Power cycle:** This command is set to the default value when the instrument power is cycled.
5. **Where saved**. Indicates where the command settings reside once they are used on an instrument. Options include:
   - **Not saved:** Command is not saved anywhere and must be typed each time you use it.
   - **Nonvolatile memory:** Storage area in the instrument where information is saved when the instrument is turned off.
   - **Saved setup:** Command is saved as part of the saved setup.
6. **Default value:** Lists the default value or constant for the command. The parameter values are defined in the Usage or Details sections of the command description.

Command usage

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

Figure 108: TSP usage description



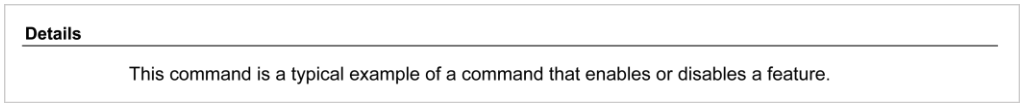
1. **Structure of command usage:** Shows how to organize the parts of the command. If a parameter is shown to the left of the command, it is the return when you print the command. Information to the right is the parameters or other items you need to enter when setting the command.
2. **User-supplied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

   Some commands have optional parameters. If there are optional parameters, they must be entered in the order presented in the Usage section. You cannot leave out any parameters that precede the optional parameter. Optional parameters are shown as separate lines in usage, presented in the required order with each valid permutation of the optional parameters.
   For example:
   `printbuffer(`*`startIndex, endIndex, buffer1`*`)`
   `printbuffer(`*`startIndex, endIndex, buffer1, buffer2`*`)`
3. **Parameter value options:** Descriptions of the options that are available for the user-defined parameter.

Command details

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

Figure 109: TSP Details description



Example section

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

Figure 110: 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 111: TSP Also see description

TSP commands

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

beeper.beep()

This function generates an audible tone.

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

Usage

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

duration	The amount of time to play the tone (0.001 s to 100 s)
frequency	The frequency of the tone in Hertz (Hz)

Details

You can use the beeper of the Model 2651A to provide an audible signal at a specified frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

The beeper does not sound if it is disabled. It can be disabled or enabled with the beeper enable command, or through the front panel.

Example

```
beeper.enable = beeper.ON
beeper.beep(2, 2400)
```

Enables the beeper and generates a two-second, 2400 Hz tone.

Also see

beeper.enable (on page 7-8)

beeper.enable

This command allows you to turn the beeper on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Recall setup Instrument reset	Saved setup	1 (beeper.ON)

Usage

```
state = beeper.enable
beeper.enable = state
```

state	Disable the beeper: `beeper.OFF` or `0` Enable the beeper: `beeper.ON` or `1`

Details

This command enables or disables the beeper. When enabled, a beep signals that a front-panel key has been pressed. Disabling the beeper also disables front-panel key clicks.

Example

```
beeper.enable = beeper.ON
beeper.beep(2, 2400)
```

Enables the beeper and generates a two-second, 2400 Hz tone.

Also see

beeper.beep() (on page 7-7)

bit.bitand()

This function performs a bitwise logical AND operation on two numbers.

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

Usage

```
result = bit.bitand(value1, value2)
```

result	Result of the logical AND operation
value1	Operand for the logical AND operation
value2	Operand for the logical AND operation

Details

Any fractional parts of *value1* and *value2* are truncated to form integers. The returned *result* is also an integer.

Example

```
testResult = bit.bitand(10, 9)
print(testResult)
```

Performs a logical AND operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 8 (binary 1000).
Output:
`8.00000e+00`

Also see

Bit manipulation and logic operations (on page 5-4)
bit.bitor() (on page 7-9)
bit.bitxor() (on page 7-10)

bit.bitor()

This function performs a bitwise logical OR operation on two numbers.

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

Usage

result = bit.bitor(*value1*, *value2*)

result	Result of the logical OR operation
value1	Operand for the logical OR operation
value2	Operand for the logical OR operation

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

Example

```
testResult = bit.bitor(10, 9)
print(testResult)
```

Performs a bitwise logical OR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 11 (binary 1011).
Output:
`1.10000e+01`

Also see

Bit manipulation and logic operations (on page 5-4)
bit.bitand() (on page 7-8)
bit.bitxor() (on page 7-10)

bit.bitxor()

This function performs a bitwise logical XOR (exclusive OR) operation on two numbers.

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

Usage

```
result = bit.bitxor(value1, value2)
```

result	Result of the logical XOR operation
value1	Operand for the logical XOR operation
value2	Operand for the logical XOR operation

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

Example

```
testResult = bit.bitxor(10, 9)
print(testResult)
```

Performs a logical XOR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 3 (binary 0011).
Output:

```
3.00000e+00
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.bitand() (on page 7-8)
bit.bitor() (on page 7-9)

bit.clear()

This function clears a bit at a specified index position.

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

Usage

```
result = bit.clear(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within value to clear (1 to 32)

Details

Any fractional part of *value* is truncated to make it an integer. The returned *result* is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Example

```
testResult = bit.clear(15, 2)
print(testResult)
```

The binary equivalent of decimal 15 is 1111. If you clear the bit at *index* position 2, the returned decimal value is 13 (binary 1101).
Output:

```
1.30000e+01
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.get() (on page 7-12)
bit.set() (on page 7-14)
bit.test() (on page 7-16)
bit.toggle() (on page 7-17)

bit.get()

This function retrieves the weighted value of a bit at a specified index position.

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

Usage

```
result = bit.get(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get (1 to 32)

Details

This function returns the value of the bit in *value* at *index*. This is the same as returning *value* with all other bits set to zero (0).

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for the number is set to zero (0), the result is zero (0).

Example

```
testResult = bit.get(10, 4)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. If you get the bit at index position 4, the returned decimal value is 8.
Output:

```
8.00000e+00
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.clear() (on page 7-11)
bit.set() (on page 7-14)
bit.test() (on page 7-16)
bit.toggle() (on page 7-17)

bit.getfield()

This function returns a field of bits from the value starting at the specified index position.

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

Usage

```
result = bit.getfield(value, index, width)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get (1 to 32)
width	The number of bits to include in the field (1 to 32)

Details

A field of bits is a contiguous group of bits. This function retrieves a field of bits from *value* starting at *index*.

The *index* position is the least significant bit of the retrieved field. The number of bits to return is specified by *width*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Example

```
myResult = bit.getfield(13, 2, 3)
print(myResult)
```

The binary equivalent of decimal 13 is 1101.
The field at *index* position 2 and *width* 3 consists of the binary bits 110. The returned value is decimal 6 (binary 110).
Output:

```
6.00000e+00
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.get() (on page 7-12)
bit.set() (on page 7-14)
bit.setfield() (on page 7-15)

bit.set()

This function sets a bit at the specified index position.

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

Usage

```
result = bit.set(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to set (1 to 32)

Details

This function returns *result*, which is *value* with the indexed bit set. The *index* must be between 1 and 32.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Any fractional part of *value* is truncated to make it an integer.

Example

```
testResult = bit.set(8, 3)

print(testResult)
```

The binary equivalent of decimal 8 is 1000. If the bit at *index* position 3 is set to 1, the returned value is decimal 12 (binary 1100).
Output:

```
1.20000e+01
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.clear() (on page 7-11)
bit.get() (on page 7-12)
bit.getfield() (on page 7-13)
bit.setfield() (on page 7-15)
bit.test() (on page 7-16)
bit.toggle() (on page 7-17)

bit.setfield()

This function overwrites a bit field at a specified index position.

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

Usage

```
result = bit.setfield(value, index, width, fieldValue)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position in *value* to set (1 to 32)
width	The number of bits to include in the field (1 to 32)
fieldValue	Value to write to the field

Details

This function returns *result*, which is *value* with a field of bits overwritten, starting at *index*. The *index* specifies the position of the least significant bit of *value*. The *width* bits starting at *index* are set to *fieldValue*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Before setting the field of bits, any fractional parts of *value* and *fieldValue* are truncated to form integers.

If *fieldValue* is wider than *width*, the most significant bits of the *fieldValue* that exceed the width are truncated. For example, if *width* is 4 bits and the binary value for *fieldValue* is 11110 (5 bits), the most significant bit of *fieldValue* is truncated and a binary value of 1110 is used.

Example

```
testResult = bit.setfield(15, 2, 3, 5)
print(testResult)
```

The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at *index* position 2, the returned *value* is decimal 11 (binary 1011).
Output:

```
1.10000e+01
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.get() (on page 7-12)
bit.set() (on page 7-14)
bit.getfield() (on page 7-13)

bit.test()

This function returns the Boolean value (`true` or `false`) of a bit at the specified index position.

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

Usage

```
result = bit.test(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within value to test (1 to 32)

Details

This function returns *result*, which is the result of the tested bit.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for *value* is 0, *result* is `false`. If the bit of *value* at *index* is 1, the returned value is `true`.

If *index* is bigger than the number of bits in *value*, the result is `false`.

Example

```
testResult = bit.test(10, 4)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. Testing the bit at *index* position 4 returns a Boolean value of `true`.

Output:

```
true
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.clear() (on page 7-11)
bit.get() (on page 7-12)
bit.set() (on page 7-14)
bit.toggle() (on page 7-17)

bit.toggle()

This function toggles the value of a bit at a specified index position.

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

Usage

```
result = bit.toggle(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to toggle (1 to 32)

Details

This function returns *result*, which is the result of toggling the bit *index* in *value*.

Any fractional part of *value* is truncated to make it an integer. The returned value is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

The indexed bit for *value* is toggled from 0 to 1, or 1 to 0.

Example

```
testResult = bit.toggle(10, 3)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. Toggling the bit at *index* position 3 returns a decimal value of 14 (binary 1110).
Output:

```
1.40000e+01
```

Also see

Bit manipulation and logic operations (on page 5-4)
bit.clear() (on page 7-11)
bit.get() (on page 7-12)
bit.set() (on page 7-14)
bit.test() (on page 7-16)

bufferVar.appendmode

This attribute sets the state of the append mode of the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See **Details**	0 (disabled)

Usage

```
state = bufferVar.appendmode
bufferVar.appendmode = state
```

state	The reading buffer append mode; set to one of the following: ■ `0`: Append mode off; new measurement data overwrites the previous buffer content ■ `1`: Append mode on; appends new measurement data to the present buffer content
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute enables or disables the buffer append mode. This value can only be changed with an empty buffer. Use *bufferVar*`.clear()` to empty the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

If the append mode is set to `0`, any stored readings in the buffer are cleared before new ones are stored. If append mode is set to `1`, any stored readings remain in the buffer and new readings are added to the buffer after the stored readings.

With append mode on, the first new measurement is stored at `rb[`*n*`+1]`, where *n* is the number of readings stored in buffer `rb`.

Example

`buffer1.appendmode = 1`	Append new readings to contents of the reading buffer named `buffer1`.

Also see

bufferVar.clear() (on page 7-22)
Reading buffers (on page 3-6)

bufferVar.basetimestamp

This attribute contains the timestamp that indicates when the first reading was stored in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	See **Details**	0

Usage

```
basetime = bufferVar.basetimestamp
```

basetime	The timestamp of the first stored reading
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

Details

This read-only attribute contains the timestamp (in seconds) of the first reading stored in a buffer (`rb[1]` stored in reading buffer `rb`). The timestamp is the number of seconds since 12:00 am January 1, 1970 (UTC) that the measurement was performed and stored.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

See the `smua.nvbufferY` attribute for details on accessing dedicated reading buffers.

Example

```
basetime = smua.nvbuffer1.basetimestamp
print(basetime)
```

Read the timestamp for the first reading stored in dedicated reading buffer 1.
Output:

```
1.57020e+09
```

This output indicates that the timestamp is 1,570,200,000 seconds (which is Friday, October 4, 2019 at 14:40:00 pm).

Also see

Reading buffers (on page 3-6)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.Y() (on page 7-298)

bufferVar.cachemode

This attribute enables or disables the reading buffer cache (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not saved	1 (enabled)

Usage

```
cacheMode = bufferVar.cachemode
bufferVar.cachemode = cacheMode
```

cacheMode	The reading buffer cache mode; set to one of the following: ■ `0`: Cache mode disabled (off) ■ `1`: Cache mode enabled (on)
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute enables or disables the reading buffer cache. When enabled, the reading buffer cache improves access speed to reading buffer data.

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when initiating successive sweeps without reconfiguring the sweep measurements or when overwriting data in the reading buffer by setting the *bufferVar*`.fillmode` attribute to `smu`*X*`.FILL_WINDOW`. To avoid this, make sure that you include commands that automatically invalidate the cache as needed (for example, explicit calls to the *bufferVar*`.clearcache()` function) or disable the cache using this attribute (*bufferVar*`.cachemode`).

Example

```
smua.nvbuffer1.cachemode = 1
```

Enables reading buffer cache of dedicated reading buffer 1.

Also see

bufferVar.clearcache() (on page 7-22)
bufferVar.fillmode (on page 7-26)
Reading buffers (on page 3-6)

bufferVar.capacity

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

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

Usage

```
bufferCapacity = bufferVar.capacity
```

bufferCapacity	The maximum number of readings the buffer can store
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This read-only attribute reads the number of readings that can be stored in the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

The capacity of the buffer does not change as readings fill the buffer. A dedicated reading buffer that only collects basic items can store over 140,000 readings. Turning on additional collection items, such as timestamps and source values, decreases the capacity of a dedicated reading buffer (for example, `smua.nvbuffer1`), but does not change the capacity of a user-defined dynamically allocated buffer. A user-defined dynamically allocated buffer has a fixed capacity that is set when the buffer is created.

See the `smua.nvbufferY` attribute for details on accessing dedicated reading buffers. See the `smua.makebuffer()` function for information on creating user-defined dynamically allocated reading buffers.

Example

```
bufferCapacity = smua.nvbuffer1.capacity

print(bufferCapacity)
```

Reads the capacity of dedicated reading buffer 1.
Output:
`1.49789e+05`
The above output indicates that the buffer can hold 149789 readings.

Also see

Reading buffers (on page 3-6)
smuX.makebuffer() (on page 7-243)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.Y() (on page 7-298)

bufferVar.clear()

This function empties the buffer.

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

Usage

```
bufferVar.clear()
```

bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This function clears all readings and related recall attributes from the buffer (for example, *bufferVar*.timestamps and *bufferVar*.statuses) from the specified buffer.

Example

```
smua.nvbuffer1.clear()
```
Clears dedicated reading buffer 1.

Also see

Reading buffers (on page 3-6)
smuX.nvbufferY (on page 7-263)

bufferVar.clearcache()

This function clears the cache.

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

Usage

```
bufferVar.clearcache()
```

bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This function clears all readings from the specified cache.

- If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you:
- Initiate successive sweeps without reconfiguring the sweep measurements. Watch for this when running Lua code remotely on more than one node, because values in the reading buffer cache may change while the Lua code is running.
- Overwrite data in the reading buffer by setting the *bufferVar*.fillmode attribute to smua.FILL_WINDOW.

To avoid this, you can include explicit calls to the *bufferVar*.clearcache() function to remove stale values from the reading buffer cache.

Example

`smua.nvbuffer1.clearcache()`	Clears the reading buffer cache for dedicated reading buffer 1.

Also see

bufferVar.cachemode (on page 7-20)
bufferVar.fillmode (on page 7-26)
Reading buffers (on page 3-12, on page 3-6)
Removing stale values from the reading buffer cache (on page 6-67)
smuX.nvbufferY (on page 7-263)

bufferVar.collectsourcevalues

This attribute sets whether or not source values are stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See **Details**	0 (disabled)

Usage

```
state = bufferVar.collectsourcevalues
bufferVar.collectsourcevalues = state
```

state	Source value collection status; set to one of the following: ■ 0: Source value collection disabled (off) ■ 1: Source value collection enabled (on)
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as smua.nvbuffer1)

Details

Assigning a value to this attribute enables or disables the storage of source values. Reading this attribute returns the state of source value collection. This value can only be changed with an empty buffer. Empty the buffer using the *bufferVar*.clear() function.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

When on, source values are stored with readings in the buffer. This requires four extra bytes of storage for each reading. Turning on additional collection items, such as source values (this attribute) and timestamps, decreases the capacity of a dedicated reading buffer, but does not change the capacity of a user-defined dynamically allocated buffer.

You cannot collect source values when smua.trigger.measure.action is set to smua.ASYNC, so *bufferVar*.collectsourcevalues must be set to 0 when the measurement action is set to be asynchronous.

Example

`smua.nvbuffer1.collectsourcevalues = 1`	Include source values with readings for dedicated reading buffer 1.

Also see

bufferVar.clear() (on page 7-22)
Reading buffers (on page 3-6)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.action (on page 7-295)
smuX.trigger.measure.Y() (on page 7-298)

bufferVar.collecttimestamps

This attribute sets whether or not timestamp values are stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See **Details**	0 (disabled)

Usage

```
state = bufferVar.collecttimestamps
bufferVar.collecttimestamps = state
```

state	Timestamp value collection status; set to one of the following: ■ `0`: Timestamp value collection disabled (off) ■ `1`: Timestamp value collection enabled (on)
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute enables or disables the storage of timestamps. Reading this attribute returns the state of timestamp collection.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

When on, timestamp values are stored with readings in the buffer. This requires four extra bytes of storage for each reading. When you turn on additional collection items, such as timestamps (this attribute) and source values, it decreases the capacity of a dedicated reading buffer (for example, `smua.nvbuffer1`), but does not change the capacity of a user-defined dynamically allocated buffer.

The *state* variable can only be changed when the buffer is empty. Empty the buffer using the *bufferVar*`.clear()` function.

Example

`smua.nvbuffer1.collecttimestamps = 1`	Include timestamps with readings for dedicated reading buffer 1.

Also see

bufferVar.clear() (on page 7-22)
Reading buffers (on page 3-6)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.Y() (on page 7-298)

bufferVar.fillcount

This attribute sets the reading buffer fill count.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See **Details**	0

Usage

```
fillCount = bufferVar.fillcount
bufferVar.fillcount = fillCount
```

fillCount	The reading buffer fill count
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

Details

The reading buffer fill count sets the number of readings to store before restarting at index 1. If the value is zero (0), then the capacity of the buffer is used. Use this attribute to control when the SMU restarts filling the buffer at index 1, rather than having it restart when the buffer is full.

If the *bufferVar*`.fillcount` attribute is set to a value higher than the capacity of the buffer, after storing the element at the end of the buffer, the SMU overwrites the reading at index 1, the reading after that overwrites the reading at index 2, and so on.

This attribute is only used when the *bufferVar*`.fillmode` attribute is set to `smu`*X*`.FILL_WINDOW`.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

`smua.nvbuffer1.fillcount = 50`	Sets fill count of dedicated reading buffer 1 to 50.

Also see

bufferVar.fillmode (on page 7-26)

bufferVar.fillmode

This attribute sets the reading buffer fill mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See **Details**	0 (smu*X*.FILL_ONCE)

Usage

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

fillMode	The reading buffer fill mode; set to one of the following: ■ `0` or `smuX.FILL_ONCE`: Do not overwrite old data ■ `1` or `smuX.FILL_WINDOW`: New readings restart at index 1 after acquiring reading at index *bufferVar*`.fillcount`
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)

Details

When this attribute is set to `smuX.FILL_ONCE`, the reading buffer does not overwrite readings. If the buffer fills up, new readings are discarded.

When this attribute is set to `smuX.FILL_WINDOW`, new readings are added after existing data until the buffer holds *bufferVar*`.fillcount` elements. Continuing the sequence, the next reading overwrites the reading at index 1, the reading after that overwrites the reading at index 2, and so on.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
smua.nvbuffer1.fillmode = smua.FILL_ONCE
```

Sets fill mode of dedicated reading buffer 1 to fill once (do not overwrite old data).

Also see

bufferVar.fillcount (on page 7-25)
Reading buffers (on page 3-6)

bufferVar.measurefunctions

This attribute contains the measurement function that was used to acquire a reading stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
measurefunction = bufferVar.measurefunctions[N]
```

measurefunction	The measurement function used (`Current`, `Voltage`, `Ohms`, or `Watts`) to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)
N	The reading number (`1` to *bufferVar*`.n`)

Details

The `measurefunctions` buffer recall attribute is like an array (a Lua table) of strings indicating the function measured for the reading.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
measurefunction = smua.nvbuffer1.measurefunctions[5]
```

Store the measure function used to make reading number 5.

Example 2

```
printbuffer(1, 5, smua.nvbuffer1.measurefunctions)
```

Print the measurement function that was used to measure the first five readings saved in dedicated reading buffer 1.
Example output:

```
Current, Current, Current, Current, Current
```

Also see

bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.measureranges

This attribute contains the measurement range values that were used for readings stored in a specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
measurerange = bufferVar.measureranges[N]
```

measurerange	The measurement range used to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)
N	The reading number (`1` to *bufferVar*`.n`)

Details

The `measureranges` buffer recall attribute is like an array (a Lua table) of full-scale range values for the measure range used when the measurement was made.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
measurerange = smua.nvbuffer1.measureranges[1]
```

Store the measure range that was used to make reading number 1.

Example 2

```
printbuffer(1, 10, smua.nvbuffer1.measureranges)
```

Print the range values that were used for the first 10 readings saved in dedicated reading buffer 1.
Example output:

```
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07,
1.00000e-07, 1.00000e-07
```

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.n

This attribute contains the number of readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
numberOfReadings = bufferVar.n
```

numberOfReadings	The number of readings stored in the buffer
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

This read-only attribute contains the number of readings presently stored in the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

```
numberOfReadings = smua.nvbuffer1.n

print(numberOfReadings)
```

Reads the number of readings stored in dedicated reading buffer 1.
Output:
`1.25000+02`
The above output indicates that there are 125 readings stored in the buffer.

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.Y() (on page 7-298)

bufferVar.readings

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

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

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

reading	The value of the reading in the specified reading buffer
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

The *bufferVar*.readings buffer recall attribute is like an array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly; that is, rb[2] and rb.readings[2] access the same value.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

`print(smua.nvbuffer1.readings[1])`	Output the first reading saved in dedicated reading buffer 1. Output: `8.81658e-08`

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.sourcefunctions

This attribute contains the source function that was being used when the readings were stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
sourcefunction = bufferVar.sourcefunctions[N]
```

sourcefunction	The source function used (`Current` or `Voltage`) to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)
N	The reading number (`1` to *bufferVar*`.n`)

Details

The *bufferVar*`.sourcefunctions` buffer recall attribute is like an array (a Lua table) of strings indicating the source function at the time of the measurement.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
sourcefunction = smua.nvbuffer1.sourcefunctions[3]
print(sourcefunction)
```

Store the source function used to make reading number 3 and output the value.

Example 2

```
printbuffer(1, 10, smua.nvbuffer1.sourcefunctions)
```

Print the source function used for 10 readings stored in dedicated reading buffer 1.
Example output:

```
Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage
```

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.sourceoutputstates

This attribute indicates the state of the source output for readings that are stored in a specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
state = bufferVar.sourceoutputstates[N]
```

state	The output state (`Off` or `On`) when reading *N* of the specified buffer was acquired
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)
N	The reading number (`1` to *bufferVar*`.n`)

Details

The *bufferVar*`.sourceoutputstates` buffer recall attribute is similar to an array (a Lua table) of strings. This array indicates the state of the source output (`Off` or `On`) at the time of the measurement.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

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

Print the source output for the first reading stored in dedicated reading buffer 1.
Example output:

```
On
```

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.sourceranges

This attribute contains the source range that was used for readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
sourcerange = bufferVar.sourceranges[N]
```

sourcerange	The source range used to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined), or a dedicated reading buffer (such as `smua.nvbuffer1`)
N	The reading number (`1` to *bufferVar*`.n`)

Details

The *bufferVar*`.sourceranges` buffer recall attribute is like an array (a Lua table) of full-scale range values for the source range used when the measurement was made.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

```
sourcerange = smua.nvbuffer1.sourceranges[1]
```

Store the source range that was used for the first reading stored in dedicated reading buffer 1.

Example 2

```
printbuffer(1, 6, smua.nvbuffer1.sourceranges)
```

Print the source ranges that were used for the first 6 readings stored in source-measure unit (SMU) A, buffer 1.
Example output:

```
1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04
```

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.sourcevalues

When enabled by the *bufferVar*.collectsourcevalues attribute, this attribute contains the source levels being output when readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

```
sourcevalue = bufferVar.sourcevalues[N]
```

sourcevalue	The output value of the source when reading *N* of the specified buffer was acquired
bufferVar	The reading buffer; can be a dynamically allocated buffer (user-defined) or a dedicated reading buffer (such as smua.nvbuffer1)
N	The reading number (1 to *bufferVar*.n)

Details

If the *bufferVar*.collectsourcevalues attribute is enabled before readings are made, the *bufferVar*.sourcevalues buffer recall attribute is like an array (a Lua table) of the sourced value in effect at the time of the reading.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example 1

`sourcevalue = smua.nvbuffer1.sourcevalues[1]`	Get the sourced value of the first reading stored in dedicated reading buffer 1.

Example 2

`printbuffer(1, 6, smua.nvbuffer1.sourcevalues)`	Print the sourced value of the first 6 readings stored in source-measure unit (SMU) A, buffer 1. Example output: `1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04, 1.00000e-04`

Also see

bufferVar.collectsourcevalues (on page 7-23)
bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.statuses (on page 7-35)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.statuses

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

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

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

statusInformation	The status value when reading *N* of the specified buffer was acquired
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer
N	The reading number *N*; can be any value from 1 to the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

This read-only buffer recall attribute is like an array (a Lua table) of the status values for all the readings in the buffer. The status values are floating-point numbers that encode the status value; see the following table for values.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Buffer status bits

Bit	Name	Hex value	Description
B0	FastADC	0x01	Fast ADC was used to make the reading
B1	Overtemp	0x02	Over temperature condition
B2	AutoRangeMeas	0x04	Measure range was autoranged
B3	AutoRangeSrc	0x08	Source range was autoranged
B4	4Wire	0x10	4-wire (remote) sense mode enabled
B5	Rel	0x20	Relative offset applied to reading
B6	Compliance	0x40	Source function was limited because the complementary function would be over the compliance limit
B7	Filtered	0x80	Reading was filtered

Also see

bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceranges (on page 7-33)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)

bufferVar.timestampresolution

This attribute contains the resolution of the timestamp.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	See **Details**	1e-6 (1 µs)

Usage

```
resolution = bufferVar.timestampresolution
```

resolution	Timestamp resolution in seconds (minimum 1 µs; rounded to an even power of 2 µs)
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer

Details

Assigning a value to this attribute sets the resolution for the timestamps. Reading this attribute returns the timestamp resolution value. This value can only be changed with an empty buffer. Empty the buffer using the *bufferVar*.clear() function.

The finest timestamp resolution is 0.000001 seconds (1 µs). At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. You can increase this value for very long tests.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

`smua.nvbuffer1.timestampresolution = 0.000008`	Sets the timestamp resolution of dedicated reading buffer 1 to 8 µs.

Also see

bufferVar.clear() (on page 7-22)
bufferVar.collecttimestamps (on page 7-24)
bufferVar.timestamps (on page 7-37)
Reading buffers (on page 3-6)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.Y() (on page 7-298)

bufferVar.timestamps

When enabled by the *bufferVar*.collecttimestamps attribute, this attribute contains the timestamp when each reading saved in the specified reading buffer occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer	See **Details**	Not applicable

Usage

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

timestamp	The complete timestamp (including date, time, and fractional seconds) of reading number *N* in the specified reading buffer when the reading was acquired
bufferVar	The reading buffer; can be a dynamically allocated user-defined buffer or a dedicated reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The *bufferVar*.timestamps information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you cannot access any time information from a reading buffer.

If enabled, this buffer recall attribute is like an array (a Lua table) that contains timestamps, in seconds, of when each reading occurred. These are relative to the *bufferVar*.basetimestamp for the buffer.

For dedicated reading buffers, all buffer attributes are saved to nonvolatile memory only when the reading buffer is saved to nonvolatile memory.

Example

`timestamp = smua.nvbuffer1.timestamps[1]`	Get the timestamp of the first reading stored in dedicated reading buffer 1.

Also see

bufferVar.clear() (on page 7-22)
bufferVar.collecttimestamps (on page 7-24)
bufferVar.measurefunctions (on page **Error! Bookmark not defined.**)
bufferVar.measureranges (on page 7-28)
bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
bufferVar.sourcefunctions (on page 7-31)
bufferVar.sourceoutputstates (on page 7-32)
bufferVar.sourceranges (on page 7-33)
bufferVar.sourcevalues (on page 7-34)
bufferVar.statuses (on page 7-35)
Reading buffers (on page 3-6)

ConfigPulseIMeasureV()

This KIPulse factory script function configures a current pulse train with a voltage measurement at each point.

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

Usage

```
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in, sync_out)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag,
    sync_in)
f, msg = ConfigPulseIMeasureV(smu, bias, level, limit, ton, toff, points, buffer, tag)
```

f	A Boolean flag; this flag is `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string that indicates successful configuration
smu	Instrument channel (set to `smua`)
bias	Bias level in amperes
level	Pulse level in amperes
limit	Voltage limit (for example, compliance) in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements are stored; if this is `nil` when the function is called, no measurements are made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of `ton`
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	Specifies whether or not to abort the pulse if an input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; `true` or `false`

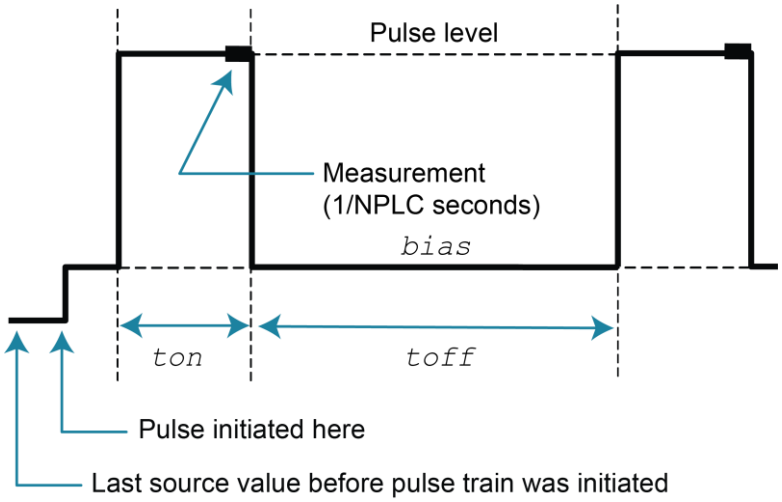
Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a current pulse train with a voltage measurement at each point. Measurements are made at the end of the *ton* time.

This function does not cause the specified *smu* to output a pulse train. It simply checks to see if all the pulse dimensions can be achieved, and if they are, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid *tag*.

Figure 112: ConfigPulseIMeasureV()



Example

```
ConfigPulseIMeasureV(smua, 0, 5, 10,
   0.001, 0.080, 1, smua.nvbuffer1, 1)
```

Set up a pulse train that uses SMU channel A. The pulse amplitude is 5 A and returns to 0 A after 1 ms. The pulse remains at 0 A for 80 ms and the voltage limit is 10 V during the pulse. The pulse train consists of only 1 pulse, and this pulse is assigned a *tag* index of 1.

Also see

InitiatePulseTest() (on page 7-128)
KIPulse factory script (on page 5-23)

ConfigPulseIMeasureVSweepLin()

This KIPulse factory script function configures a linear pulsed current sweep with a voltage measurement at each point.

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

Usage

```
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag, sync_in)
f, msg = ConfigPulseIMeasureVSweepLin(smu, bias, start, stop, limit, ton, toff,
    points, buffer, tag)
```

f	A Boolean flag; this flag is `true` if the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string indicating successful configuration
smu	Instrument channel (set to `smua`)
bias	Bias level in amperes
start	Pulse sweep start level in amperes
stop	Pulse sweep stop level in amperes
limit	Voltage limit (for example, compliance) in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements are stored; if this is `nil` when the function is called, no measurements are made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of `ton`
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; `true` or `false`

Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the *buffer* input parameter.

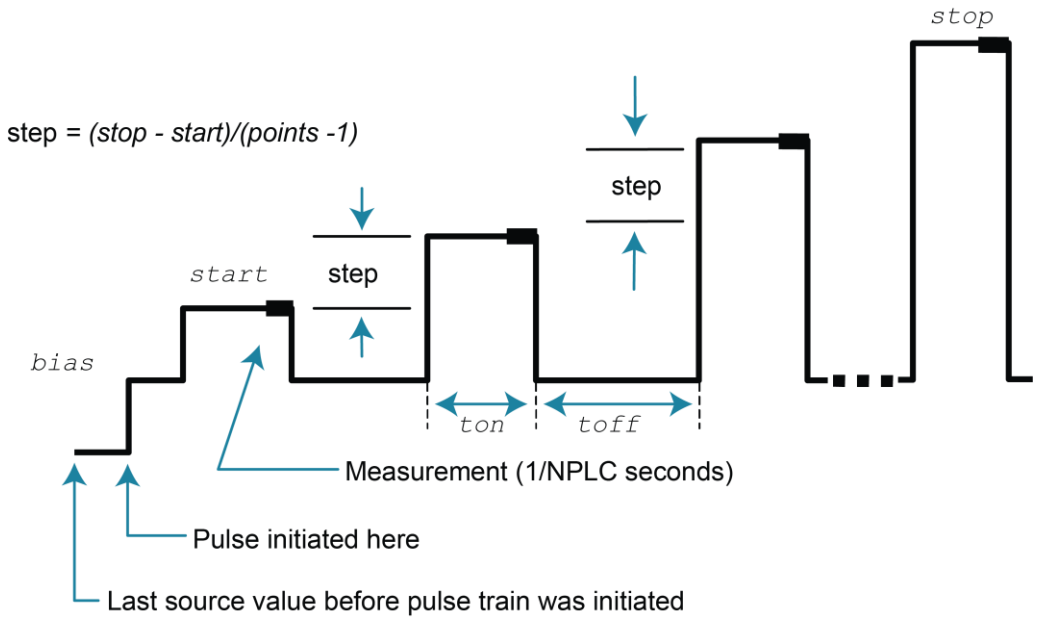
This function configures a linear pulsed current sweep with a voltage measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse is *start* amperes; the magnitude of the last pulse is *stop* amperes. The magnitude of each pulse in between is *step* amperes larger than the previous pulse, where:

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

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(tag)` function is used to initiate a pulse train assigned to a valid *tag*.

Figure 113: ConfigPulseIMeasureVSweepLin()



Example

`ConfigPulseIMeasureVSweepLin(smua, 0, 0.01, 0.05, 1, 1e-3, 0.1, 20, smua.nvbuffer2, 3)`	Set up a pulsed sweep that uses SMU channel A. The pulsed sweep starts at 10 mA, end at 50 mA, and returns to a 0 mA bias level between pulses. Each pulsed step is on for 1 ms and then at the bias level for 100 ms. The voltage limit is 1 V during the entire pulsed sweep. The pulse train is comprised of 20 pulsed steps and the pulse train is assigned a *tag* index of 3.

Also see

InitiatePulseTest() (on page 7-128)
KIPulse factory script (on page 5-23)

ConfigPulseIMeasureVSweepLog()

This KIPulse factory script function configures a voltage pulse train with a current measurement at each point.

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

Usage

```
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff,
   points, buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff,
   points, buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff,
   points, buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff,
   points, buffer, tag, sync_in)
f, msg = ConfigPulseIMeasureVSweepLog(smu, bias, start, stop, limit, ton, toff,
   points, buffer, tag)
```

f	A Boolean flag; this flag is `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string indicating successful configuration
smu	Instrument channel (set to `smua`)
bias	Bias level in amperes
start	Pulse sweep start level in amperes
stop	Pulse sweep stop level in amperes
limit	Voltage limit (for example, compliance) in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of *ton*
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; `true` or `false`

Details

Data for pulsed voltage measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a logarithmic pulsed current sweep with a voltage measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse will be *start* amperes; the magnitude of the last pulse will be *stop* amperes. The magnitude of each pulse in between will be $LogStep_n$ amperes larger than the previous pulse, where:

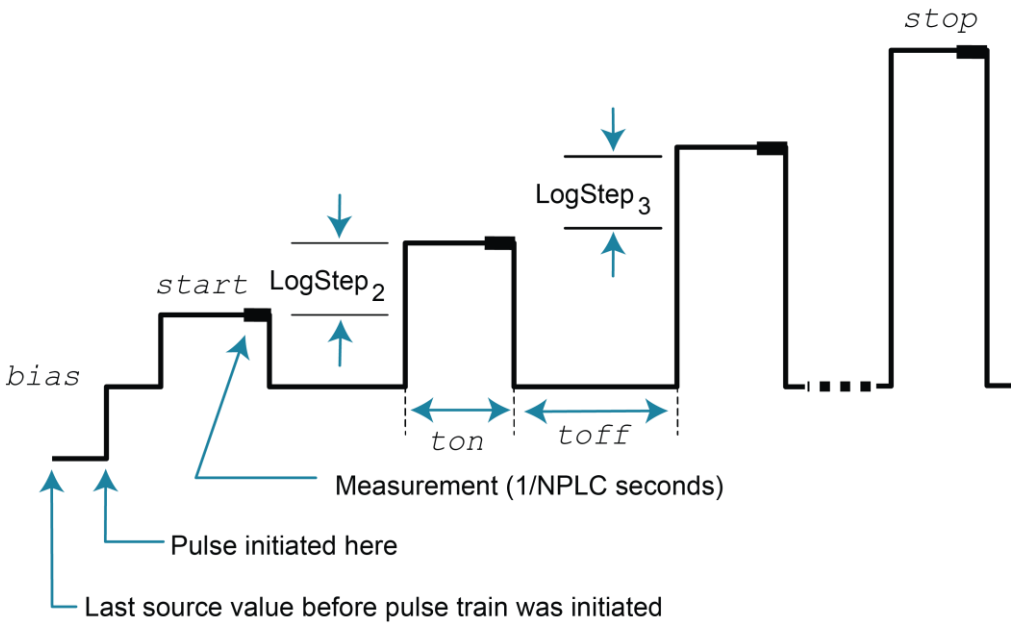
LogStepSize = (log10(stop) - log10(start)) / (points -1)

*LogStepn = (n - 1) * (LogStepSize), where* n = [1, points]

*SourceStepLeveln = antilog(LogStepn) * start*

This function does not cause the specified *smu* to output a pulse train. It simply checks to see if all of the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train.

Figure 114: ConfigPulseIMeasureVSweepLog()



Example

`ConfigPulseIMeasureVSweepLog(smua, 0, 1e-3, 0.01, 100, 10e-3, 100e-3, 10, smua.nvbuffer1, 5)`	Set up a pulsed logarithmic sweep that uses System SourceMeter® instrument channel A. The pulsed sweep will start at 1 mA, end at 10 mA, and return to a 0 A bias level between pulses. Each pulsed step will be on for 10 ms, and then at the bias level for 100 ms. The voltage limit will be 100 V during the entire pulsed sweep. The pulse train will be comprised of 10 pulsed steps, and the pulse train will be assigned a *tag* index of 5.

Also see

InitiatePulseTest() (on page 7-128)
KIPulse factory script (on page 5-23)

ConfigPulseVMeasureI()

This KIPulse factory script function configures a voltage pulse train with a current measurement at each point.

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

Usage

```
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer,
   tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer,
   tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer,
   tag, sync_in, sync_out)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer,
   tag, sync_in)
f, msg = ConfigPulseVMeasureI(smu, bias, level, limit, ton, toff, points, buffer,
   tag)
```

f	A Boolean flag; this flag is `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string indicating successful configuration
smu	Instrument channel (set to `smua`)
bias	Bias level in volts
level	Pulse level in volts
limit	Current limit (for example, compliance) in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements will be stored; if this is `nil` when the function is called, no measurements will be made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of `ton`
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; `true` or `false`

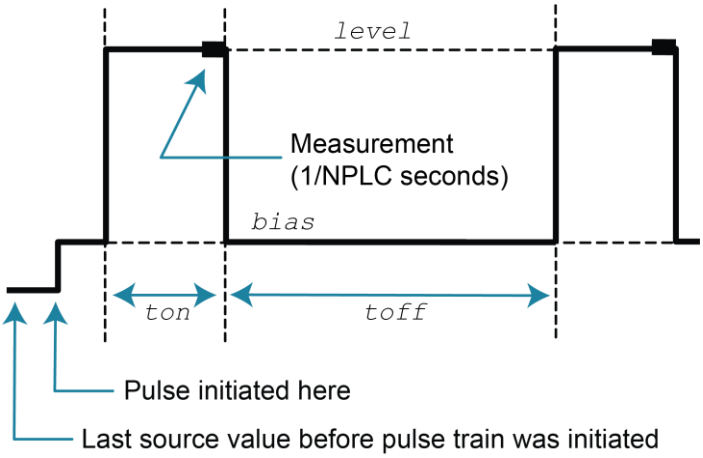
Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a voltage pulse train with a current measurement at each point. Measurements are made at the end of the *ton* time.

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid *tag*.

Figure 115: ConfigPulseVMeasureI()



Example 1

```
ConfigPulseVMeasureI(smua, 0, 20, 1,
   0.001, 0.080, 10, smua.nvbuffer1, 2)
```

Set up a pulse train that uses System SourceMeter® instrument channel A. The pulse amplitude is 20 V and returns to 0 V after 1 ms. The pulse remains at 0 V for 80 ms, and the current limit is 1 A during the pulse. The pulse train consists of 10 pulses, and the pulse train is assigned a *tag* index of 2.

Example 2

```
local timelist = { 1, 2, 3, 4, 5 }

f, msg = ConfigPulseVMeasureI(smua, 0, 1,
   100e-3, 1, timelist, 5, nil, 1)
```

Variable off time between pulses in a pulse train.
Configure a pulse with 1 second on-time and variable off-time, no measurement.

Example 3

```
rbi = smua.makebuffer(10)
rbv = smua.makebuffer(10)
rbi.appendmode = 1
rbv.appendmode = 1
rbs = { i = rbi, v = rbv }

f, msg = ConfigPulseVMeasureI(smua, 0, 10,
   1e-3, 1e-3, 1e-3, 2, rbs, 1)
```

Simultaneous IV measurement during pulse.

Also see

InitiatePulseTest() (on page 7-128)
KIPulse factory script (on page 5-23)

ConfigPulseVMeasureISweepLin()

This KIPulse factory script function configures a voltage pulse train with a current measurement at each point.

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

Usage

```
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in)
f, msg = ConfigPulseVMeasureISweepLin(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag)
```

f	A Boolean flag; this flag is `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string indicating successful configuration
smu	Instrument channel (set to `smua`)
bias	Bias level in volts
start	Pulse sweep start level in volts
stop	Pulse sweep stop level in volts
limit	Current limit (for example, compliance) in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements are stored; if this is `nil` when the function is called, no measurements are made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of `ton`
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; `true` or `false`

Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a linear pulsed voltage sweep with a current measurement at each point. Measurements are made at the end of the *ton* time.

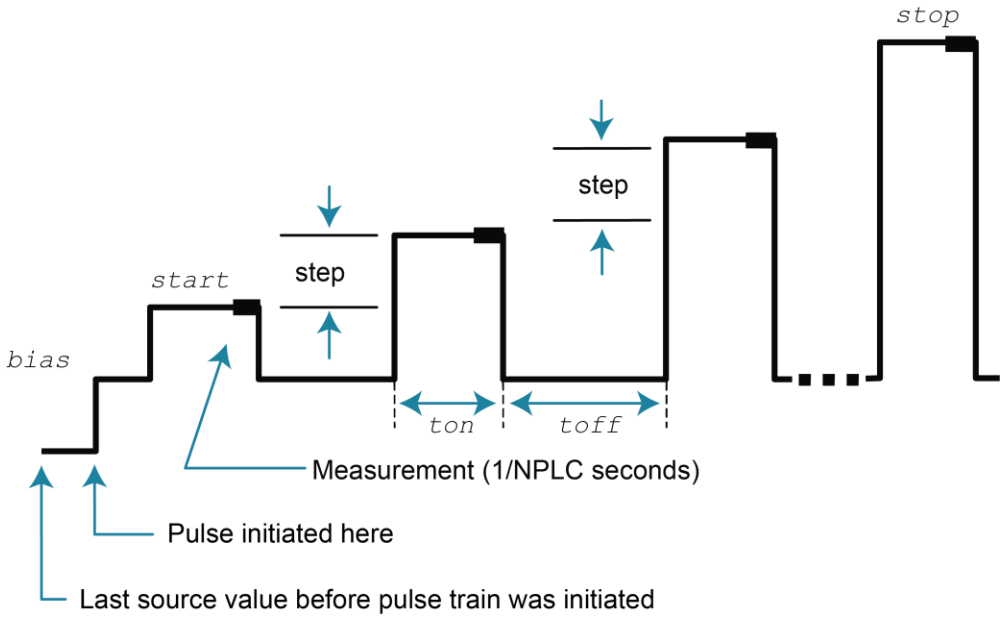
The magnitude of the first pulse is *start* volts; the magnitude of the last pulse is *stop* volts. The magnitude of each pulse in between is *step* volts larger than the previous pulse, where:

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

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train.

The `InitiatePulseTest(tag)` function is used to initiate a pulse train assigned to a valid *tag*.

Figure 116: ConfigPulseVMeasureISweepLin()



Example

```
ConfigPulseVMeasureISweepLin(smua, 0, 1,
   10, 1, 10e-3, 20e-3, 16,
   smua.nvbuffer1, 4)
```

Set up a pulsed sweep that uses SMU channel A. The pulsed sweep starts at 1 V, ends at 10 V, and returns to a 0 V bias level between pulses. Each pulsed step is on for 10 ms, and then at the bias level for 20 ms.

The current limit is 1 A during the entire pulsed sweep. The pulse train is comprised of 16 pulsed steps, and the pulse train is assigned a *tag* index of 4.

Also see

InitiatePulseTest() (on page 7-128)
KIPulse factory script (on page 5-23)

ConfigPulseVMeasureISweepLog()

This KIPulse factory script (on page 5-23) function configures a voltage pulse train with a current measurement at each point.

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

Usage

```
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout, sync_in_abort)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out, sync_in_timeout)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in, sync_out)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag, sync_in)
f, msg = ConfigPulseVMeasureISweepLog(smu, bias, start, stop, limit, ton, toff, points,
    buffer, tag)
```

f	A Boolean flag; this flag is `true` when the pulse was successfully configured, `false` when errors were encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string indicating successful configuration
smu	Instrument channel (set to `smua`)
bias	Bias level in volts
start	Pulse sweep start level in volts
stop	Pulse sweep stop level in volts
limit	Current limit (for example, compliance) in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles
buffer	Reading buffer where pulsed measurements are stored; if this is `nil` when the function is called, no measurements are made when the pulse train is initiated
tag	Numeric identifier to be assigned to the defined pulse train
sync_in	Defines a digital I/O trigger input line; if programmed, the pulse train waits for a trigger input before executing each pulse
sync_out	Defines a digital I/O trigger output line; if programmed, the pulse train generates a trigger output immediately before the start of `ton`
sync_in_timeout	Specifies the length of time (in seconds) to wait for input trigger; default value is 10 s
sync_in_abort	Specifies whether or not to abort pulse if input trigger is not received; if pulse aborts because of a missed trigger, a timer timeout message is returned; `true` or `false`

Details

Data for pulsed current measurements are stored in the reading buffer specified by the *buffer* input parameter.

This function configures a logarithmic pulsed voltage sweep with a current measurement at each point. Measurements are made at the end of the *ton* time.

The magnitude of the first pulse is *start* volts; the magnitude of the last pulse is *stop* volts. The magnitude of each pulse in between is $\texttt{LogStep}_n$ volts larger than the previous pulse, where:

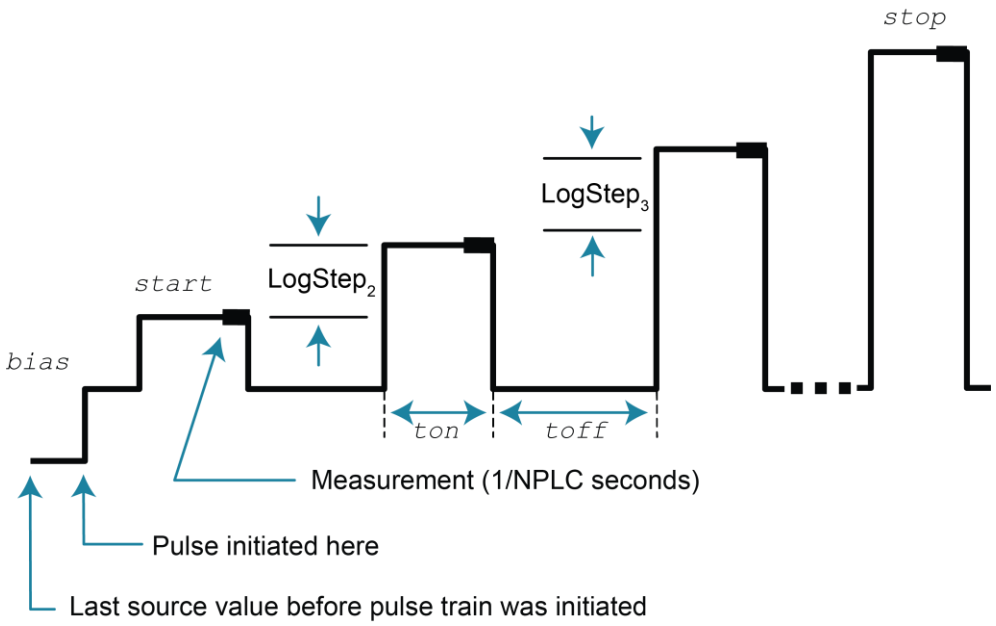
LogStepSize = (log10(stop) - log10(start)) / (points -1)

*LogStep$_n$ = (n - 1) * (LogStepSize), where n = [2, points]*

*SourceStepLevel$_n$ = antilog(LogStep$_n$) * start*

This function does not cause the specified *smu* to output a pulse train. It does check to see if all the pulse dimensions can be achieved, and if they can, assigns the indicated *tag* or index to the pulse train. The `InitiatePulseTest(`*tag*`)` function is used to initiate a pulse train assigned to a valid *tag*.

Figure 117: ConfigPulseVMeasureISweepLog()



Example

`ConfigPulseVMeasureISweepLog(smua, 0, 1, 10, 1, 10e-3, 20e-3, 10, smua.nvbuffer1, 6)`	Set up a pulsed logarithmic sweep that uses SMU channel A. The pulsed sweep starts at 1 V, ends at 10 V, and returns to a 0 V bias level between pulses. Each pulsed step is on for 10 ms, and then at the bias level for 20 ms. The current limit is1 A during the entire pulsed sweep. The pulse train is comprised of 10 pulsed steps, and the pulse train is assigned a *tag* index of 6.

Also see

InitiatePulseTest() (on page 7-128)
KIPulse factory script (on page 5-23)

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 7-52)
dataqueue.clear() (on page 7-53)
dataqueue.count (on page 7-54)
dataqueue.next() (on page 7-55)
Using the data queue for real-time communication (on page 6-67)

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 7-51)
dataqueue.clear() (on page 7-53)
dataqueue.count (on page 7-54)
dataqueue.next() (on page 7-55)
Using the data queue for real-time communication (on page 6-67)

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 7-51)
dataqueue.CAPACITY (on page 7-52)
dataqueue.count (on page 7-54)
dataqueue.next() (on page 7-55)
Using the data queue for real-time communication (on page 6-67)

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 7-51)
dataqueue.CAPACITY (on page 7-52)
dataqueue.clear() (on page 7-53)
dataqueue.next() (on page 7-55)
Using the data queue for real-time communication (on page 6-67)

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 7-51)
dataqueue.CAPACITY (on page 7-52)
dataqueue.clear() (on page 7-53)
dataqueue.count (on page 7-54)
format.asciiprecision (on page 7-111)
Using the data queue for real-time communication (on page 6-67)

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

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

Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Also see

None

digio.readbit()

This function reads one digital I/O line.

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

Usage

```
data = digio.readbit(N)
```

data	The state of the I/O line
N	Digital I/O line number to be read (1 to 14)

Details

A returned value of zero (0) indicates that the line is low. A returned value of one (1) indicates that the line is high.

Example

```
print(digio.readbit(4))
```

Assume line 4 is set high, and it is then read.
Output:

```
1.00000e+00
```

Also see

digio.readport() (on page 7-59)
digio.writebit() (on page 7-69)
digio.writeport() (on page 7-70)
Digital I/O port (on page 3-92)

digio.readport()

This function reads the digital I/O port.

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

Usage

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

data	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the I/O port. The least significant bit (bit B1) of the binary number corresponds to line 1; bit B14 corresponds to line 14.

For example, a returned value of 170 has a binary equivalent of 000000010101010, which indicates that lines 2, 4, 6, and 8 are high (1), and the other 10 lines are low (0).

Example

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

Assume lines 2, 4, 6, and 8 are set high when the I/O port is read.
Output:

```
1.70000e+02
```

This is binary 10101010.

Also see

digio.readbit() (on page 7-58)
digio.writebit() (on page 7-69)
digio.writeport() (on page 7-70)
Digital I/O port (on page 3-92)

digio.trigger[N].assert()

This function asserts a trigger pulse on one of the digital I/O lines.

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

Usage

```
digio.trigger[N].assert()
```

N	Digital I/O trigger line (1 to 14)

Details

The pulse width that is set determines how long the instrument asserts the trigger.

Example

`digio.trigger[2].assert()`	Asserts a trigger on digital I/O line 2.

Also see

digio.trigger[N].pulsewidth (on page 7-64)

digio.trigger[N].clear()

This function clears the trigger event on a digital I/O line.

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

Usage

```
digio.trigger[N].clear()
```

N	Digital I/O trigger line (1 to 14)

Details

The event detector of a trigger enters the detected state when an event is detected. It is cleared when `digio.trigger[N].wait()` or `digio.trigger[N].clear()` is called.

`digio.trigger[N].clear()` clears the event detector of the specified trigger line, discards the history of the trigger line, and clears the `digio.trigger[N].overrun` attribute.

Example

`digio.trigger[2].clear()`	Clears the trigger event detector on I/O line 2.

Also see

digio.trigger[N].overrun (on page 7-63)
digio.trigger[N].wait() (on page 7-68)

digio.trigger[N].EVENT_ID

This constant identifies the trigger event generated by the digital I/O line *N*.

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

Usage

```
eventID = digio.trigger[N].EVENT_ID
```

eventID	The trigger event number
N	Digital I/O trigger line (1 to 14)

Details

To have another trigger object respond to trigger events generated by the trigger line, set the stimulus attribute of the other object to the value of this constant.

Example1

```
digio.trigger[5].stimulus = digio.trigger[3].EVENT_ID
```

Uses a trigger event on digital I/O trigger line 3 to be the stimulus for digital I/O trigger line 5.

Example 2

```
smua.trigger.arm.stimulus = digio.trigger[3].EVENT_ID
```

Uses a trigger event on digital I/O trigger line 3 to be the trigger stimulus for the SMU.

Also see

None

digio.trigger[N].mode

This attribute sets the mode in which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Digital I/O trigger *N* reset Recall setup	Not saved	0 (digio.TRIG_BYPASS)

Usage

```
triggerMode = digio.trigger[N].mode
digio.trigger[N].mode = triggerMode
```

triggerMode	The trigger mode; see **Details** for values
N	Digital I/O trigger line (1 to 14)

Details

Set *triggerMode* to one of the following values:

Trigger mode values

triggerMode	**Description**
digio.TRIG_BYPASS or 0	Allows direct control of the line.
digio.TRIG_FALLING or 1	Detects falling-edge triggers as input; asserts a TTL-low pulse for output.
digio.TRIG_RISING or 2	If the programmed state of the line is high, the digio.TRIG_RISING mode behavior is similar to digio.TRIG_RISINGA. If the programmed state of the line is low, the digio.TRIG_RISING mode behavior is similar to digio.TRIG_RISINGM. Only use this setting if necessary for compatibility with other Keithley Instruments products.
digio.TRIG_EITHER or 3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
digio.TRIG_SYNCHRONOUSA or 4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line.
digio.TRIG_SYNCHRONOUS or 5	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
digio.TRIG_SYNCHRONOUSM or 6	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
digio.TRIG_RISINGA or 7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
digio.TRIG_RISINGM or 8	Asserts a TTL-high pulse for output. Input edge detection is not possible in this mode.

When programmed to any mode except `digio.TRIG_BYPASS`, the output state of the I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Use of either `digio.TRIG_SYNCHRONOUSA` or `digio.TRIG_SYNCHRONOUSM` is preferred over `digio.TRIG_SYNCHRONOUS`, because `digio.TRIG_SYNCHRONOUS` is provided for compatibility with the digital I/O and TSP-Link triggering on other Keithley Instruments products.

To control the line state, set the mode to `digio.TRIG_BYPASS` and use the `digio.writebit()` and `digio.writeport()` commands.

Example

```
digio.trigger[4].mode = 2
```

Sets the trigger mode for I/O line 4 to `digio.TRIG_RISING`.

Also see

digio.trigger[N].clear() (on page 7-60)
digio.trigger[N].reset() (on page 7-65)
digio.writebit() (on page 7-69)
digio.writeport() (on page 7-70)
Sweep operation (on page 3-21)

digio.trigger[N].overrun

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Digital I/O trigger *N* clear Digital I/O trigger *N* reset Recall setup	Not saved	Not applicable

Usage

```
overrun = digio.trigger[N].overrun
```

overrun	Trigger overrun state (`true` or `false`)
N	Digital I/O trigger line (1 to 14)

Details

If this is `true`, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = digio.trigger[1].overrun
print(overrun)
```

If there is no trigger overrun, the following text is output:

```
false
```

Also see

digio.trigger[N].clear() (on page 7-60)
digio.trigger[N].reset() (on page 7-65)

digio.trigger[N].pulsewidth

This attribute describes the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Digital I/O trigger *N* reset Recall setup	Not saved	10e-6 (10 µs)

Usage

```
width = digio.trigger[N].pulsewidth
digio.trigger[N].pulsewidth = width
```

width	The pulse width (seconds)
N	Digital I/O trigger line (1 to 14)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `digio.trigger[N].release()`.

Example

```
digio.trigger[4].pulsewidth = 20e-6
```

Sets the pulse width for trigger line 4 to 20 µs.

Also see

digio.trigger[N].assert() (on page 7-60)
digio.trigger[N].release() (on page 7-65)
digio.trigger[N].reset() (on page 7-65)

digio.trigger[N].release()

This function releases an indefinite length or latched trigger.

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

Usage

```
digio.trigger[N].release()
```

N	Digital I/O trigger line (1 to 14)

Details

Releases a trigger that was asserted with an indefinite pulsewidth time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Example

```
digio.trigger[4].release()
```

Releases digital I/O trigger line 4.

Also see

digio.trigger[N].assert() (on page 7-60)
digio.trigger[N].pulsewidth (on page 7-64)

digio.trigger[N].reset()

This function resets trigger values to their factory defaults.

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

Usage

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

N	Digital I/O trigger line (1 to 14)

Details

This function resets the following attributes to factory default settings:

- `digio.trigger[N].mode`
- `digio.trigger[N].pulsewidth`
- `digio.trigger[N].stimulus`

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

Example

```
digio.trigger[3].mode = 2
digio.trigger[3].pulsewidth = 50e-6
digio.trigger[3].stimulus = digio.trigger[5].EVENT_ID
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth,
    digio.trigger[3].stimulus)
digio.trigger[3].reset()
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth,
    digio.trigger[3].stimulus)
```

Set the digital I/O trigger line 3 for a falling edge with a pulsewidth of 50 µs.
Use digital I/O line 5 to trigger the event on line 3.
Reset the line back to factory default values.
Output before reset:

```
2.00000e+00         5.00000e-05         5.00000e+00
```

Output after reset:

```
0.00000e+00         1.00000e-05         0.00000e+00
```

Also see

digio.trigger[N].mode (on page 7-62)
digio.trigger[N].overrun (on page 7-63)
digio.trigger[N].pulsewidth (on page 7-64)
digio.trigger[N].stimulus (on page 7-66)

digio.trigger[N].stimulus

This attribute selects the event that causes a trigger to be asserted on the digital output line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Digital I/O trigger N reset Recall setup	Not saved	0

Usage

```
triggerStimulus = digio.trigger[N].stimulus
digio.trigger[N].stimulus = triggerStimulus
```

triggerStimulus	The event identifier for the triggering event
N	Digital I/O trigger line (1 to 14)

Details

Set this attribute to zero (0) to disable the automatic trigger output.

Do not use the stimulus attribute for generating output triggers under script control. Use `digio.trigger[N].assert()` instead.

The trigger stimulus for a digital I/O line may be set to one of the existing trigger event IDs, described in the following table.

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smu*X*.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[*N*].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[*N*].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[*N*].EVENT_ID	Occurs when a delay expires

Example 1

`digio.trigger[3].stimulus = 0`	Clear the trigger stimulus of digital I/O line 3.

Example 2

`digio.trigger[3].stimulus =` `  smua.trigger.SOURCE_COMPLETE_EVENT_ID`	Set the trigger stimulus of digital I/O line 3 to be the source complete event.

Also see

digio.trigger[N].assert() (on page 7-60)
digio.trigger[N].clear() (on page 7-60)
digio.trigger[N].reset() (on page 7-65)

digio.trigger[N].wait()

This function waits for a trigger.

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

Usage

```
triggered = digio.trigger[N].wait(timeout)
```

triggered	The value is `true` if a trigger is detected, or `false` if no triggers are detected during the timeout period
N	Digital I/O trigger line (1 to 14)
timeout	Timeout in seconds

Details

This function pauses trigger operation up to the seconds set by *timeout* for an input trigger. If one or more trigger events are detected since the last time `digio.trigger[N].wait()` or `digio.trigger[N].clear()` was called, this function returns a value immediately. After waiting for a trigger with this function, the event detector is automatically reset and ready to detect the next trigger. This is true regardless of the number of events detected.

Example

```
triggered = digio.trigger[4].wait(3)
print(triggered)
```

Waits up to three seconds for a trigger to be detected on trigger line 4, then outputs the results.
Output if no trigger is detected:
`false`
Output if a trigger is detected:
`true`

Also see

digio.trigger[N].clear() (on page 7-60)

digio.writebit()

This function sets a digital I/O line high or low.

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

Usage

```
digio.writebit(N, data)
```

N	Digital I/O trigger line (1 to 14)
data	The value to write to the bit: ■ 0 (low) ■ Non-zero (high)

Details

If the output line is write-protected using the `digio.writeprotect` attribute, the command is ignored.

The `reset()` function does not affect the present state of the digital I/O lines.

Use the `digio.writebit()` and `digio.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `digio.TRIG_BYPASS`.

The data must be zero (0) to clear the bit. Any value other than zero (0) sets the bit.

Example

`digio.writebit(4, 0)`	Sets digital I/O line 4 low (0).

Also see

digio.readbit() (on page 7-58)
digio.readport() (on page 7-59)
digio.trigger[N].mode (on page 7-62)
digio.writeport() (on page 7-70)
digio.writeprotect (on page 7-71)

digio.writeport()

This function writes to all digital I/O lines.

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

Usage

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

data	Value to write to the port (0 to 16383)

Details

The binary representation of *data* indicates the output pattern to be written to the I/O port. For example, a *data* value of 170 has a binary equivalent of 00000010101010. Lines 2, 4, 6, and 8 are set high (1), and the other 10 lines are set low (0).

Write-protected lines are not changed.

The `reset()` function does not affect the present states of the digital I/O lines.

Use the `digio.writebit()` and `digio.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `digio.TRIG_BYPASS`.

Example

```
digio.writeport(255)
```

Sets digital I/O Lines 1 through 8 high (binary 00000011111111).

Also see

digio.readbit() (on page 7-58)
digio.readport() (on page 7-59)
digio.writebit() (on page 7-69)
digio.writeprotect (on page 7-71)

digio.writeprotect

This attribute contains the write-protect mask that protects bits from changes from the `digio.writebit()` and `digio.writeport()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0

Usage

```
mask = digio.writeprotect
digio.writeprotect = mask
```

mask	Sets the value that specifies the bit pattern for write-protect

Details

Bits that are set to one cause the corresponding line to be write-protected.

The binary equivalent of *mask* indicates the mask to be set for the I/O port. For example, a mask value of 7 has a binary equivalent of 00000000000111. This mask write-protects lines 1, 2, and 3.

Example

```
digio.writeprotect = 15
```

Write-protects lines 1, 2, 3, and 4.

Also see

digio.writebit() (on page 7-69)
digio.writeport() (on page 7-70)

display.clear()

This function clears all lines of the front-panel display.

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

Usage

```
display.clear()
```

Details

This function switches to the user screen and then clears the front-panel display.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the display. For performance considerations, they update the display as soon as processing time becomes available.

Also see

display.setcursor() (on page 7-89)
display.settext() (on page 7-90)

display.getannunciators()

This function reads the annunciators (indicators) that are presently turned on.

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

Usage

```
annunciators = display.getannunciators()
```

annunciators	The bitmasked value that shows which indicators are turned on

Details

This function returns a bitmasked value showing which indicators are turned on. The 16-bit binary equivalent of the returned value is the bitmask. The return value is a sum of set annunciators, based on the weighted value, as shown in the following table.

Annunciator (indicator) bitmasked values and equivalent constants

Indicator	Bit	Weighted value	Equivalent constant
FILT	1	1	display.ANNUNCIATOR_FILTER
MATH	2	2	display.ANNUNCIATOR_MATH
4W	3	4	display.ANNUNCIATOR_4_WIRE
AUTO	4	8	display.ANNUNCIATOR_AUTO
ARM	5	16	display.ANNUNCIATOR_ARM
TRIG	6	32	display.ANNUNCIATOR_TRIGGER
* (asterisk)	7	64	display.ANNUNCIATOR_STAR
SMPL	8	128	display.ANNUNCIATOR_SAMPLE
EDIT	9	256	display.ANNUNCIATOR_EDIT
ERR	10	512	display.ANNUNCIATOR_ERROR
REM	11	1024	display.ANNUNCIATOR_REMOTE
TALK	12	2048	display.ANNUNCIATOR_TALK
LSTN	13	4096	display.ANNUNCIATOR_LISTEN
SRQ	14	8192	display.ANNUNCIATOR_SRQ
REAR	15	16384	display.ANNUNCIATOR_REAR
REL	16	32768	display.ANNUNCIATOR_REL

Example 1

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

rem = bit.bitand(testAnnunciators, 1024)
if rem > 0 then
   print("REM is on")
else
   print("REM is off")
end
```

REM indicator is turned on.
Output:

```
1.28000e+03
REM is on
```

Example 2

```
print(display.ANNUNCIATOR_EDIT)
print(display.ANNUNCIATOR_TRIGGER)
print(display.ANNUNCIATOR_AUTO)
```

Output:

```
2.56000e+02
3.20000e+01
8.00000e+00
```

Also see

bit.bitand() (on page 7-8)

display.getcursor()

This function reads the present position of the cursor on the front-panel display.

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

Usage

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

row	The row where the cursor is: `1` (top row); `2` (bottom row)
column	The column where the cursor is: ■ If the cursor is in the top row: 1 to 20 ■ If the cursor is in the bottom row: 1 to 32
style	Visibility of the cursor: ■ Invisible: `0` ■ Blinking: `1`

Details

This function switches the front-panel display to the user screen (the text set by `display.settext()`), and then returns values to indicate the row that contains the cursor and the column position and cursor style.

Columns are numbered from left to right on the display.

Example 1

```
testRow, testColumn = display.getcursor()
print(testRow, testColumn)
```

This example reads the cursor position into local variables and prints them.
Example output:

```
1.00000e+00
```

Example 2

```
print(display.getcursor())
```

This example prints the cursor position directly. In this example, the cursor is in row 1 at column 3, with an invisible cursor:

```
1.00000e+00    3.00000e+00    0.00000e+00
```

Also see

display.gettext() (on page 7-76)
display.screen (on page 7-87)
display.setcursor() (on page 7-89)
display.settext() (on page 7-90)

display.getlastkey()

This function retrieves the key code for the last pressed key.

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

Usage

```
keyCode = display.getlastkey()
```

keyCode	A returned value that represents the last front-panel key pressed; see **Details** for more information

Details

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

Pressing the EXIT (LOCAL) key normally aborts a script. To use this function with the EXIT (LOCAL) key, you must set `display.locallockout` to `display.LOCK`.

The table below lists the *keyCode* value for each front-panel action.

Key codes

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

NOTE

When using this function, use built-in constants, such as `display.KEY_RIGHT`, rather than the numeric value, such as `103`. This allows for better forward compatibility with firmware revisions.

You cannot use this function to track the OUTPUT ON/OFF control function.

Example

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

On the front panel, press the **MENU** key and then send the code shown here. This retrieves the key code for the last pressed key.
Output:
`6.80000e+01`

Also see

display.locallockout (on page 7-83)
display.sendkey() (on page 7-87)

display.gettext()

This function reads the text displayed on the front panel.

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

Usage

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

text	The returned value, which contains the text that is presently displayed
embellished	Indicates type of returned text: `false` (simple text); `true` (text with embedded character codes)
row	Selects the row from which to read the text: `1` (row 1); `2` (row 2). If *row* is not included, both rows of text are read
columnStart	Selects the first column from which to read text; for row 1, the valid column numbers are 1 to 20; for row 2, the valid column numbers are 1 to 32; if nothing is selected, 1 is used
columnEnd	Selects the last column from which to read text; for row 1, the valid column numbers are 1 to 20; for row 2, the valid column numbers are 1 to 32; the default is 20 for row 1, and 32 for row 2

Details

Using the command without any parameters returns both lines of the front-panel display.

The `$N` character code is included in the returned value to show where the top line ends and the bottom line begins. This is not affected by the value of *embellished*.

When *embellished* is set to `true`, all other character codes are returned along with the message. When *embellished* is set to `false`, only the message and the `$N` character code is returned. For information on the embedded character codes, see display.settext() (on page 7-90).

The display is not switched to the user screen (the screen set using `display.settext()`). Text is read from the active screen.

Example 1

```
display.clear()
display.setcursor(1, 1)
display.settext("ABCDEFGHIJ$DKLMNOPQRST")
display.setcursor(2, 1)
display.settext("abcdefghijklm$Bnopqrstuvwxyz$F123456")
print(display.gettext())
print(display.gettext(true))
print(display.gettext(false, 2))
print(display.gettext(true, 2, 9))
print(display.gettext(false, 2, 9, 10))
```

This example shows how to retrieve the display text in multiple ways. The output is:

```
ABCDEFGHIJKLMNOPQRST$Nabcdefghijklmnopqrstuvwxyz123456
$RABCDEFGHIJ$DKLMNOPQRST$N$Rabcdefghijklm$Bnopqrstuvwxyz$F123456
abcdefghijklmnopqrstuvwxyz123456
$Rijklm$Bnopqrstuvwxyz$F123456
ij
```

Example 2

```
display.clear()
display.settext("User Screen")
text = display.gettext()
print(text)
```

This outputs all text in both lines of the display:

```
User Screen       $N
```

This indicates that the message “User Screen” is on the top line. The bottom line is blank.

Also see

display.clear() (on page 7-71)
display.getcursor() (on page 7-74)
display.setcursor() (on page 7-89)
display.settext() (on page 7-90)

display.inputvalue()

This function displays a formatted input field on the front-panel display that the operator can edit.

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

Usage

```
display.inputvalue("format")
display.inputvalue("format", default)
display.inputvalue("format", default, minimum)
display.inputvalue("format", default, minimum, maximum)
```

format	A string that defines how the input field is formatted; see **Details** for more information
default	The default value for the input value
minimum	The minimum input value
maximum	The maximum input value

Details

The *format* parameter uses zeros (0), the decimal point, polarity sign, and exponents to define how the input field is formatted. The *format* parameter can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially	0.00000e+0
+	Allows operators to enter positive or negative values; if the "+" sign is not included, the operator cannot enter a negative value	+0.00
0	Defines the digit positions for the value; up to six zeros (0)	+00.0000e+00
.	Include to have a decimal point appear in the value	+0.00

The *default* parameter is the value shown when the value is first displayed.

You can use the *minimum* and *maximum* parameters to limit the values that can be entered. When + is not selected for *format*, the minimum limit must be more than or equal to zero (0). When limits are used, you cannot enter values above or below these limits.

The input value is limited to ±1e37.

Before calling `display.inputvalue()`, you should send a message prompt to the operator using `display.prompt()`. Make sure to position the cursor where the edit field should appear.

After this command is sent, script execution pauses until you enter a value and press the **ENTER** key.

For positive and negative entry (plus sign (+) used for the value field and/or the exponent field), polarity of a nonzero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity is also toggled when using the navigation wheel to decrease or increase the value or exponent past zero. A zero (0) value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After executing this command and pressing the EXIT (LOCAL) key, the function returns `nil`.

Example

```
display.clear()
display.settext("Enter value between$N  -0.10 and 2.00:  ")
value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)
print("Value entered = ", value)
```

Displays an editable field (+0.50) for operator input. The valid input range is −0.10 to +2.00, with a default of 0.50.
Output:

```
Value entered =    1.35000e+00
```

Also see

display.prompt() (on page 7-85)
display.setcursor() (on page 7-89)
display.settext() (on page 7-90)

display.loadmenu.add()

This function adds an entry to the USER menu, which can be accessed by pressing the **LOAD** key on the front panel.

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

Usage

```
display.loadmenu.add("displayName", "code")
display.loadmenu.add("displayName", "code", memory)
```

displayName	The name that is added to the USER menu
code	The code that is run from the USER menu
memory	Determines if code is saved to nonvolatile memory: ■ `0` or `display.DONT_SAVE`: Does not save the code to nonvolatile memory ■ `1` or `display.SAVE`: Saves the code to nonvolatile memory (default)

Details

After adding code to the load menu, you can run it from the front panel by pressing the **LOAD** key, then selecting **USER** to select from the available code to load. Pressing the **RUN** key then runs the script.

You can add items in any order. They are always displayed in alphabetical order when the menu is selected.

Any Lua code can be included in the *code* parameter. If *memory* is set to `display.SAVE`, the entry (name and code) is saved in nonvolatile memory. Scripts, functions, and variables used in the code are not saved by `display.SAVE`. Functions and variables need to be saved with the code. If the code is not saved in nonvolatile memory, it is lost when the Model 2651A is turned off. See **Example 2** below.

If you do not make a selection for *memory*, the code is automatically saved to nonvolatile memory.

NOTE

You can create a script that defines several functions, and then use the `display.loadmenu.add()` command to add items that call those individual functions. This allows the operator to run tests from the front panel.

Example 1

```
display.loadmenu.add("Test9", "Test9()")
```

Assume a user script named `Test9` is loaded into the runtime environment. Adds the menu entry to the USER menu to run the script after loading.

Example 2

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

Assume a script with a function named “DUT1” is loaded into the instrument, and the script has not been saved in nonvolatile memory.

Now assume you want to add a test named “Test” to the USER menu. You want the test to run the function named `DUT1` and sound the beeper. This example adds `Test` to the menu, defines the code, and then saves the `displayName` and code in nonvolatile memory.

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

Now assume you turn off instrument power. Because the script was not saved in nonvolatile memory, the function named `DUT1` is lost when you turn the instrument on. When `Test` is run again from the front panel, an error is generated because `DUT1` no longer exists in the instrument as a function.

Example 3

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

Adds an entry called `Part1` to the front-panel USER load menu for the code `testpart([[Part1]], 5.0)` and saves it in nonvolatile memory.

Also see

display.loadmenu.delete() (on page 7-82)

display.loadmenu.catalog()

This function creates an iterator for the user menu items accessed using the LOAD key on the front panel.

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

Usage

```
for displayName in display.loadmenu.catalog() do body end
for displayName, code in display.loadmenu.catalog() do body end
```

displayName	The name displayed in the menu
code	The code associated with the *displayName*
body	The body of the code to process the entries in the loop

Details

Each time through the loop, *displayName* and *code* take on the values in the USER menu.

The instrument goes through the list in random order.

Example

```
for displayName, code in display.loadmenu.catalog() do
   print(displayName, code)
end
```

Output:

```
Test DUT1() beeper.beep(2, 500)
Part1 testpart([[Part1]], 5.0)
Test9 Test9()
```

Also see

display.loadmenu.add() (on page 7-79)
display.loadmenu.delete() (on page 7-82)

display.loadmenu.delete()

This function removes an entry from the USER menu, which can be accessed using the **LOAD** key on the front panel.

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

Usage

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

displayName	The name to be deleted from the USER menu

Details

If you delete an entry from the USER menu, you can no longer run it by pressing the **LOAD** key.

Example

```
display.loadmenu.delete("Test9")
for displayName, code in display.loadmenu.catalog() do
   print(displayName, code)
end
```

Deletes the entry named `Test9`.
Output:

```
Test   DUT1() beeper.beep(2, 500)
Part1   testpart([[Part1]], 5.0)
```

Also see

display.loadmenu.add() (on page 7-79)
display.loadmenu.catalog() (on page 7-81)

display.locallockout

This attribute describes whether or not the EXIT (LOCAL) key on the instrument front panel is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (display.UNLOCK)

Usage

```
lockout = display.locallockout
display.locallockout = lockout
```

lockout	0 or display.UNLOCK: Unlocks EXIT (LOCAL) key 1 or display.LOCK: Locks out EXIT (LOCAL) key

Details

Set `display.locallockout` to `display.LOCK` to prevent the user from interrupting remote operation by pressing the EXIT (LOCAL) key.

Set this attribute to `display.UNLOCK` to allow the EXIT (LOCAL) key to interrupt script or remote operation.

Example

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

Disables the front-panel EXIT (LOCAL) key.

Also see

None

display.menu()

This function presents a menu on the front-panel display.

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

Usage

```
selection = display.menu("name", "items")
```

selection	Name of the variable that holds the selected menu item
name	Menu name to display on the top line
items	Menu items to display on the bottom line

Details

The menu consists of the menu name string on the top line, and a selectable list of items on the bottom line. The menu items must be a single string with each item separated by whitespace. The name for the top line is limited to 20 characters.

After sending this command, script execution pauses for the operator to select a menu item. An item is selected by rotating the navigation wheel to place the blinking cursor on the item, and then pressing the navigation wheel (or the ENTER key). When an item is selected, the text of that selection is returned.

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

Example

```
selection = display.menu("Menu", "Test1 Test2 Test3")
print(selection)
```

Displays a menu with three menu items. If the second menu item is selected, selection is given the value `Test2`.

Output:

```
Test2
```

Also see

None

display.numpad

This attribute controls whether the front panel keys act as a numeric keypad during value entry.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	1 (display.ENABLE)

Usage

```
numericKeypad = display.numpad
display.numpad = numericKeypad
```

numericKeypad	Enable the numeric keypad feature (`1` or `display.ENABLE`) Disable the numeric keypad feature (`0` or `display.DISABLE`)

Details

The numeric keypad feature is only available when editing a numeric value at the same time that the EDIT indicator is lit.

Example

```
display.numpad = display.ENABLE
```

Turn on the numeric keypad feature.

Also see

Setting a value (on page 2-16)

display.prompt()

This function prompts the user to enter a parameter from the front panel of the instrument.

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

Usage

```
display.prompt("format", "units", "help")
display.prompt("format", "units", "help", default)
display.prompt("format", "units", "help", default, minimum)
display.prompt("format", "units", "help", default, minimum, maximum)
```

format	A string that defines how the input field is formatted; see **Details** for more information
units	Set the units text string for the top line (eight characters maximum); this indicates the units (for example, "V" or "A") for the value
help	Text string to display on the bottom line (32 characters maximum)
default	The value that is shown when the value is first displayed
minimum	The minimum input value that can be entered
maximum	The maximum input value that can be entered (must be more than minimum)

Details

This function creates an editable input field at the present cursor position, and an input prompt message on the bottom line. Example of a displayed input field and prompt:

```
0.00V
Input 0 to +2V
```

The *format* parameter uses zeros (0), the decimal point, polarity sign, and exponents to define how the input field is formatted.

The *format* parameter can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially. Include a plus sign (+) for positive/negative exponent entry. Do not include the plus sign (+) to prevent negative value entry. 0 defines the digit positions for the exponent.	`0.00000E+0`
+	Allows operators to enter positive or negative values. If the plus sign (+) is not included, the operator cannot enter a negative value.	`+0.00`
0	Defines the digit positions for the value. You can use up to six zeros (0).	`+00.0000E+00`
.	The decimal point where needed for the value.	`+0.00`

You can use the *minimum* and *maximum* parameters to limit the values that can be entered. When a plus sign (+) is not selected for *format*, the minimum limit must be greater than or equal to zero (0). When limits are used, the operator cannot enter values above or below these limits.

The input value is limited to ±1e37.

After sending this command, script execution pauses for the operator to enter a value and press **ENTER**.

For positive and negative entry (plus sign (+) used for the value field and the exponent field), polarity of a nonzero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity also toggles when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After executing this command and pressing the **EXIT (LOCAL)** key, the value returns `nil`.

Example

```
value = display.prompt("0.00", "V", "Input 0 to +2V", 0.5, 0, 2)
print(value)
```

The above command prompts the operator to enter a voltage value. The valid input range is 0 to +2.00, with a default of 0.50:

```
0.50V
Input 0 to +2V
```

If the operator enters 0.70, the output is:

```
7.00000e-01
```

Also see

display.inputvalue() (on page 7-78)

display.screen

This attribute contains the selected display screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0 (display.SMUA)

Usage

```
displayID = display.screen
display.screen = displayID
```

displayID	One of the following values: ■ `0` or `display.SMUA`: Displays source-measure and compliance limit ■ `3` or `display.USER`: Displays the user screen

Details

Setting this attribute selects the display screen for the front panel. This performs the same action as pressing the DISPLAY key on the front panel. The text for the display screen is set by `display.settext()`.

Read this attribute to determine which of the available display screens was last selected.

Example

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

Selects the source-measure and compliance limit display for the SMU.

Also see

display.settext() (on page 7-90)

display.sendkey()

This function sends a code that simulates the action of a front-panel control.

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

Usage

```
display.sendkey(keyCode)
```

keyCode	A parameter that specifies the key press to simulate; see **Details** for more information

Details

This command simulates pressing a front-panel key or navigation wheel, or turning the navigation wheel one click to the left or right.

Key codes				
Value	**Key list**		**Value**	**Key list**
65	display.KEY_RANGEUP		85	display.KEY_RECALL
68	display.KEY_MENU		86	display.KEY_MEASA
69	display.KEY_MODEA		87	display.KEY_DIGITSA
70	display.KEY_RELA		88	display.KEY_OUTPUTA
71	display.KEY_RUN		92	display.KEY_TRIG
72	display.KEY_DISPLAY		93	display.KEY_LIMITA
73	display.KEY_AUTO		94	display.KEY_SPEEDA
75	display.KEY_EXIT		95	display.KEY_LOAD
77	display.KEY_FILTERA		97	display.WHEEL_ENTER
78	display.KEY_STORE		103	display.KEY_RIGHT
79	display.KEY_SRCA		104	display.KEY_LEFT
80	display.KEY_CONFIG		107	display.WHEEL_LEFT
81	display.KEY_RANGEDOWN		114	display.WHEEL_RIGHT
82	display.KEY_ENTER			

NOTE

When using this function, use built-in constants, such as `display.KEY_RIGHT`, rather than the numeric value, such as `103`. This allows for better forward compatibility with firmware revisions.

Example

`display.sendkey(display.KEY_RUN)`	Simulates pressing the RUN key.

Also see

Front panel (on page 2-2)

display.setcursor()

This function sets the position of the cursor.

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

Usage

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

row	The row number for the cursor (1 or 2)
column	The active column position to set; row 1 has columns 1 to 20, row 2 has columns 1 to 32
style	Set the cursor to invisible (0, default) or blinking (1)

Details

Sending this command selects the user screen and then moves the cursor to the given location.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the display. For performance considerations, they update the display as soon as processing time becomes available.

An out-of-range parameter for *row* sets the cursor to row 2. An out-of-range parameter for *column* sets the cursor to column 20 for row 1, or 32 for row 2.

An out-of-range parameter for *style* sets it to 0 (invisible).

A blinking cursor is only visible when it is positioned over displayed text. It cannot be seen when positioned over a space character.

Example

```
display.clear()
display.setcursor(1, 8)
display.settext("Hello")
display.setcursor(2, 14)
display.settext("World")
```

This example displays a message on the front panel, approximately center. Note that the top line of text is larger than the bottom line of text.
The front panel of the instrument displays `Hello` on the top line and `World` on the second line.

Also see

display.clear() (on page 7-71)
display.getcursor() (on page 7-74)
display.gettext() (on page 7-76)
display.screen (on page 7-87)
display.settext() (on page 7-90)

display.settext()

This function displays text on the front-panel user screen.

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

Usage

```
display.settext("text")
```

text	Text message to be displayed, with optional character codes

Details

This function selects the user display screen and displays the given text.

After the instrument is turned on, the first time you use a display command to write to the display, the message "User Screen" is cleared. After the first write, you need to use `display.clear()` to clear the message.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the display. For performance considerations, they update the display as soon as processing time becomes available.

The text starts at the present cursor position. After the text is displayed, the cursor is after the last character in the display message.

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

The text remains on the display until replaced or cleared.

The character codes described in the following table can also be included in the text string.

Display character codes

Character Code	Description
`$N`	Newline, starts text on the next line; if the cursor is already on line 2, text is ignored after the `$N` is received
`$R`	Sets text to normal intensity, nonblinking
`$B`	Sets text to blink
`$D`	Sets text to dim intensity
`$F`	Sets the text to background blink
`$$`	Escape sequence to display a single dollar symbol (`$`)

Example

```
display.clear()
display.settext("Normal $BBlinking$N")
display.settext("$DDim $FBackgroundBlink$R $$$$ 2 dollars")
```

This example sets the display to:

```
Normal Blinking
Dim BackgroundBlink $$ 2 dollars
```

with the named effect on each word.

Also see

display.clear() (on page 7-71)
display.getcursor() (on page 7-74)
display.gettext() (on page 7-76)
display.screen (on page 7-87)
display.setcursor() (on page 7-89)

display.smuX.digits

This attribute sets the front-panel display resolution of the selected measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	5 (display.DIGITS_5_5)

Usage

```
digits = display.smuX.digits
display.smuX.digits = digits
```

digits	Set digits to one of the following values: ■ Select 4½ digit resolution (`4` or `display.DIGITS_4_5`) ■ Select 5½ digit resolution (`5` or `display.DIGITS_5_5`) ■ Select 6½ digit resolution (`6` or `display.DIGITS_6_5`)

Details

This attribute sets the display resolution.

Example

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

Select 5½ digit resolution for SMU A.

Also see

Display resolution (on page 3-80)

display.smuX.limit.func

This attribute specifies the type of limit value setting displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0 (display.LIMIT_IV)

Usage

```
func = display.smuX.limit.func
display.smuX.limit.func = func
```

func	One of the following values: ■ `0` or `display.LIMIT_IV`: Displays the primary limit setting ■ `1` or `display.LIMIT_P`: Displays the power limit setting
X	Source-measure unit (SMU) channel (for example, `display.smua.limit.func` applies to SMU channel A)

Details

Selects the displayed limit function: primary (IV) or power (P).

Example

```
display.smua.limit.func = display.LIMIT_P
```

Specifies the power limit value is displayed for SMU Channel A.

Also see

display.smu*X*.measure.func (on page 7-93)
Display mode (on page 2-19)

display.smuX.measure.func

This attribute specifies the type of measurement that is being displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	1 (display.MEASURE_DCVOLTS)

Usage

```
func = display.smuX.measure.func
display.smuX.measure.func = func
```

func	The type of measurement: ■ 0 or display.MEASURE_DCAMPS: Current measurement function ■ 1 or display.MEASURE_DCVOLTS: Voltage measurement function ■ 2 or display.MEASURE_OHMS: Resistance measurement function ■ 3 or display.MEASURE_WATTS: Power measurement function
X	Source-measure unit (SMU) channel (for example, display.smua.measure.func applies to SMU channel A)

Details

Selects the measurement function that is displayed on the front panel: Amps, volts, ohms, or watts.

Example

```
display.smua.measure.func = display.MEASURE_DCAMPS
```

Selects the current measure function for SMU A.

Also see

display.smuX.limit.func (on page 7-92)

display.trigger.clear()

This function clears the front-panel trigger event detector.

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

Usage

```
display.trigger.clear()
```

Details

The trigger event detector remembers if an event has been detected since the last `display.trigger.wait()` call. This function clears the trigger event detector and discards the previous history of TRIG key presses.

This attribute also clears the `display.trigger.overrun` attribute.

Also see

display.trigger.overrun (on page 7-95)
display.trigger.wait() (on page 7-95)

display.trigger.EVENT_ID

This constant is the event ID of the event generated when the front-panel TRIG key is pressed.

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

Usage

```
eventID = display.trigger.EVENT_ID
```

eventID	The trigger event number

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to front-panel trigger key events.

Also see

None

display.trigger.overrun

This attribute contains the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Display trigger clear Instrument reset Recall setup	Not saved	false

Usage

```
overrun = display.trigger.overrun
```

overrun	The trigger overrun state (`true` or `false`)

Details

Indicates if a trigger event was ignored because the event detector was already in the detected state when the TRIG button was pressed.

Indicates the overrun state of the event detector built into the display.

This attribute does not indicate whether an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = display.trigger.overrun
```

Sets the variable `overrun` equal to the present state of the event detector built into the display.

Also see

display.trigger.clear() (on page 7-94)

display.trigger.wait()

This function waits for the TRIG key on the front panel to be pressed.

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

Usage

```
triggered = display.trigger.wait(timeout)
```

triggered	`true`: Trigger was detected `false`: The operation timed out
timeout	Timeout in seconds

Details

If the trigger key was previously pressed and one or more trigger events were detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Use the `display.trigger.clear()` call to clear the trigger event detector.

Example

`triggered = display.trigger.wait(5)` `print(triggered)`	Waits up to five seconds for the TRIG key to be pressed. If TRIG is pressed within five seconds, the output is `true`. If not, the output is `false`.

Also see

display.trigger.clear() (on page 7-94)

display.waitkey()

This function captures the key code value for the next front-panel action.

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

Usage

keyCode = display.waitkey()

keyCode	See **Details** for more information

Details

After you send this function, script execution pauses until a front-panel action (for example, pressing a key or the navigation wheel, or turning the navigation wheel). After the action, the value of the key (or action) is returned.

If the EXIT (LOCAL) key is pressed while this function is waiting for a front-panel action, the script is not aborted.

A typical use for this function is to prompt the user to press the EXIT (LOCAL) key to abort the script or press any other key to continue. For example, if the `keyCode` value `75` is returned (the EXIT (LOCAL) key was pressed), the `exit()` function can be called to abort the script.

The table below lists the *keyCode* value for each front panel action.

Key codes			
Value	**Key (or action)**	**Value**	**Key (or action)**
65	display.KEY_RANGEUP	85	display.KEY_RECALL
68	display.KEY_MENU	86	display.KEY_MEASA
69	display.KEY_MODEA	86	display.KEY_DIGITSA
70	display.KEY_RELA	88	display.KEY_OUTPUTA
71	display.KEY_RUN	92	display.KEY_TRIG
72	display.KEY_DISPLAY	93	display.KEY_LIMITA
73	display.KEY_AUTO	94	display.KEY_SPEEDA
75	display.KEY_EXIT	95	display.KEY_LOAD
77	display.KEY_FILTERA	97	display.WHEEL_ENTER
78	display.KEY_STORE	103	display.KEY_RIGHT
79	display.KEY_SRCA	104	display.KEY_LEFT
80	display.KEY_CONFIG	107	display.WHEEL_LEFT
81	display.KEY_RANGEDOWN	114	display.WHEEL_RIGHT
82	display.KEY_ENTER		

NOTE

When using this function, use built-in constants, such as `display.KEY_RIGHT`, rather than the numeric value, such as `103`. This allows for better forward compatibility with firmware revisions.

Example

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

Pause script execution until the operator presses a key or the navigation wheel, or rotates the navigation wheel.
If the output is:
`8.60000e+01`
It indicates that the MEAS(A) key was pressed.

Also see

Capturing key-press codes (on page 3-90)
display.getlastkey() (on page 7-75)
display.sendkey() (on page 7-87)
display.settext() (on page 7-90)

errorqueue.clear()

This function clears all entries out of the error queue.

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

Usage

```
errorqueue.clear()
```

Details

See Error queue (on page 15-3) for additional information about the error queue.

Also see

errorqueue.count (on page 7-98)
errorqueue.next() (on page 7-99)

errorqueue.count

This attribute gets the number of entries in the error queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle Clearing error queue Reading error messages	Not applicable	Not applicable

Usage

```
count = errorqueue.count
```

count	The number of entries in the error queue

Example

```
count = errorqueue.count
print(count)
```

Returns the number of entries in the error queue.
The output below indicates that there are four entries in the error queue:

```
4.00000e+00
```

Also see

Error queue (on page 15-3)
errorqueue.clear() (on page 7-98)
errorqueue.next() (on page 7-99)

errorqueue.next()

This function reads the oldest entry from the error queue and removes it from the queue.

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

Usage

```
errorCode, message, severity, errorNode = errorqueue.next()
```

errorCode	The error code number for the entry
message	The message that describes the error code
severity	The severity level (0, 10, 20, 30, or 40); see **Details** for more information
errorNode	The node number where the error originated

Details

Entries are stored in a first-in, first-out (FIFO) queue. This function reads the oldest entry and removes it from the queue.

Error codes and messages are listed in the Error summary list (on page 8-3).

If there are no entries in the queue, code 0, `Queue is Empty`, is returned.

Returned severity levels are described in the following table.

Number	Error level	Description
0	NO_SEVERITY	The message is information only. This level is used when the Error Queue is empty; the message does not represent an error.
10	INFORMATIONAL	The message is information only. This level is used to indicate status changes; the message does not represent an error.
20	RECOVERABLE	The error was caused by improper use of the instrument or by conditions that can be corrected. This message indicates that an error occurred. The instrument is still operating normally.
30	SERIOUS	There is a condition that prevents the instrument from functioning properly. The message indicates that the instrument is presently operating in an error condition. If the condition is corrected, the instrument returns to normal operation.
40	FATAL	There is a condition that cannot be corrected that prevents the instrument from functioning properly. Disconnect the DUT and turn the power off and then on again. If the error is a hardware fault that persists after cycling the power, the instrument must be repaired.

In an expanded system, each TSP-Link enabled instrument is assigned a node number. The variable *errorNode* stores the node number where the error originated.

Example

```
errorcode, message = errorqueue.next()
print(errorcode, message)
```

Reads the oldest entry in the error queue. The output below indicates that the queue is empty.
Output:

```
0.00000e+00 Queue Is Empty
```

Also see

Error queue (on page 15-3)
errorqueue.clear() (on page 7-98)
errorqueue.count (on page 7-98)
Error summary list (on page 8-3)

eventlog.all()

This function returns all entries from the event log as a single string and removes them from the event log.

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

Usage

```
logString = eventlog.all()
```

logString	A listing of all event log entries

Details

This function returns all events in the event log. Logged items are shown from oldest to newest. The response is a string that has the messages delimited with a new line character.

This function also clears the event log.

If there are no entries in the event log, this function returns the value `nil`.

Example

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

Get and print all entries from the event log and remove the entries from the log.
Output:

```
17:26:35.690 10 Oct 2019, LAN0, 192.168.1.102, LXI, 0, 1570728395,
      1192037155.733269000, 0, 0x0
17:26:39.009 10 Oct 2019, LAN5, 192.168.1.102, LXI, 0, 1570728399,
      1192037159.052777000, 0, 0x0
```

Also see

eventlog.clear() (on page 7-101)
eventlog.count (on page 7-101)
eventlog.enable (on page 7-102)
eventlog.next() (on page 7-102)
eventlog.overwritemethod (on page 7-103)

eventlog.clear()

This function clears the event log.

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

Usage

```
eventlog.clear()
```

Details

This command removes all messages from the event log.

Also see

eventlog.all() (on page 7-100)
eventlog.count (on page 7-101)
eventlog.enable (on page 7-102)
eventlog.next() (on page 7-102)
eventlog.overwritemethod (on page 7-103)

eventlog.count

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

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Clearing event log Reading event log	Not applicable	Not applicable

Usage

```
N = eventlog.count
```

N	The number of events in the event log

Example

```
print(eventlog.count)
```

Displays the present number of events in the instrument event log.
Output looks similar to:

```
3.00000e+00
```

Also see

eventlog.all() (on page 7-100)
eventlog.clear() (on page 7-101)
eventlog.enable (on page 7-102)
eventlog.next() (on page 7-102)
eventlog.overwritemethod (on page 7-103)

eventlog.enable

This attribute enables or disables the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Not saved	1 (eventlog.ENABLE)

Usage

```
status = eventlog.enable
eventlog.enable = status
```

status	The enable status of the event log: 1 or eventlog.ENABLE: Event log enable 0 or eventlog.DISABLE: Event log disable

Details

When the event log is disabled (eventlog.DISABLE or 0), no new events are added to the event log. You can, however, read and remove existing events.

When the event log is enabled, new events are logged.

Example

```
print(eventlog.enable)
eventlog.enable = eventlog.DISABLE
print(eventlog.enable)
```

Displays the present status of the Model 2651A event log.
Output:

```
1.00000e+00
0.00000e+00
```

Also see

eventlog.all() (on page 7-100)
eventlog.clear() (on page 7-101)
eventlog.count (on page 7-101)
eventlog.next() (on page 7-102)
eventlog.overwritemethod (on page 7-103)

eventlog.next()

This function returns the oldest unread event message from the event log and removes it from the event log.

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

Usage

```
logString = eventlog.next()
```

logString	The next log entry

Details

Returns the next entry from the event log and removes it from the log.

If there are no entries in the event log, returns the value `nil`.

Example 1

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

Get the oldest message in the event log and remove that entry from the log.
Output:

```
17:28:22.085 10 Oct 2019, LAN2, 192.168.1.102, LXI, 0, 1570728502, <no time>, 0, 0x0
```

Example 2

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

If you send this command when there is nothing in the event log, you get the following output:

```
nil
```

Also see

eventlog.all() (on page 7-100)
eventlog.clear() (on page 7-101)
eventlog.count (on page 7-101)
eventlog.enable (on page 7-102)
eventlog.overwritemethod (on page 7-103)

eventlog.overwritemethod

This attribute controls how the event log processes events if the event log is full.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Not saved	1 (eventlog.DISCARD_OLDEST)

Usage

```
method = eventlog.overwritemethod
eventlog.overwritemethod = method
```

method	Set to one of the following values: ■ `0` or `eventlog.DISCARD_NEWEST`: New entries are not logged ■ `1` or `eventlog.DISCARD_OLDEST`: Old entries are deleted as new events are logged

Details

When this attribute is set to `eventlog.DISCARD_NEWEST`, new entries are not logged.

When this attribute is set to `eventlog.DISCARD_OLDEST`, the oldest entry is discarded when a new entry is added.

Example

```
eventlog.overwritemethod = 0
```

When the log is full, the event log ignores new entries.

Also see

eventlog.all() (on page 7-100)
eventlog.clear() (on page 7-101)
eventlog.count (on page 7-101)
eventlog.enable (on page 7-102)
eventlog.next() (on page 7-102)

exit()

This function stops a script that is presently running.

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

Usage

```
exit()
```

Details

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

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

Also see

waitcomplete() (on page 7-459)

fileVar:close()

This function closes the file that is represented by the *fileVar* variable.

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

Usage

```
fileVar:close()
```

fileVar	The file descriptor variable to close

Details

This command is equivalent to `io.close(fileVar)`.

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

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

print("\n*** fileVar:close")
do
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
end
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
myfile:close()
os.remove(fileName)
```

Opens file `myfile.txt` for writing. If no errors were found while opening, writes `Removing file` and closes the file.

Also see

fileVar:flush() (on page 7-105)
fileVar:read() (on page 7-107)
fileVar:seek() (on page 7-108)
fileVar:write() (on page 7-110)
io.close() (on page 7-129)
io.open() (on page 7-132)

fileVar:flush()

This function writes buffered data to a file.

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

Usage

fileVar:flush()

fileVar	The file descriptor variable to flush

Details

The *fileVar*:write() or io.write() functions buffer data, which may not be written immediately to the USB flash drive. Use *fileVar*:flush() to flush this data. Using this function removes the need to close a file after writing to it, allowing the file to be left open to write more data. Data may be lost if the file is not closed or flushed before a script ends.

If there is going to be a time delay before more data is written to a file, and you want to keep the file open, flush the file after you write to it to prevent loss of data.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

errorqueue.clear()
print("\n*** io.read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1\n")
myfile:flush()
myfile:close()
do
fileHandle = io.input(fileName)
value = io.read("*a")
print(value)
end
fileHandle:close()

print(errorqueue.next())
```

Writes data to a USB flash drive.

Also see

fileVar:write() (on page 7-110)
io.open() (on page 7-132)
io.write() (on page 7-136)

fileVar:read()

This function reads data from a file.

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

Usage

```
data1 = fileVar:read()
data1 = fileVar:read(format1)
data1, data2 = fileVar:read("format1", "format2")
data1, ..., datan = fileVar:read("format1", ..., "formatn")
```

data1	First data read from the file
data2	Second data read from the file
datan	Last data read from the file
fileVar	The descriptor of the file to be read
format1	A string or number indicating the first type of data to be read
format2	A string or number indicating the second type of data to be read
formatn	A string or number indicating the last type of data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

"*n": Returns a number.

"*a": Returns the whole file, starting at the current position (returns an empty string if the current file position is at the end of the file).

"*l": Returns the next line, skipping the end of line; returns nil if the current file position is at the end of file.

n: Returns a string with up to *n* characters; returns an empty string if n is zero; returns nil if the current file position is at the end of file.

If no format parameters are provided, the function performs as if the function is passed the value "*l".

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

print("fileVar:read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
contents = myfile:read("*a")
print(contents)
end
myfile:close()
os.remove(fileName)
```

Reads data from the input file.

Also see

fileVar:write() (on page 7-110)
io.input() (on page 7-131)
io.open() (on page 7-132)

fileVar:seek()

This function sets and gets the present position of a file.

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

Usage

```
position, errorMsg = fileVar:seek()
position, errorMsg = fileVar:seek("whence")
position, errorMsg = fileVar:seek("whence", offset)
```

position	The new file position, measured in bytes from the beginning of the file
errorMsg	A string containing the error message
fileVar	The file descriptor variable
whence	A string indicating the base against which *offset* is applied; the default is `"cur"`
offset	The intended new position, measured in bytes from a base indicated by *whence* (default is `0`)

Details

The *whence* parameters may be any of the following:

"set": Beginning of file

"cur": Current position

"end": End of file

If an error is encountered, it is logged to the error queue, and the command returns nil and the error string.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

errorqueue.clear()

print("\n*** fileVar:seek")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
position = myfile:seek("end", -1)
print(position)
end
myfile:close()
os.remove(fileName)
```

Get the present position of a file.

Also see

io.open() (on page 7-132)

fileVar:write()

This function writes data to a file.

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

Usage

```
fileVar:write(data)
fileVar:write(data1, data2)
fileVar:write(data1, ..., datan)
```

fileVar	The file descriptor variable
data	Write all data to the file
data1	The first data to write to the file
data2	The second data to write to the file
datan	The last data to write to the file
...	One or more entries (or values) separated by commas

Details

This function may buffer data until a flush (*fileVar*:flush() or io.flush()) or close (*fileVar*:close() or io.close()) operation is performed.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

errorqueue.clear()

print("\n*** fileVar:write")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
do
myfile:write("Line 1")
end
myfile:close()
os.remove(fileName)
```

Write data to a file.

Also see

fileVar:close() (on page 7-104)
fileVar:flush() (on page 7-105)
io.close() (on page 7-129)
io.flush() (on page 7-130)
io.open() (on page 7-132)

format.asciiprecision

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

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset Recall setup	Not saved	6

Usage

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

precision	A number representing the number of digits to be printed for numbers printed with the `print()`, `printbuffer()`, and `printnumber()` functions; must be a number between `1` and `16`

Details

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

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

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x)
```

Output:

```
2.540000000e+00

2.54e+00
```

Also see

format.byteorder (on page 7-112)
format.data (on page 7-113)
print() (on page 7-190)
printbuffer() (on page 7-191)
printnumber() (on page 7-192)

format.byteorder

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

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Not saved	1 (format.LITTLEENDIAN)

Usage

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

order	Byte order value as follows: ■ Most significant byte first: `0`, `format.NORMAL`, `format.NETWORK`, or `format.BIGENDIAN` ■ Least significant byte first: `1`, `format.SWAPPED` or `format.LITTLEENDIAN`

Details

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

`format.NORMAL`, `format.BIGENDIAN`, and `format.NETWORK` select the same byte order. `format.SWAPPED` and `format.LITTLEENDIAN` select the same byte order. Selecting which to use is a matter of preference.

Select the `format.SWAPPED` or `format.LITTLEENDIAN` byte order when sending data to a computer with a Microsoft Windows operating system.

Example

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

The output depends on the terminal program you use, but it looks something like:

```
#0¤p??
#0??p¤
```

Also see

format.asciiprecision (on page 7-111)
format.data (on page 7-113)
printbuffer() (on page 7-191)
printnumber() (on page 7-192)

format.data

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

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset Recall setup	Not saved	1 (format.ASCII)

Usage

```
value = format.data
format.data = value
```

value	The format to use for data, set to one of the following values: ■ ASCII format: `1` or `format.ASCII` ■ Single-precision IEEE Std 754 binary format: `2`, `format.SREAL`, or `format.REAL32` ■ Double-precision IEEE Std 754 binary format: `3`, `format.REAL`, `format.REAL64`, or `format.DREAL`

Details

The precision of numeric values can be controlled with the `format.asciiprecision` attribute. The byte order of `format.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64` can be selected with the `format.byteorder` attribute.

REAL32 and SREAL select the same single precision format. REAL and REAL64 select the same double-precision format. They are alternative identifiers. Selecting which to use is a matter of preference.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with `#0` and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

NOTE

Binary formats are not intended to be interpreted by humans.

Example

```
format.asciiprecision = 10
x = 3.14159265
format.data = format.ASCII
printnumber(x)
format.data = format.REAL64
printnumber(x)
```

Output a number represented by *x* in ASCII using a precision of 10, then output the same number in binary using double-precision format.
Output:

```
3.141592650e+00
#0ñÔÈSû!    @
```

Also see

format.asciiprecision (on page 7-111)
format.byteorder (on page 7-112)
printbuffer() (on page 7-191)
printnumber() (on page 7-192)

fs.chdir()

This function sets the current working directory.

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

Usage

```
workingDirectory = fs.chdir("path")
```

workingDirectory	Returned value containing the working path
path	A string indicating the new working directory path

Details

The new working directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist.

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.
Verify that `/usb1/temp` is a directory and change it to be the current working directory.
Set the variable for the current working directory to be `testPath`.
The return should be:

```
/usb1/temp
```

If `/usb1/temp` is not a directory, set the variable for the current working directory to be `testPath`.
The return is:

```
/usb1
```

Also see

None

fs.cwd()

This function returns the absolute path of the current working directory.

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

Usage

```
path = fs.cwd()
```

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 is:
`/usb1`

Also see

None

fs.is_dir()

This function tests whether or not the specified path refers to a directory.

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

Usage

```
status = fs.is_dir("path")
```

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 7-117)

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 7-116)

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 7-120)

fs.readdir()

This function returns a list of the file system entries in the directory.

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

Usage

```
files = fs.readdir("path")
```

files	A table containing the names of all the file system entries in the specified directory
path	The directory path

Details

The directory path may be absolute or relative to the current working directory.

This command is nonrecursive. For example, entries in subfolders are not returned.

An error is logged to the error queue if the given path does not exist or does not represent a directory.

Example

```
rootDirectory = "/usb1/"
entries = fs.readdir(rootDirectory)
count = table.getn(entries)
print("Found a total of "..count.." files and directories")
for i = 1, count do
   print(entries[i])
end
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Set `entries` as the variable for the file system entries in `rootDirectory`.
Return the number of files and directories in the directory.

Also see

None

fs.rmdir()

This function removes a directory from the file system.

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

Usage

```
fs.rmdir("path")
```

path	The path of the directory to remove

Details

This path may be absolute or relative to the present working directory.

An error is logged to the error queue if the given path does not exist or does not represent a directory. An error is also logged if the directory is not empty.

Example

```
rootDirectory = "/usb1/"
tempDirectoryName = "temp"
if fs.is_dir(rootDirectory..tempDirectoryName) == false then
    fs.mkdir(rootDirectory..tempDirectoryName)
end
fs.rmdir(rootDirectory..tempDirectoryName)
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Set `tempDirectoryName` to be equivalent to `temp`.
Check to see if `tempDirectoryName` exists.
If it does not exist, create a directory named `temp`.
Remove the directory.

Also see

fs.mkdir() (on page 7-118)

gettimezone()

This function retrieves the local time zone.

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

Usage

```
timeZone = gettimezone()
```

timeZone	The local time zone of the instrument

Details

See `settimezone()` for additional details about the time zone format and a description of the fields.

timeZone can be in either of the following formats:

- If one parameter was used with `settimezone()`, the format used is:
  `GMThh:mm:ss`
- If four parameters were used with `settimezone()`, the format used is:
  `GMThh:mm:ssGMThh:mm:ss,Mmm.w.dw/hh:mm:ss,Mmm.w.dw/hh:mm:ss`

Example

```
timezone = gettimezone()
```

Reads the value of the local time zone.

Also see

settimezone() (on page 7-219)

gm_isweep()

This KIParlib factory script function performs a linear current sweep and calculates the transconductance (G_m) at each point.

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

Usage

gm_array, vbuf, ibuf = gm_isweep(*smu, start_i, stop_i, points*)

gm_array	A Lua table containing the calculated G_m values at each point
vbuf	A reading buffer containing the measured voltage at each point
ibuf	A reading buffer containing the measured current at each point
smu	Instrument channel (set to `smua`)
start_i	Starting current level of the sweep
stop_i	Ending current level of the sweep
points	Number of measurements between *start_i* and *stop_i* (must be ≥2)

Details

Output data includes transconductance values, reading buffer with measured voltages, reading buffer with measured voltages and currents.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

The `gm_isweep()` function performs a linear current sweep, measuring voltage and current, and then calculating the transconductance (G_m) at each point using the central difference method. It can return an array of G_m values, a reading buffer with the measured voltages, and a reading buffer with the measured currents.

Example

`gm_array = gm_isweep(smua, 0, 0.01, 20)`	Source-measure unit (SMU) A returns G_m values only.
`gm_array, vbuf = gm_isweep(smua, 0, 0.01, 20)`	SMU A returns G_m and reading buffer with measured voltages.
`gm_array, vbuf, ibuf = gm_isweep(smua, 0, 0.01, 20)`	SMU A returns G_m and reading buffers with measured voltages and currents.

Also see

gm_vsweep() (on page 7-123)
KIParlib factory script (on page 5-25)

gm_vsweep()

This KIParlib factory script function performs a linear voltage sweep and calculates the transconductance (G_m) at each point.

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

Usage

```
gm_array, ibuf, vbuf = gm_vsweep(smu, start_v, stop_v, points)
```

gm_array	A Lua table containing the calculated G_m values at each point
ibuf	A reading buffer containing the measured current at each point
vbuf	A reading buffer containing the measured voltage at each point
smu	Instrument channel (set to `smua`)
start_v	Starting voltage level of the sweep
stop_v	Ending voltage level of the sweep
points	Number of measurements between *start_v* and *stop_v* (must be ≥ 2)

Details

Output data includes transconductance values, reading buffer with measured currents, reading buffer with measured currents and voltages.

The `gm_vsweep()` function performs a linear voltage sweep, measuring voltage and current, and then calculating the transconductance (G_m) at each point using the central difference method. It can return an array of G_m values, a reading buffer with the measured currents, and a reading buffer with the measured voltages.

Example

`gm_array = gm_vsweep(smua, 0, 5, 20)`	SMU A returns G_m values only.
`gm_array, ibuf = gm_vsweep(smua, 0, 5, 20)`	SMU A returns G_m and reading buffer with measured currents.
`gm_array, ibuf, vbuf = gm_vsweep(smua, 0, 5, 20)`	SMU A returns G_m and reading buffers with measured currents and voltages.

Also see

gm_isweep() (on page 7-122)
KIParlib factory script (on page 5-25)

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	26

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 use care 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 2-83)

i_leakage_measure()

This KIHighC factory script function performs a current leakage measurement after stepping the output voltage.

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

Usage

imeas = i_leakage_measure(smu*X*, *levelv*, *limiti*, *sourcedelay*, *measurei*, *measuredelay*)

imeas	The measured current
smux	Instrument channel (for example, smua refers to SMU channel A)
levelv	Voltage level to step to when this function is called
limiti	Current limit setting for the voltage step
sourcedelay	Delay to wait before lowering the current limit for measurement
measurei	Current limit (and measure range); the current limit is lower at this level and because high-capacitance mode is active, the measure range follows
measuredelay	Delay to wait after lowering the current limit before making the measurement

Details

Use this function when high-capacitance mode is active.

When the instrument is in high-capacitance mode, this function causes the SMU to:

- Change its current limit to *limiti* with a voltage output of *levelv* for *sourcedelay* time, and then change its current limit to *measurei* (that also changes the measurement range to *measurei*) for *measuredelay* time
- When *measuredelay* time expires, a measurement is made and returned as *imeas*

When measuring leakage current:

- Charge the capacitor before calling this function (the output of the instrument is usually at a nonzero voltage before calling this function; when measuring leakage, this function does not charge the capacitor)
- Set *levelv* = 0

Example

`smua.source.highc = smua.ENABLE`
`smua.source.levelv = 5`
`smua.source.output = smua.OUTPUT_ON`
`delay(1)`
`imeas = i_leakage_measure(smua, 0, 1, 300e-3, 10e-6, .1)`
	Enable high-capacitance mode. Charge the capacitor at 5 V for 1 second set by `delay(1)`.
	
	The parameters passed on to the `i_leakage_measure()` function in this example are:
	`smu` `= smua`
	`levelv = 0 V`
	`limiti` `= 1 A`
	`sourcedelay` `= 300 ms`
	`measurei` `= 10 µA range`
	`measuredelay` `= 100 ms`
	The levels and delays depend on the value and type of capacitor used.

Also see

i_leakage_threshold() (on page 7-126)
High-capacitance mode (on page 3-73)
KIHighC factory script (on page 5-24)

i_leakage_threshold()

This KIHighC factory script function measures the current and compares it to a threshold. This continues until either the measured current drops below the threshold or the timeout expires.

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

Usage

```
f = i_leakage_threshold(smuX, levelv, limiti, sourcedelay, measurei, measuredelay,
    threshold, timeout)
```

`f`	A Boolean flag; this flag is `true` when the current is below the threshold, `false` if threshold is not reached before timeout expires
`smux`	Source-measure unit (SMU) channel (for example, `smua` applies to SMU channel A)
`levelv`	Voltage level to step to when this function is called
`limiti`	Current limit setting for the voltage step
`sourcedelay`	Delay to wait before lowering the current limit for measurement
`measurei`	Current limit (and measure range); the current limit is lower at this level and because high-capacitance mode is active, the measure range follows
`measuredelay`	Delay before the first measurement after measure range is changed
`threshold`	The specified current that establishes the test limit
`timeout`	Amount of time (in seconds) to wait for the current to drop to *`threshold`* after all the delays have occurred

Details

Use this function when high-capacitance mode is active.

When the instrument is in high-capacitance mode, this function causes the SMU to:

- Change its current limit to *limiti* with a voltage output of *levelv* for *sourcedelay* time, and then changes its current limit to *measurei* (that also changes the measurement range to *measurei*) for *measuredelay* time.
- When *measuredelay* time expires, measurements are made at a rate determined by the smu*X*.measure.nplc setting.

When testing the leakage current threshold:

- Charge the capacitor before calling this function. The output of the instrument is usually at a non-zero voltage before calling this function; when measuring leakage, this function does not charge the capacitor.
- If testing the leakage current threshold of the device, set *levelv* = 0.

Example

```
smua.source.highc = smua.ENABLE
smua.source.levelv = 5
smua.source.output = smua.OUTPUT_ON
delay(1)
pass = i_leakage_threshold(smua, 0, 1,
   300e-3, 10e-6, 100e-3, 1e-6, 1)
```

Enable high-capacitance mode.
Charge the capacitor.

The parameters passed on to the i_threshold_measure() function in this example are:
smu = smua
levelv = 0 V
limiti = 1 A
sourcedelay = 300 ms
measurei = 10 µA range
measuredelay = 100 ms
threshold = 1 µA
timeout = 1 s
The levels and delays depend on the value and type of capacitor used.
Sets pass = true if the current is measured below 1 µA in less than 1 second.

Also see

High-capacitance mode (on page 3-73)
i_leakage_measure() (on page 7-125)
KIHighC factory script (on page 5-24)

InitiatePulseTest()

This KIPulse factory script function initiates the pulse configuration assigned to *tag*.

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

Usage

```
f, msg = InitiatePulseTest(tag)
```

f	A Boolean flag; this flag is `true` when the pulse was successfully configured, `false` when errors are encountered
msg	A string message; if the *f* flag is `false`, *msg* contains an error message; if it is `true`, *msg* contains a string that indicates successful configuration
tag	Numeric identifier of the pulse configuration to be initiated

Details

This function only initiates configured pulse trains assigned to a valid *tag*. Configure the pulse before initiating it using one of the `ConfigurePulse*` functions (refer to the **Also see** section).

Example

```
smua.reset()

smua.source.rangev = 5
smua.source.rangei = 1
smua.source.levelv = 0

smua.measure.rangev = 5
smua.measure.rangei = 1
smua.measure.nplc = 0.01
smua.measure.autozero = smua.AUTOZERO_ONCE

smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode = 1

smua.source.output = smua.OUTPUT_ON

f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5,
   1, 0.002, 0.2, 10, smua.nvbuffer1, 1)

if f1 == true then
   f2, msg2 = InitiatePulseTest(1)
   print("Initiate message:", msg2)
else
   print("Config errors:", msg1)
end
```

Configure SMU channel A to generate a pulse train. If no errors are encountered, initiate the pulse train. Channel A pulses voltage from a bias level of 0 V to a pulse level of 5 V. The pulse level is present for 2 ms and the bias level for 200 ms, with a 1 A limit setting. A total of 10 pulses is generated, and the measurement data is stored in `smua.nvbuffer1`. This pulse train is assigned to `tag = 1`.

Also see

ConfigPulseIMeasureV() (on page 7-38)
ConfigPulseIMeasureVSweepLin() (on page 7-40)
ConfigPulseIMeasureVSweepLog() (on page 7-42)
ConfigPulseVMeasureI() (on page 7-44)
ConfigPulseVMeasureISweepLin() (on page 7-47)
ConfigPulseVMeasureISweepLog() (on page 7-49)
KIPulse factory script (on page 5-23)

io.close()

This function closes a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
io.close()
io.close(file)
```

file	The descriptor of the file to close

Details

If a file is not specified, the default output file closes.

Only `io.close()`, used without specifying a parameter, can be accessed from a remote node.

Example

```
testFile, testError = io.open("testfile.txt", "w")
if nil == testError then
   testFile:write("This is my test file")
   io.close(testFile)
end
```

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

Also see

io.open() (on page 7-132)

io.flush()

This function saves buffered data to a file.

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

Usage

```
io.flush()
```

Details

You must use the `io.flush()` or `io.close()` functions to write data to the file system.

NOTE

Data is not automatically written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB flash drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

This function only flushes the default output file.

Using this command removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written (and when you want to keep the file open and not close it), flush the file after writing to it.

Also see

fileVar:flush() (on page 7-105)
fileVar:write() (on page 7-110)
io.write() (on page 7-136)

io.input()

This function assigns a previously opened file, or opens a new file, as the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
fileVar = io.input()
fileVar = io.input("newfile")
```

fileVar	The descriptor of the input file or an error message (if the function fails)
newfile	A string representing the path of a file to open as the default input file, or the file descriptor of an open file to use as the default input file

Details

The *newfile* path may be absolute or relative to the current working directory.

When using this function from a remote TSP-Link® node, this command does not accept a file descriptor and does not return a value.

If the function fails, an error message is returned.

Also see

io.open() (on page 7-132)
io.output() (on page 7-133)

io.open()

This function opens a file for later reference.

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

Usage

```
fileVar, errorMsg = io.open("path")
fileVar, errorMsg = io.open("path", "mode")
```

fileVar	The descriptor of the opened file
errorMsg	Indicates whether an error was encountered while processing the function
path	The path of the file to open
mode	A string representing the intended access mode (`"r"` = read, `"w"` = write, and `"a"` = append)

Details

The path to the file to open may be absolute or relative to the current working directory. If you successfully open the file, *errorMsg* is nil and *fileVar* has the descriptor used to access the file.

If an error is encountered, the command returns `nil` for *fileVar* and an error string.

Example

```
testFile, testError = io.open("testfile.txt", "w")
if testError == nil then
   testFile:write("This is my test file")
   io.close(testFile)
end
```

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

Also see

io.close() (on page 7-129)

io.output()

This function assigns a previously opened file or opens a new file as the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
fileVar = io.output()
fileVar = io.output("newfile")
```

fileVar	The descriptor of the output file or an error message (if the function fails)
newfile	A file descriptor to assign (or the path of a file to open) as the default output file

Details

The path of the file to open may be absolute or relative to the current working directory.

When accessed from a remote node using the TSP-Link network, this command does not accept a file descriptor parameter and does not return a value.

If the function fails, an error message is returned.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

errorqueue.clear()

print("\n*** io.output")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
fileHandle = io.output(fileName)
print(fileHandle)
end
io.close(fileHandle)
print(fileHandle)
os.remove(fileName)
```

Assign the file to be the default output file.

Also see

io.input() (on page 7-131)
io.open() (on page 7-132)

io.read()

This function reads data from the default input file.

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

Usage

```
data1 = io.read()
data1 = io.read("format1")
data1, data2 = io.read("format1", "format2")
data1, ..., dataN = io.read("format1", ..., "formatN")
```

data1	The data read from the file
data2	The data read from the file
dataN	The data read from the file; the number of return values matches the number of format values given
format1	A string or number indicating the type of data to be read
format2	A string or number indicating the type of data to be read
formatN	A string or number indicating the type of data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

Format parameter	Description
"*n"	Returns a number
"*a"	Returns the whole file, starting at the present position; returns an empty string if it is at the end of file
"*l"	Default setting; returns the next line, skipping the end of line; returns nil if the present file position is at the end of file
N	Returns a string with up to *N* characters; returns an empty string if *N* is zero (0); returns nil if the present file position is at the end of file

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

errorqueue.clear()

-- io.read
print("\n*** io.read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1\n")
myfile:flush()
myfile:close()
do
fileHandle = io.input(fileName)
value = io.read("*a")
print(value)
end
fileHandle:close()

print(errorqueue.next())
```

Read data from the default input file.

Also see

None

io.type()

This function checks whether or not a given object is a file handle.

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

Usage

type = io.type(*obj*)

type	Indicates whether the object is an open file handle
obj	Object to check

Details

Returns the string "file" if the object is an open file handle. If it is not an open file handle, nil is returned.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
    os.remove(fileName)
    print("Removing file")
else
    print("Nothing removed")
end

errorqueue.clear()

print("\n*** io.type")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
fileHandle = io.output(fileName)
state = io.type(fileHandle)
print(state)
end
io.close(fileHandle)
local state = io.type(fileHandle)
print(state)
os.remove(fileName)
```

Check whether or not `fileName` is a file handle.

Also see

io.open() (on page 7-132)

io.write()

This function writes data to the default output file.

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

Usage

```
io.write()
io.write(data1)
io.write(data1, data2)
io.write(data1, ..., dataN)
```

data1	The data to be written
data2	The data to be written
dataN	The data to be written
...	One or more values separated by commas

Details

All data parameters must be either strings or numbers.

NOTE

Data is not immediately written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB flash drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

Example

```
local fileName = "/usb1/myfile.txt"

if fs.is_file(fileName) then
   os.remove(fileName)
   print("Removing file")
else
   print("Nothing removed")
end

errorqueue.clear()

print("\n*** io.write")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
fileHandle = io.output(fileName)
io.write("Line 2")
end
io.close(fileHandle)
os.remove(fileName)
```

Writes data to the default output file.

Also see

io.flush() (on page 7-130)

lan.applysettings()

This function re-initializes the LAN interface with new settings.

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

Usage

```
lan.applysettings()
```

Details

Disconnects all existing LAN connections to the instrument and re-initializes the LAN with the present configuration settings.

This function initiates a background operation. LAN configuration could be a lengthy operation. Although the function returns immediately, the LAN initialization continues to run in the background.

Even though the LAN configuration settings may not have changed since the LAN was last connected, new settings may take effect due to the dynamic nature of dynamic host configuration protocol (DHCP) or dynamic link local addressing (DLLA) configuration.

Re-initialization takes effect even if the configuration has not changed since the last time the instrument connected to the LAN.

Example

```
lan.applysettings()
```

Re-initialize the LAN interface with new settings.

Also see

None

lan.autoconnect

This attribute is used to enable or disable link monitoring.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.autoconnect
lan.autoconnect = state
```

state	LAN link monitoring state: 1 or lan.ENABLE: Enables automatic link reconnection and monitoring 0 or lan.DISABLE: Disables automatic link reconnection and monitoring

Details

This attribute sets the LAN link monitoring and automatic connection state.

When this is set to lan.ENABLE, all connections are closed if the link to the LAN is lost for more than the time specified by lan.linktimeout.

Set this attribute to lan.ENABLE to automatically reset the LAN connection after the LAN link is established.

Example

`lan.autoconnect = lan.ENABLE`	Enable LAN link monitoring.

Also see

lan.linktimeout (on page 7-149)
lan.restoredefaults() (on page 7-152)

lan.config.dns.address[N]

Configures DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
dnsAddress = lan.config.dns.address[N]
lan.config.dns.address[N] = "dnsAddress"
```

dnsAddress	DNS server IP address
N	Entry index (1 or 2)

Details

This attribute is an array of Domain Name System (DNS) server addresses. These addresses take priority for DNS lookups and are consulted before any server addresses that are obtained using DHCP. This allows local DNS servers to be specified that take priority over DHCP-configured global DNS servers.

You can specify up to two addresses. The address specified by `1` is consulted first for DNS lookups. *dnsAddress* must be a string specifying the IP address of the DNS server in dotted decimal notation.

Unused entries are returned as `"0.0.0.0"` when read. To disable an entry, set its value to `"0.0.0.0"` or the empty string `""`.

Although only two addresses may be manually specified here, the instrument uses up to three DNS server addresses. If two are specified here, only one that is given by a DHCP server is used. If no entries are specified here, up to three addresses that are given by a DHCP server are used.

Example

```
dnsaddress = "164.109.48.173"
lan.config.dns.address[1] = dnsaddress
```

Set the DNS address 1 to `164.109.48.173`.

Also see

lan.config.dns.domain (on page 7-141)
lan.config.dns.dynamic (on page 7-142)
lan.config.dns.hostname (on page 7-143)
lan.config.dns.verify (on page 7-144)
lan.restoredefaults() (on page 7-152)

lan.config.dns.domain

Configures the dynamic DNS domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	""

Usage

```
domain = lan.config.dns.domain
lan.config.dns.domain = "domain"
```

domain	Dynamic DNS registration domain; use a string of 255 characters or less

Details

This attribute holds the domain to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the domain specified in this attribute with the DNS server.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 255 characters are allowed, you must make sure the combined length is also no more than 255 characters.

Example

```
print(lan.config.dns.domain)
```

Outputs the present dynamic DNS domain. For example, if the domain is "Matrix", the response is:

```
Matrix
```

Also see

lan.config.dns.dynamic (on page 7-142)
lan.config.dns.hostname (on page 7-143)
lan.config.dns.verify (on page 7-144)
lan.restoredefaults() (on page 7-152)

lan.config.dns.dynamic

Enables or disables the dynamic DNS registration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.config.dns.dynamic
lan.config.dns.dynamic = state
```

state	The dynamic DNS registration state. It may be one of the following values: ■ `1` or `lan.ENABLE`: Enabled ■ `0` or `lan.DISABLE`: Disabled

Details

Dynamic DNS registration works with DHCP to register the host name with the DNS server. The host name is specified in the `lan.config.dns.hostname` attribute.

Example

```
print(lan.config.dns.dynamic)
```

Outputs the dynamic registration state.
If dynamic DNS registration is enabled, the response is:
`1.00000e+00`

Also see

lan.config.dns.hostname (on page 7-143)
lan.restoredefaults() (on page 7-152)

lan.config.dns.hostname

This attribute defines the dynamic DNS host name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	Instrument specific (see **Details**)

Usage

```
hostName = lan.config.dns.hostname
lan.config.dns.hostname = "hostName"
```

hostName	The host name to use for dynamic DNS registration; the host name must: ■ be a string of 15 characters or less ■ start with a letter ■ end with a letter or digit ■ contain only letters, digits, and hyphens

Details

This attribute holds the host name to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the host name specified in this attribute with the DNS server.

The factory default value for *hostName* is "k-<model number>-<serial number>", where <model number> and <serial number> are replaced with the actual model number and serial number of the instrument (for example, "k-2651A-1234567"). Note that hyphens separate the characters of *hostName*.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 15 characters can be entered here, you must make sure the combined length is no more than 255 characters.

Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero or a string that consists entirely of whitespace characters) reverts the host name to the factory default value.

Example

```
print(lan.config.dns.hostname)
```

Outputs the present dynamic DNS host name.

Also see

lan.config.dns.domain (on page 7-141)
lan.config.dns.dynamic (on page 7-142)
lan.restoredefaults() (on page 7-152)

lan.config.dns.verify

This attribute defines the DNS host name verification state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.config.dns.verify
lan.config.dns.verify = state
```

state	DNS hostname verification state: ■ 1 or lan.ENABLE: DNS host name verification enabled ■ 0 or lan.DISABLE: DNS host name verification disabled

Details

When this is enabled, the instrument performs DNS lookups to verify that the DNS host name matches the value specified by lan.config.dns.hostname.

Example

```
print(lan.config.dns.verify)
```

Outputs the present DNS host name verification state.
If it is enabled, the output is:
1.00000e+00

Also see

lan.config.dns.hostname (on page 7-143)
lan.restoredefaults() (on page 7-152)

lan.config.duplex

This attribute defines the LAN duplex mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.FULL)

Usage

```
duplex = lan.config.duplex
lan.config.duplex = duplex
```

duplex	LAN duplex setting can be one of the following values: ■ 1 or lan.FULL: Selects full-duplex operation ■ 0 or lan.HALF: Selects half-duplex operation

Details

This attribute does not indicate the actual setting currently in effect. Use the `lan.status.duplex` attribute to determine the present operating state of the LAN.

Example

`lan.config.duplex = lan.FULL`	Set the LAN duplex mode to full.

Also see

lan.restoredefaults() (on page 7-152)

lan.config.gateway

This attribute contains the LAN default gateway address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
gatewayAddress = lan.config.gateway
lan.config.gateway = "gatewayAddress"
```

gatewayAddress	LAN default gateway address; must be a string specifying the default IP address of the gateway in dotted decimal notation

Details

This attribute specifies the default gateway IP address to use when manual or DLLA configuration methods are used to configure the LAN. If DHCP is enabled, this setting is ignored.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.gateway` attribute to determine the present operating state of the LAN.

The IP address must be formatted in four groups of numbers, each separated by a decimal.

Example

```
print(lan.config.gateway)
```

Outputs the default gateway address. For example, you might see the output:
`192.168.0.1`

Also see

lan.restoredefaults() (on page 7-152)
lan.status.gateway (on page 7-155)

lan.config.ipaddress

This command specifies the LAN IP address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"192.168.0.2"

Usage

```
ipAddress = lan.config.ipaddress
lan.config.ipaddress = "ipAddress"
```

ipAddress	LAN IP address; must be a string specifying the IP address in dotted decimal notation

Details

This command specifies the LAN IP address to use when the LAN is configured using the manual configuration method. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.ipaddress` attribute to determine the present operating state of the LAN.

Example

```
ipaddress = lan.config.ipaddress
```

Retrieves the presently set LAN IP address.

Also see

lan.restoredefaults() (on page 7-152)
lan.status.ipaddress (on page 7-156)

lan.config.method

This attribute contains the LAN settings configuration method.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0 (lan.AUTO)

Usage

```
method = lan.config.method
lan.config.method = method
```

method	The method for configuring LAN settings; it can be one of the following values: ■ `0` or `lan.AUTO`: Selects automatic sequencing of configuration methods ■ `1` or `lan.MANUAL`: Use only manually specified configuration settings

Details

This attribute controls how the LAN IP address, subnet mask, default gateway address, and DNS server addresses are determined.

When method is `lan.AUTO`, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, it uses the manually specified settings.

When method is `lan.MANUAL`, only the manually specified settings are used. Neither DHCP nor DLLA are attempted.

Example

```
print(lan.config.method)
```

Outputs the present method.
For example:
`1.00000e+00`

Also see

lan.restoredefaults() (on page 7-152)

lan.config.speed

This attribute contains the LAN speed used when restarting in manual configuration mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	100 (100 Mbps)

Usage

```
speed = lan.config.speed
lan.config.speed = speed
```

speed	LAN speed setting in Mbps (10 or 100)

Details

This attribute stores the speed that is used if the LAN is restarted for manual configuration operation.

This attribute does not indicate the actual setting presently in effect. Use the `lan.status.speed` attribute to determine the present operating state of the LAN.

The LAN speed is measured in megabits per second (Mbps).

Example

`lan.config.speed = 100`	Configure LAN speed for 100.

Also see

lan.restoredefaults() (on page 7-152)
lan.status.speed (on page 7-159)

lan.config.subnetmask

This attribute contains the LAN subnet mask.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"255.255.255.0"

Usage

```
mask = lan.config.subnetmask
lan.config.subnetmask = "mask"
```

mask	String that specifies the LAN subnet mask value in dotted decimal notation

Details

This attribute specifies the LAN subnet mask that is used when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting presently in effect. Use the `lan.status.subnetmask` attribute to determine the present operating state of the LAN.

Example

```
print(lan.config.subnetmask)
```

Outputs the LAN subnet mask, such as:
`255.255.255.0`

Also see

lan.restoredefaults() (on page 7-152)
lan.status.subnetmask (on page 7-159)

lan.linktimeout

This attribute contains the LAN link timeout period.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	20 (20 s)

Usage

```
timeout = lan.linktimeout
lan.linktimeout = timeout
```

timeout	The LAN link monitor time-out period (in seconds)

Details

You must enable the command `lan.autoconnect` before you can use this attribute.

The *timeout* value represents the amount of time that passes before the instrument disconnects from the LAN due to the loss of the LAN link integrity.

The LAN interface does not disconnect if the connection to the LAN is reestablished before the *timeout* value expires.

If the LAN link integrity is not restored before the *timeout* value expires, the instrument begins to monitor for a new connection.

Example

`print(lan.linktimeout)`	Outputs the present LAN link timeout setting.

Also see

lan.autoconnect (on page 7-139)
lan.restoredefaults() (on page 7-152)

lan.lxidomain

This attribute contains the LXI domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0

Usage

```
domain = lan.lxidomain
lan.lxidomain = domain
```

domain	The LXI domain number (0 to 255)

Details

This attribute sets the LXI domain number.

All outgoing LXI packets are generated with this domain number. All inbound LXI packets are ignored unless they have this domain number.

Example

```
print(lan.lxidomain)
```

Displays the LXI domain.

Also see

None

lan.nagle

This attribute controls the state of the LAN Nagle algorithm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (lan.DISABLE)

Usage

```
state = lan.nagle
lan.nagle = state
```

state	The state of the Nagle algorithm: ■ `1` or `lan.ENABLE`: Enable the LAN Nagle algorithm for TCP connections ■ `0` or `lan.DISABLE`: Disable the Nagle algorithm for TCP connections

Details

This attribute enables or disables the use of the LAN Nagle algorithm on transmission control protocol (TCP) connections.

Also see

lan.restoredefaults() (on page 7-152)

lan.reset()

This function resets the LAN interface.

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

Usage

```
lan.reset()
```

Details

This function resets the LAN interface. It performs the commands `lan.restoredefaults()` and `lan.applysettings()`. It also resets the LAN password.

Also see

lan.applysettings() (on page 7-138)
lan.restoredefaults() (on page 7-152)
localnode.password (on page 7-175)

lan.restoredefaults()

This function resets LAN settings to default values.

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

Usage

```
lan.restoredefaults()
```

Details

The settings that are restored are shown in the following table.

Settings that are restored to default	
Attribute	**Default setting**
lan.autoconnect	lan.ENABLE
lan.config.dns.address[N]	"0.0.0.0"
lan.config.dns.domain	""
lan.config.dns.dynamic	lan.ENABLE
lan.config.dns.verify	lan.ENABLE
lan.config.duplex	lan.FULL
lan.config.gateway	"0.0.0.0"
lan.config.ipaddress	"192.168.0.2"
lan.config.method	lan.AUTO
lan.config.speed	100
lan.config.subnetmask	"255.255.255.0"
lan.linktimeout	20 (seconds)
lan.lxidomain	0
lan.nagle	lan.DISABLE
lan.timedwait	20 (seconds)

The `lan.restoredefaults()` function does not reset the LAN password. The `localnode.password` attribute controls the web password, which can be reset separately.

This command is run when `lan.reset()` is sent.

Example

```
lan.restoredefaults()
```

Restores the LAN defaults.

Also see

lan.reset() (on page 7-151)
localnode.password (on page 7-175)

lan.status.dns.address[N]

This attribute contains the DNS server IP addresses.

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

Usage

```
dnsAddress = lan.status.dns.address[N]
```

dnsAddress	DNS server IP address
N	Entry index (1, 2, or 3)

Details

This attribute is an array of DNS server addresses. The instrument can use up to three addresses.

Unused or disabled entries are returned as `"0.0.0.0"` when read. The *dnsAddress* returned is a string specifying the IP address of the DNS server in dotted decimal notation.

You can only specify two addresses manually. However, the instrument uses up to three DNS server addresses. If two are specified, only the one given by a DHCP server is used. If no entries are specified, up to three address given by a DHCP server are used.

The value of `lan.status.dns.address[1]` is referenced first for all DNS lookups. The values of `lan.status.dns.address[2]` and `lan.status.dns.address[3]` are referenced second and third, respectively.

Example

```
print(lan.status.dns.address[1])
```

Outputs DNS server address 1, for example:

```
164.109.48.173
```

Also see

lan.status.dns.name (on page 7-154)

lan.status.dns.name

This attribute contains the present DNS fully qualified host name.

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

Usage

```
hostName = lan.status.dns.name
```

hostName	Fully qualified DNS host name that can be used to connect to the instrument

Details

A fully qualified domain name (FQDN) specifies its exact location in the tree hierarchy of the Domain Name System (DNS).

A FQDN is the complete domain name for a specific computer or host on the LAN. The FQDN consists of two parts: The host name and the domain name.

If the DNS host name for an instrument is not found, this attribute stores the IP address in dotted decimal notation.

Example

```
print(lan.status.dns.name)
```

Outputs the dynamic DNS host name.

Also see

lan.config.dns.address[N] (on page 7-140)
lan.config.dns.hostname (on page 7-143)

lan.status.duplex

This attribute contains the duplex mode presently in use by the LAN interface.

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

Usage

```
duplex = lan.status.duplex
```

duplex	LAN duplex setting can be one of the following values: ■ `0` or `lan.HALF`: half-duplex operation ■ `1` or `lan.FULL`: full-duplex operation

Example

```
print(lan.status.duplex)
```

Outputs the present LAN duplex mode, such as:

```
1.00000e+00
```

Also see

None

lan.status.gateway

This attribute contains the gateway address presently in use by the LAN interface.

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

Usage

```
gatewayAddress = lan.status.gateway
```

gatewayAddress	LAN gateway address presently being used

Details

The value of *gatewayAddress* is a string that indicates the IP address of the gateway in dotted decimal notation.

Example

```
print(lan.status.gateway)
```

Outputs the gateway address, such as:

```
192.168.0.1
```

Also see

lan.config.gateway (on page 7-145)

lan.status.ipaddress

This attribute contains the LAN IP address presently in use by the LAN interface.

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

Usage

```
ipAddress = lan.status.ipaddress
```

ipAddress	LAN IP address specified in dotted decimal notation

Details

The IP address is a character string that represents the IP address assigned to the instrument.

Example

```
print(lan.status.ipaddress)
```

Outputs the LAN IP address currently in use, such as:

```
192.168.0.2
```

Also see

lan.config.ipaddress (on page 7-146)

lan.status.macaddress

This attribute contains the LAN MAC address.

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

Usage

```
macAddress = lan.status.macaddress
```

macAddress	The instrument MAC address

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets (see Example).

Example

```
print(lan.status.macaddress)
```

Outputs the MAC address of the instrument, for example:

```
08:00:11:00:00:57
```

Also see

None

lan.status.port.dst

This attribute contains the LAN dead socket termination port number.

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

Usage

```
port = lan.status.port.dst
```

port	Dead socket termination socket port number

Details

This attribute holds the TCP port number used to reset all other LAN socket connections.

To reset all LAN connections, open a connection to the DST port number.

Example

```
print(lan.status.port.dst)
```

Outputs the LAN dead socket termination port number, such as:

```
5.03000e+03
```

Also see

None

lan.status.port.rawsocket

This attribute contains the LAN raw socket connection port number.

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

Usage

```
port = lan.status.port.rawsocket
```

port	Raw socket port number

Details

The TCP port number used to connect the instrument and to control the instrument over a raw socket communication interface.

Example

```
print(lan.status.port.rawsocket)
```

Outputs the LAN raw socket port number, such as:

```
5.02500e+03
```

Also see

None

lan.status.port.telnet

This attribute contains the LAN Telnet connection port number.

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

Usage

```
port = lan.status.port.telnet
```

port	Telnet port number

Details

This attribute holds the TCP port number used to connect to the instrument to control it over a Telnet interface.

Example

```
print(lan.status.port.telnet)
```

Get the LAN Telnet connection port number.
Output:

```
2.30000e+01
```

Also see

None

lan.status.port.vxi11

This attribute contains the LAN VXI-11 connection port number.

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

Usage

```
port = lan.status.port.vxi11
```

port	LAN VXI-11 port number

Details

This attribute stores the TCP port number used to connect to the instrument over a VXI-11 interface.

Example

```
print(lan.status.port.vxi11)
```

Outputs the VXI-11 number, such as:

```
1.02400e+03
```

Also see

None

lan.status.speed

This attribute contains the LAN speed.

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

Usage

```
speed = lan.status.speed
```

speed	LAN speed in Mbps, either 10 or 100

Details

This attribute indicates the transmission speed currently in use by the LAN interface.

Example

```
print(lan.status.speed)
```

Outputs the transmission speed of the instrument presently in use, such as:

```
1.00000e+02
```

Also see

None

lan.status.subnetmask

This attribute contains the LAN subnet mask that is presently in use by the LAN interface.

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

Usage

```
mask = lan.status.subnetmask
```

mask	A string specifying the subnet mask in dotted decimal notation

Details

Use this attribute to determine the present operating state of the LAN. This attribute returns the present LAN subnet mask value if the LAN is manually configured, or when DLLA or DHCP is used.

Example

```
print(lan.status.subnetmask)
```

Outputs the subnet mask of the instrument that is presently in use, such as:

```
255.255.255.0
```

Also see

lan.config.subnetmask (on page 7-149)

lan.timedwait

This attribute contains the LAN timed-wait state interval.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	20 (20 s)

Usage

```
timeout = lan.timedwait
lan.timedwait = timeout
```

timeout	The LAN timed-wait state interval in seconds

Details

This attribute controls the amount of time that resources are allocated to closed TCP connections. When a TCP connection is closed, the connection is put in a timed-wait state and resources remain allocated for the connection until the timed-wait state ends. During the timed-wait interval, the instrument processes delayed packets that arrive after the connection is closed.

Use this attribute to tailor the timed-wait state interval for the instrument.

Example

`lan.timedwait = 30`	Set the amount of time resources are allocated to TCP connection to 30 s.

Also see

lan.restoredefaults() (on page 7-152)

lan.trigger[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event ID.

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

Usage

```
lan.trigger[N].assert()
```

N	The LAN event number (1 to 8)

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudo line state to the appropriate state.

The following indexes provide the listed LXI events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- …
- 8:LAN7

Example

```
lan.trigger[5].assert()
```

Creates a trigger with LAN packet 5.

Also see

lan.lxidomain (on page 7-150)
lan.trigger[N].clear() (on page 7-161)
lan.trigger[N].mode (on page 7-166)
lan.trigger[N].overrun (on page 7-167)
lan.trigger[N].stimulus (on page 7-170)
lan.trigger[N].wait() (on page 7-171)
Understanding hardware value and pseudo line state (on page 3-58)

lan.trigger[N].clear()

This function clears the event detector for a LAN trigger.

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

Usage

```
lan.trigger[N].clear()
```

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

```
lan.trigger[5].clear()
```

Clears the event detector with LAN packet 5.

Also see

lan.trigger[N].assert() (on page 7-161)
lan.trigger[N].overrun (on page 7-167)
lan.trigger[N].stimulus (on page 7-170)
lan.trigger[N].wait() (on page 7-171)

lan.trigger[N].connect()

This function prepares the event generator for outgoing trigger events.

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

Usage

```
lan.trigger[N].connect()
```

N	The LAN event number (1 to 8)

Details

This command prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event is changed.

Example

```
lan.trigger[1].protocol = lan.MULTICAST
lan.trigger[1].connect()
lan.trigger[1].assert()
```

Set the protocol for LAN trigger 1 to be multicast when sending LAN triggers. Then, after connecting the LAN trigger, send a message on LAN trigger 1 by asserting it.

Also see

lan.trigger[N].assert() (on page 7-161)
lan.trigger[N].ipaddress (on page 7-165)
lan.trigger[N].overrun (on page 7-167)
lan.trigger[N].protocol (on page 7-168)
lan.trigger[N].stimulus (on page 7-170)
lan.trigger[N].wait() (on page 7-171)

lan.trigger[N].connected

This attribute stores the LAN event connection state.

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

Usage

```
connected = lan.trigger[N].connected
```

connected	The LAN event connection state: ■ `true`: Connected ■ `false`: Not connected
N	The LAN event number (1 to 8)

Details

This read-only attribute is set to `true` when the LAN trigger is connected and ready to send trigger events following a successful `lan.trigger[N].connect()` command; if the LAN trigger is not ready to send trigger events, this value is `false`.

This attribute is also `false` when either `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes are changed or the remote connection closes the connection.

Example

```
lan.trigger[1].protocol = lan.MULTICAST
print(lan.trigger[1].connected)
```

Outputs `true` if connected, or `false` if not connected.
Example output:

```
false
```

Also see

lan.trigger[N].connect() (on page 7-162)
lan.trigger[N].ipaddress (on page 7-165)
lan.trigger[N].protocol (on page 7-168)

lan.trigger[N].disconnect()

This function disconnects the LAN trigger.

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

Usage

```
lan.trigger[N].disconnect()
```

N	The LAN event number (1 to 8)

Details

For TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes for this event are changed.

Also see

lan.trigger[N].ipaddress (on page 7-165)
lan.trigger[N].protocol (on page 7-168)

lan.trigger[N].EVENT_ID

This constant is the event identifier used to route the LAN trigger to other subsystems (using stimulus properties).

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

Usage

```
lan.trigger[N].EVENT_ID
```

N	The LAN event number (1 to 8)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to incoming LAN trigger packets.

Example

```
digio.trigger[14].stimulus = lan.trigger[1].EVENT_ID
```

Route occurrences of triggers on LAN trigger 1 to digital I/O trigger 14.

Also see

None

lan.trigger[N].ipaddress

This attribute specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Not saved	"0.0.0.0"

Usage

```
ipAddress = lan.trigger[N].ipaddress
lan.trigger[N].ipaddress = "ipAddress"
```

ipAddress	The LAN address for this attribute as a string in dotted decimal notation
N	The LAN event number (1 to 8)

Details

Sets the IP address for outgoing trigger events.

Set to `"0.0.0.0"` for multicast.

After changing this setting, the `lan.trigger[N].connect()` command must be called before outgoing messages can be sent.

Example

```
lan.trigger[3].protocol = lan.TCP
lan.trigger[3].ipaddress = "192.168.1.100"
lan.trigger[3].connect()
```

Set the protocol for LAN trigger 3 to be `lan.TCP` when sending LAN triggers.
Use IP address `"192.168.1.100"` to connect the LAN trigger.

Also see

lan.trigger[N].connect() (on page 7-162)

lan.trigger[N].mode

This attribute sets the trigger operation and detection mode of the specified LAN event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Not saved	0 (lan.TRIG_EITHER)

Usage

```
mode = lan.trigger[N].mode
lan.trigger[N].mode = mode
```

mode	A number representing the trigger mode (0 to 7); see the **Details** section for more information
N	A number representing the LAN event number (1 to 8)

Details

This command controls how the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values

Mode	Number	Trigger packets detected as input	LAN trigger packet generated for output with a…
lan.TRIG_EITHER	0	Rising or falling edge (positive or negative state)	negative state
lan.TRIG_FALLING	1	Falling edge (negative state)	negative state
lan.TRIG_RISING	2	Rising edge (positive state)	positive state
lan.TRIG_RISINGA	3	Rising edge (positive state)	positive state
lan.TRIG_RISINGM	4	Rising edge (positive state)	positive state
lan.TRIG_SYNCHRONOUS	5	Falling edge (negative state)	positive state
lan.TRIG_SYNCHRONOUSA	6	Falling edge (negative state)	positive state
lan.TRIG_SYNCHRONOUSM	7	Rising edge (positive state)	negative state

lan.TRIG_RISING and lan.TRIG_RISINGA are the same.

lan.TRIG_RISING and lan.TRIG_RISINGM are the same.

Use of either lan.TRIG_SYNCHRONOUSA or lan.TRIG_SYNCHRONOUSM instead of lan.TRIG_SYNCHRONOUS is preferred. Use of lan.TRIG_SYNCHRONOUS is provided for compatibility with older products and other Keithley Instruments products.

Example

```
print(lan.trigger[1].mode)
```

Outputs the present LAN trigger mode of LAN event 1.

Also see

Digital I/O (on page 3-92)
TSP-Link system expansion interface (on page 6-56)

lan.trigger[N].overrun

This attribute contains the overrun status of the LAN event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger N clear LAN trigger N reset Instrument reset Recall setup	Not applicable	Not applicable

Usage

```
overrun = lan.trigger[N].overrun
```

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. Output trigger overrun indications are provided in the status model.

Example

```
overrun = lan.trigger[5].overrun
print(overrun)
```

Checks the overrun status of a trigger on LAN5 and outputs the value, such as:

```
false
```

Also see

lan.trigger[N].assert() (on page 7-161)
lan.trigger[N].clear() (on page 7-161)
lan.trigger[N].stimulus (on page 7-170)
lan.trigger[N].wait() (on page 7-171)

lan.trigger[N].protocol

This attribute sets the LAN protocol to use for sending trigger messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Not saved	0 (lan.TCP)

Usage

```
protocol = lan.trigger[N].protocol
lan.trigger[N].protocol = protocol
```

protocol	The protocol to use for messages from the trigger: ■ 0 or lan.TCP ■ 1 or lan.UDP ■ 2 or lan.MULTICAST
N	The LAN event number (1 to 8)

Details

The LAN trigger listens for trigger messages on all supported protocols, but uses the designated protocol for sending outgoing messages. After changing this setting, `lan.trigger[N].connect()` must be called before outgoing event messages can be sent.

When the `lan.MULTICAST` protocol is selected, the `lan.trigger[N].ipaddress` attribute is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

```
print(lan.trigger[1].protocol)
```

Get LAN protocol to use for sending trigger messages for LAN event 1.

Also see

lan.trigger[N].connect() (on page 7-162)
lan.trigger[N].ipaddress (on page 7-165)

lan.trigger[N].pseudostate

This attribute sets the simulated line state for the LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Not saved	1

Usage

```
pseudostate = lan.trigger[N].pseudostate
lan.trigger[N].pseudostate = pseudostate
```

pseudostate	The simulated line state (`0` or `1`)
N	A number representing the LAN event number (1 to 8)

Details

This attribute can be set to initialize the pseudo line state to a known value.

Setting this attribute does not cause the LAN trigger to generate any events or output packets.

Example

```
print(lan.trigger[1].pseudostate)
```

Get the present simulated line state for the LAN event 1.

Also see

None

lan.trigger[N].stimulus

This attribute specifies events that cause this trigger to assert.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Not saved	0

Usage

```
triggerStimulus = lan.trigger[N].stimulus
lan.trigger[N].stimulus = triggerStimulus
```

triggerStimulus	The LAN event identifier used to trigger the event
N	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set *triggerStimulus* to one of the trigger event IDs, which are shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smu*X*.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[*N*].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[*N*].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[*N*].EVENT_ID	Occurs when a delay expires

Setting this attribute to zero disables automatic trigger generation.

If any events are detected prior to calling lan.trigger[*N*].connect(), the event is ignored and the action overrun is set.

Example

```
lan.trigger[5].stimulus = trigger.timer[1].EVENT_ID
```

Use timer 1 trigger event as the source for LAN packet 5 trigger stimulus.

Also see

lan.trigger[N].assert() (on page 7-161)
lan.trigger[N].clear() (on page 7-161)
lan.trigger[N].connect() (on page 7-162)
lan.trigger[N].overrun (on page 7-167)
lan.trigger[N].wait() (on page 7-171)

lan.trigger[N].wait()

This function waits for an input trigger.

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

Usage

triggered = lan.trigger[*N*].wait(*timeout*)

triggered	Trigger detection indication (`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 `lan.trigger[N].wait()` or `lan.trigger[N].clear()` was called, this function returns immediately.

After waiting for a LAN trigger event with this function, the event detector is automatically reset and rearmed regardless of the number of events detected.

Example

```
triggered = lan.trigger[5].wait(3)
```

Wait for a trigger with LAN packet 5 with a timeout of 3 seconds.

Also see

lan.trigger[N].assert() (on page 7-161)
lan.trigger[N].clear() (on page 7-161)
lan.trigger[N].overrun (on page 7-167)
lan.trigger[N].stimulus (on page 7-170)

localnode.autolinefreq

This attribute enables or disables automatic power line frequency detection at start-up.

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

Usage

```
flag = localnode.autolinefreq
localnode.autolinefreq = flag
```

flag	The auto line frequency detection setting: ■ `true`: Enable automatic line frequency detection at start-up ■ `false`: Disable automatic line frequency detection at start-up

Details

When this attribute is set to `true`, the power line frequency is detected automatically the next time the Model 2651A powers up. After the power line frequency is automatically detected at power-up, the `localnode.linefreq` attribute is set automatically to `50` or `60`.

If the `localnode.linefreq` attribute is explicitly set, `localnode.autolinefreq` is automatically set to `false`.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].autolinefreq`.

Also see

localnode.linefreq (on page 7-174)

localnode.description

This attribute stores a user-defined description and mDNS service name of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	Instrument specific (see **Details**)

Usage

```
localnode.description = "description"
description = localnode.description
```

description	User-defined description and mDNS service name of the instrument; use a string of 63 characters or less

Details

This attribute stores a string that contains a description of the instrument. This value appears on LXI welcome page of the instrument. The value of this attribute is also used as the mDNS service name of the instrument.

The factory default value of this attribute is "Keithley Instruments SMU <model number> - <serial number>", where <model number> and <serial number> are replaced with the actual model number and serial number of the instrument (for example, "Keithley Instruments SMU 2651A - 1349810"). Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero or a string consisting entirely of whitespace characters) reverts the description to the factory default value.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].description`.

Example

```
description = "System in Lab 05"
localnode.description = description
```

Set `description` to `System in Lab 05`.

Also see

None

localnode.license

This attribute returns the product license agreements.

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

Usage

```
license_agreement = localnode.license
```

license_agreement	The text of the license agreements

Example

```
print(localnode.license)
```

Returns the license agreements for the Model 2651A.

Also see

None

localnode.linefreq

This attribute contains the power line frequency setting that is used for NPLC calculations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	60 (60 Hz)

Usage

```
frequency = localnode.linefreq
localnode.linefreq = frequency
```

frequency	An integer representing the detected or specified line frequency of the instrument

Details

To achieve optimum noise rejection when performing measurements at integer NPLC apertures, set the line frequency attribute to match the frequency (50 Hz or 60 Hz) of the ac power line.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].linefreq`.

When this attribute is set, the `localnode.autolinefreq` attribute is automatically set to `false`. You can have the instrument automatically detect the ac power line frequency and set this attribute with the line frequency detected when the instrument power is turned on by setting the `localnode.autolinefreq` attribute to `true`.

Example 1

`frequency = localnode.linefreq`	Reads line frequency setting.

Example 2

`localnode.linefreq = 60`	Sets the line frequency to 60 Hz.

Also see

localnode.autolinefreq (on page 7-172)

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: `2657A`

Also see

localnode.serialno (on page 7-180)

localnode.password

This attribute stores the remote access password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	LAN reset LAN restore defaults	Nonvolatile memory	""

Usage

localnode.password = "*password*"

password	A string that contains the remote interface password, up to 255 characters

Details

This write-only attribute stores the password that is set for any remote interface. When password usage is enabled (`localnode.passwordmode`), you must supply a password to change the configuration or to control an instrument from a remote command interface.

The instrument continues to use the old password for all interactions until the command to change it executes. When changing the password, give the instrument time to execute the command before attempting to use the new password.

You cannot retrieve a lost password from any command interface.

You can reset the password by resetting the LAN from the front panel or by sending the `lan.reset()` command.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].password`.

Example

```
localnode.password = "N3wpa55w0rd"
```

Changes the remote interface password to `N3wpa55w0rd`.

Also see

lan.reset() (on page 7-151)
localnode.passwordmode (on page 7-176)

localnode.passwordmode

This attribute stores the password enable mode for remote access to the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	1 (localnode.PASSWORD_WEB)

Usage

```
mode = localnode.passwordmode
localnode.passwordmode = mode
```

mode	The remote password enable mode

Details

This attribute controls if and where remote access passwords are required. Set this attribute to one of the values below to enable password checking:

- `localnode.PASSWORD_NONE` or `0`: Disable passwords everywhere
- `localnode.PASSWORD_WEB` or `1`: Use passwords on the web interface only
- `localnode.PASSWORD_LAN` or `2`: Use passwords on the web interface and all LAN interfaces
- `localnode.PASSWORD_ALL` or `3`: Use passwords on the web interface and all remote command interfaces

NOTE

When a password is set for the web interface, you cannot make changes using the web interface options Reading Buffers, Flash Upgrade, or TSB Embedded.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].passwordmode`.

If you enable password mode, you must also assign a password.

Example

`mode = localnode.PASSWORD_WEB`
`localnode.passwordmode = mode`
`localnode.password = "SMU1234"`
	Sets value of `mode` to `PASSWORD_WEB`.
	Allows use of passwords on the web interface only.
	Set the password to `SMU1234`.

Also see

localnode.password (on page 7-175)

localnode.prompts

This attribute determines if the instrument generates prompts in response to command messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (disabled)

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting	Prompting mode: ■ Do not generate prompts: `1` ■ Generate prompts: `0`

Details

When the prompting mode is enabled, the instrument generates prompts when the instrument is ready to take another command. Because the prompt is not generated until the previous command completes, enabling prompts provides handshaking with the instrument to prevent buffer overruns.

When prompting is enabled, the instrument might generate the following prompts:

- **TSP>.** The standard prompt, which indicates that the previous command completed normally.
- **TSP?.** The prompt that is issued if there are unread entries in the error queue when the prompt is issued. Like the TSP> prompt, it indicates that processing of the command is complete. It does not mean the previous command generated an error, only that there were still errors in the queue when the command processing was complete.
- **>>>>.** The continuation prompt, which occurs when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the present command.

Commands do not generate prompts. The instrument generates prompts in response to command completion.

Prompts are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable prompts when the LAN connection is active, they are not enabled for a subsequent USB connection.

NOTE

Do not disable prompting when using Test Script Builder. Test Script Builder requires prompts and sets the prompting mode automatically. If you disable prompting, the instrument stops responding when you communicate using Test Script Builder because it is waiting for a common complete prompt from Test Script Builder.

Example

```
localnode.prompts = 1
```

Enable prompting.

Also see

localnode.prompts4882 (on page 7-178)
localnode.showerrors (on page 7-181)
tsplink.reset() (on page 7-426)

localnode.prompts4882

This attribute enables and disables the generation of prompts for IEEE Std 488.2 common commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	1 (enabled)

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

prompting	IEEE Std 488.2 prompting mode: ■ Disable prompting: `0` ■ Enable prompting: `1`

Details

When this attribute is enabled, the IEEE Std 488.2 common commands generate prompts if prompting is enabled with the `localnode.prompts` attribute. If `localnode.prompts4882` is enabled, limit the number of `*trg` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When this attribute is disabled, IEEE Std 488.2 common commands do not generate prompts. When using the `*trg` command with a script that executes `trigger.wait()` repeatedly, disable prompting to avoid problems associated with the command interface input queue filling.

Example

```
localnode.prompts4882 = 0
```

Disables IEEE Std 488.2 common command prompting.

Also see

localnode.prompts (on page 7-177)

localnode.reset()

This function resets the local node instrument.

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

Usage

```
localnode.reset()
```

Details

If you want to reset a specific instrument or a subordinate node, use the `node[X].reset()` command.

A local node reset includes:

- Source-measure unit (SMU) attributes affected by a SMU reset are reset
- Other settings are restored back to factory default settings

A `localnode.reset()` is different than a `reset()` because `reset()` resets the entire system.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].reset()`.

Example

```
localnode.reset()
```

Resets the local node.

Also see

reset() (on page 7-197)
smuX.reset() (on page 7-264)

localnode.revision

This attribute stores the firmware revision level.

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

Usage

```
revision = localnode.revision
```

revision	Firmware revision level

Details

This attribute indicates the revision number of the firmware that is presently running in the instrument.

When using this command from a remote node, replace `localnode` with the node reference. For example, `node[5].revision`.

Example

```
print(localnode.revision)
```

Outputs the present revision level.
Sample output:
`1.1.0`

Also see

localnode.description (on page 7-173)
localnode.model (on page 7-175)
localnode.serialno (on page 7-180)

localnode.serialno

This attribute stores the serial number of the instrument.

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

Usage

```
serialno = localnode.serialno
```

serialno	The serial number of the instrument

Details

This indicates the instrument serial number.

Example

```
display.clear()
display.settext(localnode.serialno)
```

Clears the instrument display.
Places the serial number of the instrument on the top line of its display.

Also see

localnode.description (on page 7-173)
localnode.model (on page 7-175)
localnode.revision (on page 7-180)

localnode.showerrors

This attribute sets whether or not the instrument automatically sends generated errors.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (disabled)

Usage

```
errorMode = localnode.showerrors
localnode.showerrors = errorMode
```

errorMode	Show error setting: ■ Show errors: `1` ■ Do not show errors: `0`

Details

If this attribute is set to `1`, the instrument automatically sends any generated errors stored in the error queue, and then clears the queue. Errors are processed after executing a command message (just before issuing a prompt if prompts are enabled).

If this attribute is set to `0`, errors are left in the error queue and must be explicitly read or cleared.

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].showerrors`.

Example

```
localnode.showerrors = 1
```

Enables sending of generated errors.

Also see

localnode.prompts (on page 7-177)

makegetter()

This function creates a function to get the value of an attribute.

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

Usage

```
getter = makegetter(table, "attributeName")
```

getter	The return value
table	Read-only table where the attribute is located
attributeName	A string representing the name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the function created with `makegetter()` executes more quickly than accessing the attribute directly.

Creating a getter function is only useful if it is going to be called several times. Otherwise, the overhead of creating the getter function outweighs the overhead of accessing the attribute directly.

Example

```
getlevel = makegetter(smua.source, "levelv")
v = getlevel()
```

Creates a getter function called `getlevel`.
When `getlevel()` is called, it returns the value of `smua.source.levelv`.

Also see

makesetter() (on page 7-183)

makesetter()

This function creates a function that, when called, sets the value of an attribute.

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

Usage

```
setter = makesetter(table, "attributeName")
```

setter	Function that sets the value of the attribute
table	Read-only table where the attribute is located
attributeName	The string name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the *setter* function execute more quickly than accessing the attribute directly.

Creating a *setter* function is only useful if it is going to be called several times. If you are not calling the *setter* function several times, it is more efficient to access the attribute directly.

Example

```
setlevel = makesetter(smua.source, "levelv")
for v = 1, 10 do
   setlevel(v)
end
```

Creates a setter function called `setlevel`.
Using `setlevel()` in the loop sets the value of `smua.source.levelv`, performing a source sweep.

Also see

makegetter() (on page 7-182)

meminfo()

This function returns the present amount of available memory and the total amount of memory in the instrument.

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

Usage

```
freeMem, totalMem = meminfo()
```

freeMem	The amount of free dynamically allocated memory available
totalMem	The total amount of dynamically allocated memory in the instrument

Details

This function returns two values:

- The amount of free dynamically allocated memory available in kilobytes
- The total amount of dynamically allocated memory on the instrument in kilobytes

The difference between the two values is the amount presently used.

Example

`print(meminfo())`	Retrieve the amount of free and total memory in the instrument. Output: `2.89840e+04  3.27680e+04`

Also see

None

node[N].execute()

This function starts test scripts on a remote TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
node[N].execute("scriptCode")
```

N	The node number of this instrument (1 to 63)
scriptCode	A string containing the source code

Details

This command is only applicable to TSP-Link systems. You can use this command to use the remote master node to run a script on the specified node. This function does not run test scripts on the master node; only on the subordinate node when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

Example 1

```
node[2].execute(sourcecode)
```

Runs script code on node 2. The code is in a string variable called `sourcecode`.

Example 2

```
node[3].execute("x = 5")
```

Runs script code in string constant (`x = 5`) to set `x` equal to 5 on node 3.

Example 3

```
node[32].execute(TestDut.source)
```

Runs the test script stored in the variable `TestDut` (previously stored on the master node) on node 32.

Also see

TSP advanced features (on page 6-61)
tsplink.group (on page 7-421)

node[N].getglobal()

This function returns the value of a global variable.

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

Usage

```
value = node[N].getglobal("name")
```

value	The value of the variable
N	The node number of this instrument (1 to 64)
name	The global variable name

Details

This function retrieves the value of a global variable from the runtime environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

```
print(node[5].getglobal("test_val"))
```

Retrieves and outputs the value of the global variable named `test_val` from node 5.

Also see

node[N].setglobal() (on page 7-187)
TSP advanced features (on page 6-61)

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 64)
name	The global variable name to set
value	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

```
node[3].setglobal("x", 5)
```

Sets the global variable `x` on node 3 to the value of 5.

Also see

node[N].getglobal() (on page 7-186)
TSP advanced features (on page 6-61)

opc()

This function sets the operation complete status bit when all overlapped commands are completed.

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

Usage

```
opc()
```

Details

This function causes the operation complete bit in the Standard Event Status Register to be set when all previously started local overlapped commands are complete.

Note that each node independently sets its operation complete bits in its own status model. Any nodes that are not actively performing overlapped commands set their bits immediately. All remaining nodes set their own bits as they complete their own overlapped commands.

Example

```
opc()
waitcomplete()
print("1")
```

Output:
```
1
```

Also see

Status model (on page 15-1)
waitcomplete() (on page 7-459)

os.remove()

This function deletes the file or directory with a given name.

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

Usage

```
success, msg = os.remove("filename")
```

success	A success indicator (`true` or `nil`)
msg	A message value (`nil` or an error message)
filename	A string representing the name of the file or directory to delete

Details

Directories must be empty before using the `os.remove()` function to delete them.

If this function fails, it returns `nil` (for *success*) and an error message string (for *msg*).

Example

`os.remove("testFile")`	Delete the file named `testFile`.

Also see

os.rename() (on page 7-189)

os.rename()

This function renames an existing file or directory.

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

Usage

```
success, msg = os.rename("oldname", "newname")
```

success	A success indicator (`true` or `nil`)
msg	A message value (`nil` or an error message)
oldname	String representing the name of the file or directory to rename
newname	String representing the new name of the file or directory

Details

If this function fails, it returns `nil` (for *success*) and an error message string (for *msg*).

Example

`os.rename("testFile", "exampleFile")`	Changes the name of the existing file `testFile` to the name `exampleFile`.

Also see

os.remove() (on page 7-188)

os.time()

This function generates a time value in UTC time.

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

Usage

```
utcTime = os.time()
utcTime = os.time(timespec)
```

utcTime	Time value in UTC time
timespec	The date and time (year, month, day, hour, and minute)

Details

The *timespec* is a table using the fields listed in the table below.

year	The year (1970 or later)
month	The month (1 to 12)
day	The day (1 to 31)
hour	The hour (00 to 23)
min	The minute (00 to 59)
sec	The second (00 to 59)

If the time (hour, minute, and second) options are not used, they default to noon for that day. When called without a parameter (the first form), the function returns the current time.

Set the time zone before calling the `os.time()` function.

Example

```
systemTime = os.time({year = 2019,
       month = 3,
       day = 31,
       hour = 14,
       min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2019 at 2:25 pm.

Also see

settime() (on page 7-218)
settimezone() (on page 7-219)

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 7-111)

printbuffer()

This function prints data from tables or reading buffer subtables.

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

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

startIndex	Beginning index of the buffer to print; this must be more than one and less than *endIndex*
endIndex	Ending index of the buffer to print; this must be more than *startIndex* and less than the index of the last entry in the tables
bufferVar	First table or reading buffer subtable to print
bufferVar2	Second table or reading buffer subtable to print
bufferVarN	The last table or reading buffer subtable to print
...	One or more tables or reading buffer subtables separated with commas

Details

If *startIndex* ≤ 1, 1 is used as *startIndex*. If *n* < *endIndex*, *n* is used as *endIndex*.

When any given reading buffers are used in overlapped commands that have not yet completed (at least to the specified index), this function outputs data as it becomes available.

When there are outstanding overlapped commands to acquire data, *n* refers to the index that the last entry in the table has after all the measurements have completed.

If you pass a reading buffer instead of a reading buffer subtable, the default subtable for that reading buffer is used.

This command generates a single response message that contains all data. The response message is stored in the output queue.

The `format.data` attribute controls the format of the response message.

Example

```
format.data = format.ASCII
format.asciiprecision = 6
printbuffer(1, rb1.n, rb1)
```

This assumes that `rb1` is a valid reading buffer in the runtime environment. The use of `rb1.n` (*bufferVar*`.n`) indicates that the instrument should output all readings in the reading buffer. In this example, `rb1.n` equals 10.

Example of output data (`rb1.readings`):

```
4.07205e-05, 4.10966e-05, 4.06867e-05, 4.08865e-05, 4.08220e-05, 4.08988e-05,
    4.08250e-05, 4.09741e-05, 4.07174e-05, 4.07881e-05
```

Also see

bufferVar.n (on page 7-29)
bufferVar.readings (on page 7-30)
format.asciiprecision (on page 7-111)
format.byteorder (on page 7-112)
format.data (on page 7-113)
printnumber() (on page 7-192)

printnumber()

This function prints numbers using the configured format.

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

Usage

```
printnumber(value1)
printnumber(value1, value2)
printnumber(value1, ..., valueN)
```

value1	First value to print in the configured format
value2	Second value to print in the configured format
valueN	Last value to print in the configured format
...	One or more values separated with commas

Details

There are multiple ways to use this function, depending on how many numbers are to be printed.

This function prints the given numbers using the data format specified by `format.data` and `format.asciiprecision`.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x, 2.54321, 3.1)
```

Configure the ASCII precision to 10 and set `x` to 2.54.
Read the value of `x` based on these settings.
Change the ASCII precision to 3.
View how the change affects the output of `x` and some numbers.
Output:

```
2.540000000e+00
2.54e+00, 2.54e+00, 3.10e+00
```

Also see

format.asciiprecision (on page 7-111)
format.byteorder (on page 7-112)
format.data (on page 7-113)
print() (on page 7-190)
printbuffer() (on page 7-191)

PulseIMeasureV()

This KIPulse factory script function performs a specified number of pulse I, measure V cycles.

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

Usage

```
PulseIMeasureV(smu, bias, level, ton, toff, points)
```

smu	Instrument channel (set to `smua`)
bias	Bias level in amperes
level	Pulse level in amperes
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles

Details

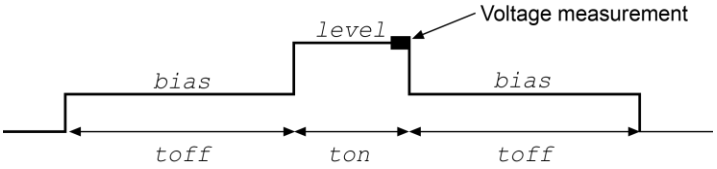
Data for pulsed voltage measurements, current levels, and timestamps are stored in `smua.nvbuffer1`.

If any parameters are omitted or `nil`, the operator is prompted to enter them using the front panel.

To perform the specified number of pulse I, measure V cycles, this function:

1. Sets the *smu* to output *bias* amperes and dwell for *toff* seconds.
2. Sets the *smu* to output *level* amperes and dwell for *ton* seconds.
3. Performs voltage measurement with source at *level* amperes.
4. Sets the *smu* to output *bias* amperes for *toff* seconds.
5. Repeats steps 2 through 4 for all remaining *points* pulse-measure cycles.

Figure 118: PulseIMeasureV



Example

```
PulseIMeasureV(smua, 0.001, 1.0,
   20e-3, 40e-3, 10)
```

SMU A outputs 1 mA and dwells for 40 ms, outputs 1 A and dwells for 20 ms. The voltage measurements occur during each 20 ms dwell period. After the measurement, the output returns to 1 mA and dwells for 40 ms. This pulse-measure process repeats nine more times.

Also see

KIPulse factory script (on page 5-23)

PulseVMeasureI()

This KIPulse factory script function performs a specified number of pulse V, measure I cycles.

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

Usage

```
PulseVMeasureI(smu, bias, level, ton, toff, points)
```

smu	Instrument channel (set to `smua`)
bias	Bias level in volts
level	Pulse level in volts
ton	Pulse on time in seconds
toff	Pulse off time in seconds
points	Number of pulse-measure cycles

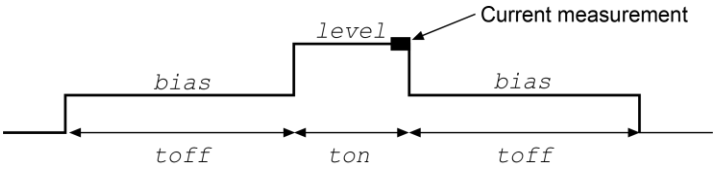
Details

If any parameters are omitted or `nil`, the operator is prompted to enter them using the front panel. Data for pulsed current measurements, voltage levels, and timestamps are stored in `smuX.nvbuffer1`.

To perform the specified number of pulse V, measure I cycles, this function:

1. Sets the *smu* to output *bias* volts and dwell for *toff* seconds
2. Sets the *smu* to output *level* volts and dwell for *ton* seconds
3. Performs voltage measurement with source at *level* volts
4. Sets the *smu* to output *bias* volts for *toff* seconds
5. Repeats steps 2 through 4 for the remaining *points* pulse-measure cycles

Figure 119: PulseVMeasureI()



Example

`smua.measure.nplc = 0.001` `PulseVMeasureI(smua, -1, 1, 1E-3, 2E-3, 20)`	SMU A outputs −1 V and dwells for 2 ms, outputs 1 V and dwells for 1 ms. The current measurement occurs during the dwell period. After the measurement, the output returns to −1 V and dwells for 2 ms. This pulse-measure process repeats 19 more times.

Also see

KIPulse factory script (on page 5-23)

QueryPulseConfig()

This KIPulse factory script function allows you to inspect the settings of the preconfigured pulse train assigned to *tag*.

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

Usage

tbl = QueryPulseConfig(*tag*)

tag	Numeric identifier to be assigned to the defined pulse train
tbl	Returned table

Details

Once a pulse train is configured and assigned to a *tag*, you can use the QueryPulseConfig() command to inspect the settings of this preconfigured pulse train.

This function returns a table that contains the settings associated with the *tag* input parameter.

Return values:	
tostring()	A function that returns most settings in a string that is convenient for printing
tag	Identifying tag for this pulse train
smu	The SMU configured for pulsing
func	Pulse function: smua.OUTPUT_DCAMPS or smua.OUTPUT_DCVOLTS
bias	Pulse bias level
level	Pulse level for non-sweeping pulses
start	Starting level for sweep pulses
stop	Ending level for sweep pulses
limit	Limit value
ton	On time in seconds
toff	Off time in seconds
points	The number of points in this pulse train
buf	Reference to the buffer that contains measurement data
sync_in	The sync_in digio line, if used
sync_out	The sync_out digio line, if used
sourcevalues	A table containing the source value for each point in the pulse train

Example

```
smua.reset()

smua.source.rangev = 5
smua.source.rangei = 1
smua.source.levelv = 0

smua.measure.rangev = 5
smua.measure.rangei = 1
smua.measure.nplc = 0.01
smua.measure.autozero = smua.AUTOZERO_ONCE

smua.nvbuffer1.clear()
smua.nvbuffer1.appendmode = 1

smua.source.output = smua.OUTPUT_ON

f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5,
   1, 0.002, 0.2, 10, smua.nvbuffer1, 1)

print(QueryPulseConfig(1).tostring())
```

Configure channel A to generate a pulse train, query configuration, and then display as a string. Channel A pulses voltage from a bias level of 0 V to a pulse level of 5 V. The pulse level is present for 2 ms, and the bias level for 200 ms with a 1 A limit setting. A total of 10 pulses are generated, and the measurement data is stored in smua.nvbuffer1. This pulse train is assigned to tag = 1.

Output:

```
>> tag = 1
>> smu = smua
>> func = volts
>> type = pulse
>> bias = 0
>> level = 5
>> limit = 1
>> time on = 0.002
>> time off = 0.2
>> points = 10
>> measure = yes
>> sync_in = 0
>> sync_out = 0
>> sync_in_timeout = 0
>> sync_out_abort = 0
>> { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 }
```

Also see

ConfigPulseIMeasureV() (on page 7-38)
ConfigPulseIMeasureVSweepLin() (on page 7-40)
ConfigPulseIMeasureVSweepLog() (on page 7-42)
ConfigPulseVMeasureI() (on page 7-44)
ConfigPulseVMeasureISweepLin() (on page 7-47)
ConfigPulseVMeasureISweepLog() (on page 7-49)
KIPulse factory script (on page 5-23)

reset()

This function resets commands to their default settings.

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

Usage

```
reset()
reset(system)
```

system	What to reset: ■ `true`: If the node is the master, the entire system is reset (default) ■ `false`: Only the local group is reset

Details

The `reset()` command in its simplest form resets the entire TSP-enabled system, including the controlling node and all subordinate nodes.

If you want to reset a specific instrument, use either the `localnode.reset()` or `node[X].reset()` command. Use the `localnode.reset()` command for the local instrument. Use the `node[X].reset()` command to reset an instrument on a subordinate node.

You can only reset the entire system using `reset(true)` if the node is the master. If the node is not the master node, executing this command generates an error.

Example

```
reset(true)
```

If the node is the master node, the entire system is reset; if the node is not the master node, an error is generated.

Also see

localnode.reset() (on page 7-179)

savebuffer()

This KISavebuffer factory script function saves a specified reading buffer as either a CSV file or an XML file.

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

Usage

```
savebuffer(buffer, "formatType", "fileName")
```

buffer	The reading buffer to save
formatType	A string indicating which file type to use: `csv` or `xml`
fileName	The file name of the saved buffer

Details

Use this function to save the specified buffer to a USB flash drive.

This function only saves to a USB flash drive.

You are not required to qualify the path to the USB flash drive, but you can add `/usb1/` before the *fileName* (see Example 2).

Example 1

```
savebuffer(smua.nvbuffer1, "csv", "mybuffer.csv")
```

Save `smua` dedicated reading buffer 1 as a CSV file named `mybuffer.csv`.

Example 2

```
savebuffer(smua.nvbuffer1, "csv", "/usb1/mybuffer.csv")
```

Save `smua` dedicated reading buffer 1 to an installed USB flash drive as a CSV file named `mybuffer.csv`.

Also see

KISavebuffer factory script (on page 5-25)
smuX.savebuffer() (on page 7-265)

script.anonymous

This is a reference to the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	See **Details**	See **Details**	Not applicable

Usage

```
scriptVar = script.anonymous
```

scriptVar	The name of the variable that references the script

Details

You can use the `script.anonymous` script like any other script. Also, you can save the anonymous script as a user script by giving it a name.

This script is replaced by loading a script with the `loadscript` or `loadandrunscript` commands when they are used without a name.

Example 1

```
script.anonymous.list()
```

Displays the content of the anonymous script.

Example 2

```
print(script.anonymous.source)
```

Retrieves the source of the anonymous script.

Also see

Anonymous scripts (on page 6-4)
scriptVar.autorun (on page 7-206)
scriptVar.list() (on page 7-207)
scriptVar.name (on page 7-208)
scriptVar.run() (on page 7-209)
scriptVar.save() (on page 7-210)
scriptVar.source (on page 7-211)

script.delete()

This function deletes a script from nonvolatile memory.

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

Usage

```
script.delete("scriptName")
```

scriptName	The string that represents the name of the script

Example

```
script.delete("test8")
```

Deletes a user script named `test8` from nonvolatile memory.

Also see

Delete user scripts from the instrument (on page 6-52)
scriptVar.save() (on page 7-210)

script.factory.catalog()

This function returns an iterator that can be used in a `for` loop to iterate over all the factory scripts.

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

Usage

```
for name in script.factory.catalog() do body end
```

name	String representing the name of the script
body	Code that implements the body of the `for` loop to process the names in the catalog

Details

Accessing this catalog of scripts allows you to process the factory scripts. The entries are enumerated in no particular order.

Each time the body of the function executes, *name* takes on the name of one of the factory scripts. The `for` loop repeats until all scripts have been iterated.

Example

```
for name in script.factory.catalog() do
   print(name)
end
```

Retrieve the catalog listing for factory scripts.

Also see

None

script.load()

This function creates a script from a specified file.

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

Usage

```
scriptVar = script.load("file")
scriptVar = script.load("file", "name")
```

scriptVar	The created script; this is `nil` if an error is encountered
file	The path and file name of the script file to load
name	The name of the script to be created

Details

The file path may be absolute or relative to the current working directory. The root folder of the USB flash drive has the absolute path `"/usb1/"`. Both the forward slash (/) and backslash (\) are supported as directory separators.

The file to be loaded must start with the `loadscript` or `loadandrunscript` keywords, contain the body of the script, and end with the `endscript` keyword.

Script naming:

- If the *name* parameter is an empty string, or *name* is absent (or `nil`) and the script name cannot be extracted from the file, *scriptVar* is the only handle to the created script.
- If *name* is given (and not `nil`), any script name embedded in the file is ignored.
- If *name* conflicts with the name of an existing script in the `script.user.scripts` table, the existing script's name attribute is set to an empty string before it is replaced in the `script.user.scripts` table by the new script.
- If *name* is absent or `nil`, the command attempts to extract the name of the script from the file. Any conflict between the extracted name and that of an existing script in the scripts table generates an error. If the script name cannot be extracted, the created script's name attribute is initialized to the empty string and must be set to a valid nonempty string before saving the script to nonvolatile memory.

Example

```
myTest8 =
      script.load("/usb1/filename.tsp",
   "myTest8")
```

Loads the script `myTest8` from the USB flash drive.

Also see

script.new() (on page 7-202)

script.new()

This function creates a script.

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

Usage

```
scriptVar = script.new("code")
scriptVar = script.new("code", "name")
```

scriptVar	The name of the variable that references the script
code	A string containing the body of the script
name	The name of the script

Details

The *name* parameter is the name that is added to the `script.user.scripts` table. If *name* is not provided, an empty string is used, and the script is unnamed. If the name already exists in `script.user.scripts`, the `name` attribute of the existing script is set to an empty string before it is replaced by the new script.

Note that *name* is the value that is used for the instrument front panel display. If this value is not defined, the script is not available from the front panel.

You must save the new script into nonvolatile memory to keep it when the instrument is turned off.

Example 1

```
myTest8 = script.new(
   "display.clear() display.settext('Hello from myTest8')", "myTest8")
myTest8()
```

Creates a new script referenced by the variable `myTest8` with the name `myTest8`.
Runs the script. The instrument displays `Hello from myTest8`.

Example 2

```
autoexec = script.new(
   "display.clear() display.settext('Hello from autoexec')", 'autoexec')
```

Creates a new autoexec script that clears the display when the instrument is turned on and displays `Hello from autoexec`.

Also see

Create a script using the script.new() command (on page 6-45)
Global variables and the script.user.scripts table (on page 6-44)
Named scripts (on page 6-4)
scriptVar.save() (on page 7-210)
script.newautorun() (on page 7-203)

script.newautorun()

This function creates a script and enables autorun.

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

Usage

```
scriptVar = script.newautorun("code")
scriptVar = script.newautorun("code", "name")
```

scriptVar	The name of the variable that references the script
code	A string that contains the body of the script
name	The name of the script

Details

The *name* parameter is the name that is added to the `script.user.scripts` table. If *name* is not provided, an empty string is used, and the script is unnamed. If the name already exists in `script.user.scripts`, the `name` attribute of the existing script is set to an empty string before it is replaced by the new script.

Note that *name* is the value that is used for the instrument front panel display. If this value is not defined, the script is not available from the front panel.

You must save the new script into nonvolatile memory to keep it when the instrument is turned off.

The script is run automatically immediately after it is created.

This command is the same as the `script.new()` function except that the script is automatically run.

Example

```
NewAuto = script.newautorun("print('Hello from new auto run command')", 'NewAuto')
print(NewAuto.autorun)
print(NewAuto.name)
```

Creates a new script called `NewAuto` that automatically has the autorun attribute set to yes after it is created. The *name* is set to `NewAuto`.

Output:

```
Hello from new auto run command
yes
NewAuto
```

Also see

Create a script using the script.new() command (on page 6-45)
Global variables and the script.user.scripts table (on page 6-44)
Named scripts (on page 6-4)
script.new() (on page 7-202)
scriptVar.autorun (on page 7-206)
scriptVar.save() (on page 7-210)

script.restore()

This function restores a script that was removed from the runtime environment.

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

Usage

```
script.restore(name)
```

name	The name of the script to be restored

Details

This command copies the script from nonvolatile memory into the runtime environment. It also creates a global variable with the same name as the name of the script.

Example

```
script.restore("test9")
```

Restores a script named `test9` from nonvolatile memory.

Also see

script.delete() (on page 7-200)

script.run()

This function runs the anonymous script.

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

Usage

```
script.run()
run()
```

Details

Each time the `script.run()` command is given, the anonymous script is executed. This script can be run using this command many times without having to re-send it.

Example

```
run()
```

Runs the anonymous script.

Also see

script.anonymous (on page 7-199)

script.user.catalog()

This function returns an iterator that can be used in a `for` loop to iterate over all the scripts stored in nonvolatile memory.

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

Usage

```
for name in script.user.catalog() do body end
```

name	String representing the name of the script
body	Code that implements the body of the `for` loop to process the names in the catalog

Details

This function accesses the catalog of scripts stored in nonvolatile memory, which allows you to process all scripts in nonvolatile memory. The entries are enumerated in no particular order.

Each time the body of the function executes, *name* takes on the name of one of the scripts stored in nonvolatile memory. The `for` loop repeats until all scripts have been iterated.

Example

```
for name in script.user.catalog() do
   print(name)
end
```

Retrieve the catalog listing for user scripts.

Also see

None

scriptVar.autorun

This attribute controls the autorun state of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	See **Details**	See **Details**

Usage

```
scriptVar.autorun = "state"
state = scriptVar.autorun
```

scriptVar	The name of the variable that references the script
state	String that indicates whether or not the script runs automatically when powered on: ■ `yes`: Script runs automatically ■ `no`: Script does not run automatically

Details

Autorun scripts run automatically when the instrument is turned on. You can set any number of scripts to autorun.

The run order for autorun scripts is arbitrary, so make sure the run order is not important.

The default value for *scriptVar*`.autorun` depends on how the script was loaded. The default is `no` if the script was loaded with `loadscript` or `script.new()`. It is `yes` for scripts loaded with `loadandrunscript` or `script.newautorun()`.

NOTE

Make sure to save the script in nonvolatile memory after setting the `autorun` attribute so that the instrument keeps the setting.

Example

```
test5.autorun = "yes"
test5.save()
```

Assume a script named `test5` is in the runtime environment.
The next time the instrument is turned on, `test5` script automatically loads and runs.

Also see

None

scriptVar.list()

This function generates a script listing.

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

Usage

```
scriptVar.list()
```

scriptVar	The name of the variable that references the script

Details

This function generates output in the form of a sequence of response messages (one message for each line of the script). It also generates output of the script control messages (`loadscript` or `loadandrunscript` and `endscript`).

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')",
   "test7")
test7()
test7.save()
test7.list()
```

The above example code creates a script named `test7` that displays text on the front panel and lists the script with the following output:

```
loadscript test7
display.clear() display.settext("Hello from my test")
endscript
```

Also see

Retrieve source code one line at a time (on page 6-50)

scriptVar.name

This attribute contains the name of a script in the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not applicable	Not applicable

Usage

```
scriptVar.name = "scriptName"
scriptName = scriptVar.name
```

scriptVar	Name of the variable that references the script
scriptName	A string that represents the name of the script

Details

When setting the script name, this attribute renames the script that the variable *scriptVar* references.

This attribute must be either a valid Lua identifier or the empty string. Changing the name of a script changes the index that is used to access the script in the `script.user.scripts` table. Setting the attribute to an empty string removes the script from the table completely, and the script becomes an unnamed script.

As long as there are variables referencing an unnamed script, the script can be accessed through those variables. When all variables that reference an unnamed script are removed, the script is removed from the runtime environment.

If the new name is the same as a name that is already used for another script, the name of the other script is set to an empty string, and that script becomes unnamed.

NOTE

Changing the name of a script does not change the name of any variables that reference that script. The variables still reference the script, but the names of the script and variables may not match.

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
test7()
print(test7.name)

test7.name = "test7"
print(test7.name)

test7.save()
```

This example calls the `script.new()` function to create a script with no name, runs the script, names the script `test7`, and then saves the script in nonvolatile memory.

Also see

Rename a script (on page 6-48)
script.new() (on page 7-202)
scriptVar.save() (on page 7-210)

scriptVar.run()

This function runs a script.

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

Usage

```
scriptVar.run()
scriptVar()
```

scriptVar	The name of the variable that references the script

Details

The *scriptVar*.run() function runs the script referenced by *scriptVar*. You can also run the script by using *scriptVar()*.

To run a factory script, use script.factory.scripts.*scriptName*(), replacing *scriptName* with the name of the factory script.

Example

```
test8.run()
```

Runs the script referenced by the variable test8.

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory or to a USB flash drive.

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

Usage

```
scriptVar.save()
scriptVar.save("filename")
```

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 file name parameter is an empty string), the script is saved to internal nonvolatile memory. Only a script with *filename* defined can be saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

You can add the file extension, but it is not required. The only allowed extension is .tsp (see Example 2).

Example 1

```
test8.save()
```

Saves the script referenced by the variable test8 to nonvolatile memory.

Example 2

```
test8.save("/usb1/myScript.tsp")
```

Saves the script referenced by the variable test8 to a file named myScript.tsp on your USB flash drive.

Also see

Save a user script (on page 6-11)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW) (see **Details**)	No	Not applicable	Not saved	Not applicable

Usage

```
code = scriptVar.source
scriptVar.source = nil
```

scriptVar	The name of the variable that references the script that contains the source code
code	A string that contains the body of the script

Details

The `loadscript` or `loadandrunscript` and `endscript` keywords are not included in the source code.

The body of the script is a single string with lines separated by the new line character.

The instrument automatically stores the source for all scripts that are loaded on the instrument. To free up memory or to obfuscate the code, assign `nil` to the source attribute of the script. Although this attribute is writable, it can only be set to the `nil` value.

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
print(test7.source)
```

This example creates a script called `test7` that displays a message on the front panel and retrieves the source code.
Output:

```
display.clear() display.settext('Hello from my test')
```

Also see

scriptVar.list() (on page 7-207)

serial.baud

This attribute configures the baud rate for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	9600

Usage

```
baud = serial.baud
serial.baud = baud
```

baud	The baud rate (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200)

Details

A new baud rate setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set the baud rate from one of the other command interfaces or from the front panel.

The reset function has no effect on data bits.

Example

`serial.baud = 1200`	Sets the baud rate to 1200.

Also see

RS-232 interface operation (on page 2-90)
serial.databits (on page 7-213)
serial.flowcontrol (on page 7-214)
serial.parity (on page 7-215)

serial.databits

This attribute configures character width (data bits) for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	8

Usage

```
bits = serial.databits
serial.databits = bits
```

bits	An integer representing the character width (7 or 8)

Details

A new data width setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set the character width from one of the other command interfaces or from the front panel.

The reset function has no effect on data bits.

Example

`serial.databits = 8`	Sets data width to 8.

Also see

RS-232 interface operation (on page 2-90)
serial.baud (on page 7-212)
serial.flowcontrol (on page 7-214)
serial.parity (on page 7-215)

serial.flowcontrol

This attribute configures flow control for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	"none" (serial.FLOW_NONE)

Usage

```
flow = serial.flowcontrol
serial.flowcontrol = flow
```

flow	A string or value that represents flow control configuration; set to: ■ `"none"` or `serial.FLOW_NONE` (selects no flow control) ■ `"hardware"` or `serial.FLOW_HARDWARE` (selects hardware flow control)

Details

A new flow control setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set the flow control from one of the other command interfaces or from the front panel.

The reset function has no effect on flow control.

Example

`serial.flowcontrol = serial.FLOW_NONE`	Sets flow control to `none`.

Also see

serial.baud (on page 7-212)
serial.databits (on page 7-213)
serial.parity (on page 7-215)

serial.parity

This attribute configures parity for the RS-232 port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	"none" (serial.PARITY_NONE)

Usage

```
parity = serial.parity
serial.parity = parity
```

parity	Set parity to one of the following values: ■ Select no parity (`"none"` or `serial.PARITY_NONE`) ■ Select even parity (`"even"` or `serial.PARITY_EVEN`) ■ Select odd parity (`"odd"` or `serial.PARITY_ODD`)

Details

A new parity setting takes effect when the command to change it is processed.

NOTE

Allow ample time for the command to be processed before attempting to communicate with the instrument again. If possible, set parity from one of the other command interfaces or from the front panel.

The reset function has no effect on parity.

Example

`serial.parity = serial.PARITY_NONE`	Sets parity to `none`.

Also see

RS-232 interface operation (on page 2-90)
serial.baud (on page 7-212)
serial.databits (on page 7-213)
serial.flowcontrol (on page 7-214)

serial.read()

This function reads available characters (data) from the serial port.

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

Usage

```
data = serial.read(maxchars)
```

data	A string that consists of all data read from the serial port
maxchars	An integer that specifies the maximum number of characters to read

Details

This function reads available characters from the serial port. It does not wait for new characters to arrive. As long as *maxchars* is less than 200 characters, all characters that are received by the serial port (before the `serial.read()` command is executed) are returned. If too many characters are received between calls to this function, the RS-232 buffers overflow and some characters may be lost.

Call this function as many times as necessary to receive the required number of characters. For optimal performance, use a small delay between repeated calls to this function.

The data returned is the raw data stream read from the port. No characters, such as control characters or terminator characters, are interpreted.

If you attempt to use this function when the serial port is enabled as a command interface, a settings conflict error is generated.

Example

```
data = serial.read(200)

print(data)
```

Read data from the serial port.

Output:

```
John Doe
```

The above output indicates that the string "John Doe" was read from the serial port.

Also see

serial.write() (on page 7-217)

serial.write()

This function writes data to the serial port.

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

Usage

```
serial.write("data")
```

data	A string representing the data to write

Details

This function writes the specified string to the serial port, where it can be read by connected equipment (for example, a component handler).

No terminator characters are added to the data, and data is written exactly as specified by the *data* parameter.

Example

```
serial.write("1 2 3 4")
```

Write data string `"1 2 3 4"` to the serial port.

Also see

serial.read() (on page 7-216)

settime()

This function sets the real-time clock (sets present time of the system).

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

Usage

```
settime(time)
```

time	The time in seconds since January 1, 1970 UTC

Details

This function sets the date and time of the instrument based on the *time* parameter (specified in UTC time). UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

```
systemTime = os.time({year = 2020,
       month = 3,
       day = 31,
       hour = 14,
       min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2020 at 2:25 pm.

Also see

gettimezone() (on page 7-121)
os.time() (on page 7-189)
settimezone() (on page 7-219)

settimezone()

This function sets the local time zone.

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

Usage

```
settimezone(offset)
settimezone("offset", "dstOffset", "dstStart", "dstEnd")
```

offset	String representing offset from UTC
dstOffset	String representing the daylight savings offset from UTC
dstStart	String representing when daylight savings time starts
dstEnd	String representing when daylight savings time ends

Details

You only need to set the time zone if you use the `os.time()` and `os.date()` functions.

If only one parameter is given, the same time offset is used throughout the year. If four parameters are given, time is adjusted twice during the year for daylight savings time.

offset and *dstOffset* are strings of the form `"[+|-]hh[:mm[:ss]]"` that indicate how much time must be added to the local time to get UTC time:

- `hh` is a number between 0 and 23 that represents hours
- `mm` is a number between 0 and 59 that represents minutes
- `ss` is a number between 0 and 59 that represents seconds

The minute, second, +, and − fields are optional.

For example, to set the UTC-5 time zone, you specify the string `"5"`, because UTC-5 is 5 hours behind UTC and you must add 5 hours to the local time to determine UTC time. To specify the time zone UTC4, you specify `"-4"`, because UTC4 is 4 hours ahead of UTC and 4 hours must be subtracted from the local time to determine UTC.

dstStart and *dstEnd* are strings of the form `"MM.w.dw/hh[:mm[:ss]]"` that indicate when daylight savings time begins and ends respectively:

- `MM` is a number between 1 and 12 that represents the month
- `w` is a number between 1 and 5 that represents the week in the month
- `dw` is a number between 0 and 6 that represents the day of the week (where 0 is Sunday)

The rest of the fields represent the time of day that the change takes effect:

- `hh` represents hours
- `mm` represents minutes
- `ss` represents seconds

The minutes and seconds fields are optional.

The week of the month and day of the week fields are not specific dates.

Example

```
settimezone("8", "1", "3.3.0/02", "11.2.0/02")
   settimezone(offset)
```

Sets `offset` to equal +8 hours, +1 hour for DST, starts on Mar 14 at 2:00 am, ends on Nov 7 at 2:00 am.
Sets local time zone to `offset`.

Also see

gettimezone() (on page 7-121)
os.time() (on page 7-189)
settime() (on page 7-218)

setup.poweron

This attribute specifies which saved setup to recall when the instrument is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0

Usage

```
id = setup.poweron
setup.poweron = id
```

id	An integer that specifies the setup to recall when the instrument power is turned on (0 to 5)

Details

When *id* = 0, the instrument uses the factory default setup when it is turned on. When *id* is set to 1 to 5, it uses the setup saved with `setup.save()`.

Only setups stored in nonvolatile memory are available (you cannot recall a script from a USB flash drive with this command).

To save a script that is used when the instrument is powered on, you can create a configuration script and name it `autoexec`.

Example

`setup.poweron = 0`	Set the instrument to use the factory default setup when power is turned on.

Also see

setup.save() (on page 7-222)
Start-up (power-on) configuration

setup.recall()

This function recalls settings from a saved setup.

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

Usage

```
setup.recall(id)
```

id	An integer or string that specifies the location of the setup to recall: ■ Factory default setup: `0` ■ User-saved setup in nonvolatile memory: `1` to `5` User-saved setup on a USB flash drive: "*/path/filename*"

Details

When the *id* parameter is an integer (*n*), it is interpreted as the setup number to restore from the instrument's nonvolatile memory. When *n* = 0, the instrument recalls the factory default setup; when *n* = 1 to 5, the instrument recalls a user-saved setup.

When the *id* parameter is a string, it is interpreted as the path and file name of the setup to restore from a file on a USB flash drive. The path may be absolute or relative to the current working directory.

Before a setup is recalled, an instrument reset is performed.

Example 1

`setup.recall(1)`	Recall the user-saved setup at location 1.

Example 2

`setup.recall("/usb1/KEITHLEY_30730.set")`	Recall a user-saved setup stored in a file named `KEITHLEY_30730` on a USB flash drive.

Also see

Saved setups (on page 2-42)
setup.save() (on page 7-222)

setup.save()

This function saves the present setup as a user-saved setup.

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

Usage

```
setup.save(id)
```

id	An integer or string specifying where to save the user setup: ■ Save in nonvolatile memory (1 to 5) ■ Save as user-saved setup on a USB flash drive ("*/path/filename*")

Details

When the *id* parameter is an integer (*n*), it is interpreted as the setup number to save to the nonvolatile memory of the instrument.

NOTE

When you save to a specified integer (1 to 5) in nonvolatile memory, the previous setup at that same location is overwritten.

When the *id* parameter is a string, it is interpreted as the path and file name of the location to save the present setup on a USB flash drive. The path may be absolute or relative to the current working directory.

Example

`setup.save(5)`	Saves the present setup to the internal memory of the instrument at location 5.

Also see

Saved setups (on page 2-42)
setup.recall() (on page 7-221)

smuX.abort()

This function terminates all overlapped operations on the source-measure unit (SMU).

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

Usage

```
smuX.abort()
```

X	Source-measure unit (SMU) channel (for example, `smua.abort()` applies to SMU channel A)

Details

The `smuX.abort()` function does not turn the output off or change any settings.

If this function is used to abort a sweep, when it is executed, the SMU exits its trigger model immediately and returns to the idle state of the trigger model.

Example

```
smua.abort()
```
Terminates all overlapped operations.

Also see

smuX.measure.overlappedY() (on page 7-257)
smuX.trigger.initiate() (on page 7-294)

smua.buffer.getstats()

This function returns the statistics for a specified reading buffer.

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

Usage

```
statistics = smua.buffer.getstats(bufferVar)
```

statistics	The statistical data about the data in the reading buffer
bufferVar	The reading buffer to process

Details

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

The SMU automatically updates reading buffer statistics as data is added to the reading buffer. When the reading buffer is configured to wrap around and overwrite older data with new data, the buffer statistics include the data that was overwritten.

The table returned from this function is a snapshot. Although the SMU continues to update the statistics, the table returned is not updated. To get fresh statistics, call this function again.

The *statistics* parameter has the attributes described in the following table.

Attribute	When returned	Description
n	Always	The number of data points on which the statistics are based
mean	When n > 0	The average of all readings added to the buffer
stddev	When n > 1	The standard deviation of all readings (samples) added to the buffer
min	When n > 0	A table containing data about the minimum reading value added to the buffer
max	When n > 0	A table containing data about the maximum reading value added to the buffer

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

The min and max entries each have the attributes defined in the following table.

Attribute	Description
measurefunction	String indicating the function that was measured for the reading (current, voltage, ohms, or watts)
measurerange	The full-scale range value for the measurement range used when the measurement was made
reading	The reading value
sourcefunction	String indicating the source function at the time of the measurement (current or voltage)
sourceoutputstate	String indicating the state of the source (off or on)
sourcerange	Full-scale range value for the source range used when the measurement was made
sourcevalue	If *bufferVar*.collectsourcevalues is enabled, the sourced value in effect at the time of the reading
status	Status value for the reading; the status value is a floating-point number that encodes the status value into a floating-point value
timestamp	If *bufferVar*.collecttimestamps is enabled, the timestamp, in seconds, between when the reading was acquired and when the first reading in the buffer was acquired; adding this value to the base timestamp produces the actual time the measurement was acquired

Example

```
reset()
smua.nvbuffer1.clear()
smua.measure.count = 10
smua.measure.v(smua.nvbuffer1)
stats = smua.buffer.getstats(smua.nvbuffer1)
print("n= "..stats.n)
print("mean= "..stats.mean)
print("stddev= "..stats.stddev)
print("min= "..stats.min.reading)
print("max= "..stats.max.reading)
```

Make measurements and store them in nvbuffer1. Print the statistics for the data.
Example output:

```
n= 10
mean= -2.3851394871599e-05
stddev= 4.406545187484e-07
min= -2.4557113647461e-05
max= -2.322196996829e-05
```

Also see

smuX.buffer.recalculatestats() (on page 7-225)

smuX.buffer.recalculatestats()

This function recalculates the statistics of the specified reading buffer.

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

Usage

```
smuX.buffer.recalculatestats(bufferVar)
```

bufferVar	The reading buffer to process

Details

This function causes the SMU to regenerate the reading buffer statistics about the specified reading buffer. Because the SMU automatically updates reading buffer statistics when data is added to the reading buffer, this function is generally not needed. When the reading buffer is configured to wrap around and overwrite older data with new data, the buffer statistics include the data that was overwritten. Use this function to recalculate the statistics that include only the data that is presently stored in the buffer.

Example

```
smua.buffer.recalculatestats(smua.nvbuffer1)
```

Recalculates the statistics of buffer `smua.nvbuffer1`.

Also see

bufferVar.fillmode (on page 7-26)
smuX.buffer.getstats() (on page 7-223)

smuX.cal.adjustdate

This attribute stores the date of the last calibration adjustment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU cal. restore	SMU nonvolatile memory	Initially set to factory calibration date

Usage

```
adjustDate = smuX.cal.adjustdate
smuX.cal.adjustdate = adjustDate
```

adjustDate	Date of the last adjustment

Details

This attribute stores the adjustment date associated with the active calibration set. The adjustment date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock()` function.

You cannot change the adjustment date without first making a change to the calibration constants.

Once you change any calibration constants, you must set the adjustment date before you can save the calibration data to the nonvolatile memory of the SMU.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute reflects the date stored with that set.

`smuX.cal.adjustdate` must be set to the date the adjustment was done using the UTC time and date. The date is stored as the number of seconds since UTC, 12:00 am Jan 1, 1970.

Due to the internal storage format, `smuX.cal.adjustdate` is only accurate to within a few minutes of the value set.

Example

`smua.cal.adjustdate = os.time()`	Sets the adjustment date to the current time set on the instrument.

Also see

Adjustment (on page 12-15)
os.time() (on page 7-189)
smuX.cal.date (on page 7-227)
smuX.cal.due (on page 7-228)
smuX.cal.lock() (on page 7-230)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.state (on page 7-235)
smuX.cal.unlock() (on page 7-236)

smuX.cal.date

This attribute stores the calibration date of the active calibration set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU calibration restore	SMU nonvolatile memory	Initially set to factory calibration date

Usage

```
calDate = smuX.cal.date
smuX.cal.date = calDate
```

calDate	The calibration date of the active calibration set

Details

This attribute stores the calibration date that is associated with the active calibration set. The calibration date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock()` function. It is typically set to the date when the instrument was calibrated.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute reflects the date stored with that set.

`smuX.cal.date` must be set to the date the calibration was done using the UTC time and date. The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.

Due to the internal storage format, `smuX.cal.date` is accurate to within a few minutes of the value set.

Example

`smua.cal.date = os.time()`	Sets calibration date to the present time set on the instrument.

Also see

Adjustment (on page 12-15)
os.time() (on page 7-189)
smuX.cal.adjustdate (on page 7-226)
smuX.cal.due (on page 7-228)
smuX.cal.lock() (on page 7-230)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.state (on page 7-235)
smuX.cal.unlock() (on page 7-236)

smuX.cal.due

This attribute stores the calibration due date for the next calibration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU cal. restore	SMU nonvolatile memory	0

Usage

```
calDue = smuX.cal.due
smuX.cal.due = calDue
```

calDue	Due date of next calibration (`0` indicates that no date is set)

Details

This attribute stores the calibration due date associated with the active calibration set. The calibration due date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock()` function. It is typically set to the date when the next calibration should be performed.

This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute reflects the due date stored with that set.

`smuX.cal.due` must be set to the date the next calibration is required using the UTC time and date. The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.

Due to the internal storage format, `smuX.cal.due` is only accurate to within a few minutes of the value set.

Example

`smua.cal.due = os.time() + 365 * 24 * 60 * 60`	Sets the calibration due date equal to one year from the present time set on the instrument.

Also see

Adjustment (on page 12-15)
os.time() (on page 7-189)
smuX.cal.adjustdate (on page 7-226)
smuX.cal.date (on page 7-227)
smuX.cal.lock() (on page 7-230)
smuX.cal.restore() (on page 7-233)
smuX.cal.state (on page 7-235)
smuX.cal.unlock() (on page 7-236)

smuX.cal.fastadc()

This function performs calibration of the fast analog-to-digital converter (fast ADC).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.fastadc()
```

X	Source-measure unit (SMU) channel (for example, `smua.cal.fastadc()` specifies SMU channel A)

Details

This function automatically performs all the steps required to calibrate the fast ADC. This function uses the readings obtained when calibrating the measure ranges. Make sure both voltage and current calibration has been completed before calling this function.

Example

```
smua.cal.fastadc()
```

Performs fast ADC calibration for SMU Channel A.

Also see

Adjustment (on page 12-15)
smuX.measure.calibrateY() (on page 7-246)

smuX.cal.lock()

This function disables the commands that change calibration settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.lock()
```

Details

Before you can lock calibration, the calibration constants must be written to nonvolatile memory or a previous calibration set must be restored. Error code 5012, "Cal data not saved - save or restore before lock," results if this function is called when the calibration state is `smuX.CALSTATE_CALIBRATING`.

Example

`smua.cal.lock()`	Disables calibration functions.

Also see

Adjustment (on page 12-15)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.state (on page 7-235)
smuX.cal.unlock() (on page 7-236)

smuX.cal.password

This attribute stores the password required to enable calibration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	Not applicable	SMU nonvolatile memory	"KI0026XX"

Usage

```
smuX.cal.password = "newPassword"
```

newPassword	A string that contains the new password

Details

A new password can only be assigned when calibration has been unlocked.

The calibration password is write-only and cannot be read.

Example

`smua.cal.password = "LetMeIn"`	Assigns a new calibration password.

Also see

Adjustment (on page 12-15)
smuX.cal.unlock() (on page 7-236)

smuX.cal.polarity

This attribute controls which calibration constants are used for all subsequent measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset SMU cal. lock Recall setup	Not saved	0 (smu*X*.CAL_AUTO)

Usage

```
calPolarity = smuX.cal.polarity
smuX.cal.polarity = calPolarity
```

calPolarity	The polarity to use for measurements. Set to one of the following values: ■ `0` or `smuX.CAL_AUTO`: Automatic polarity detection ■ `1` or `smuX.CAL_POSITIVE`: Measure with positive polarity calibration constants ■ `2` or `smuX.CAL_NEGATIVE`: Measure with negative polarity calibration constants
X	SMU channel (for example, `smua.cal.polarity` applies to SMU channel A)

Details

This attribute controls which polarity calibration constants are used to make all subsequent measurements.

This attribute does not affect the `smuX.measure.calibrateY()` command. The polarity for `smuX.measure.calibrateY()` is dictated by the range parameter that is defined for it. The measurement calibration commands require the measurements provided to have been made using the polarity being calibrated.

When making measurements for calibration points far away from zero, the desired polarity constants are inherently used. However, when making measurements near zero, it is possible that the instrument could use the calibration constants from the wrong polarity. Setting `smuX.cal.polarity` to positive or negative forces measurements to be made using the calibration constants for a given polarity, rather than basing the choice on the raw measurement data.

This attribute can only be set to positive or negative when calibration is unlocked. This attribute is automatically set to `smuX.CAL_AUTO` when calibration is locked.

Example

```
smua.cal.polarity = smua.CAL_POSITIVE
```

Selects positive calibration constants for all subsequent measurements.

Also see

Adjustment (on page 12-15)
reset() (on page 7-197)
smuX.cal.lock() (on page 7-230)
smuX.cal.unlock() (on page 7-236)
smuX.measure.calibrateY() (on page 7-246)
smuX.reset() (on page 7-264)
smuX.source.calibrateY() (on page 7-268)

smuX.cal.restore()

This function loads a stored set of calibration constants.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.restore()
smuX.cal.restore(calset)
```

calset	The calibration set to be loaded; set *calset* to one of the following values: ■ 0 or smuX.CALSET_NOMINAL: A set of calibration constants that are uncalibrated, but set to nominal values to allow rudimentary functioning of the instrument ■ 1 or smuX.CALSET_FACTORY: The calibration constants when the instrument left the factory ■ 2 or smuX.CALSET_DEFAULT: The normal calibration set ■ 3 or smuX.CALSET_PREVIOUS: The calibration set that was used before the last default set was overwritten

Details

This function overwrites the present set of calibration constants with constants read from nonvolatile memory.

This function is disabled until a successful call to smuX.cal.unlock() is made.

If *calset* is not specified, smuX.CALSET_DEFAULT is used.

Example

smua.cal.restore()	Restores factory calibration constants.

Also see

Adjustment (on page 12-15)
smuX.cal.lock() (on page 7-230)
smuX.cal.unlock() (on page 7-236)

smuX.cal.save()

This function stores the active calibration constants to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.save()
```

Details

This function stores the active set of calibration constants to nonvolatile memory. The previous calibration constants (from the default calibration set) are copied to the previous calibration set (`smuX.CALSET_PREVIOUS`) before overwriting the default calibration set.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

If any of the calibration constants have been changed, this function is disabled unless the calibration date, the calibration due date, and the calibration adjust date have been assigned new values.

Example

`smua.cal.save()`	Stores calibration constants in nonvolatile memory.

Also see

Adjustment (on page 12-15)
smuX.cal.adjustdate (on page 7-226)
smuX.cal.date (on page 7-227)
smuX.cal.due (on page 7-228)
smuX.cal.lock() (on page 7-230)
smuX.cal.restore() (on page 7-233)
smuX.cal.unlock() (on page 7-236)

smuX.cal.state

This attribute returns the present calibration state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
calState = smuX.cal.state
```

calState	The present calibration state; when reading this attribute, *calState* has one of the following values: ■ 0 or smuX.CALSTATE_LOCKED: Calibration is locked ■ 1 or smuX.CALSTATE_CALIBRATING: The calibration constants or dates have been changed but not yet saved to nonvolatile memory ■ 2 or smuX.CALSTATE_UNLOCKED: Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore

Details

This read-only attribute indicates the calibration state of the instrument: Locked, calibrating, or unlocked.

Example

`calstate = smua.cal.state` `print(calstate)`	Reads calibration state. Output: `0.000000e+00` The above output indicates that calibration is locked.

Also see

Adjustment (on page 12-15)
smuX.cal.lock() (on page 7-230)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.unlock() (on page 7-236)

smuX.cal.unlock()

This function enables the commands that change calibration settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.cal.unlock(password)
```

password	Calibration password

Details

This function enables the calibration functions to change the calibration settings.

The password when the instrument is shipped from the factory is "KI0026XX".

Example

```
smua.cal.unlock("KI0026XX")
```
Unlocks calibration.

Also see

Adjustment (on page 12-15)
smuX.cal.lock() (on page 7-230)
smuX.cal.password (on page 7-231)
smuX.cal.restore() (on page 7-233)
smuX.cal.state (on page 7-235)

smuX.contact.calibratehi()

This function adjusts the high/sense high contact check measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.contact.calibratehi(cp1Measured, cp1Reference, cp2Measured, cp2Reference)
```

cp1Measured	The value measured by this SMU for point 1
cp1Reference	The reference measurement for point 1 as measured externally
cp2Measured	The value measured by this SMU for point 2
cp2Reference	The reference measurement for point 2 as measured externally

Details

Contact check measurement calibration does not require range information.

Typically, points one and two are near 0 Ω and 50 Ω, respectively.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the calibration set that is active. If not, corruption of the calibration constants may result.

The new calibration constants are activated immediately but are not written to nonvolatile memory. Use `smuX.cal.save()` to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

Example

```
-- Short SENSE LO and LO terminals.
-- Short SENSE HI and HI terminals.
-- Allow readings to settle, then get measurements.
r0_hi, r0_lo = smua.contact.r()

-- Connect 50 OHM resistor between SENSE LO and LO.
-- Connect 50 OHM resistor between SENSE HI and HI.
-- Allow readings to settle, then get measurements.
r50_hi, r50_lo = smua.contact.r()
smua.contact.calibratelo(r0_lo, Z_actual_lo, r50_lo, 50_ohm_actual_lo)
smua.contact.calibratehi(r0_hi, Z_actual_hi, r50_hi, 50_ohm_actual_hi)
```

The instrument performs a contact check.
Install and measure two resistors.
The user sends the contact check LO calibration command.
The user sends the contact check HI calibration command.

Also see

Adjustment (on page 12-15)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.unlock() (on page 7-236)
smuX.contact.calibratelo() (on page 7-238)

smuX.contact.calibratelo()

This function adjusts the low/sense low contact check measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.contact.calibratelo(cp1Measured, cp1Reference, cp2Measured, cp2Reference)
```

cp1Measured	The value measured by this SMU for point 1
cp1Reference	The reference measurement for point 1 as measured externally
cp2Measured	The value measured by this SMU for point 2
cp2Reference	The reference measurement for point 2 as measured externally

Details

Contact check measurement adjustment does not require range information.

Typically, points one and two are near 0 Ω and 50 Ω, respectively.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the active calibration set. If not, corruption of the calibration constants may result.

The new calibration constants are activated immediately but are not written to nonvolatile memory. Use `smuX.cal.save()` to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is disabled until a successful call to `smuX.cal.unlock()` is made.

Example

```
-- Short SENSE LO and LO terminals.
-- Short SENSE HI and HI terminals.
-- Allow readings to settle, then get measurements.
r0_hi, r0_lo = smua.contact.r()

-- Connect 50 OHM resistor between SENSE LO and LO.
-- Connect 50 OHM resistor between SENSE HI and HI.
-- Allow readings to settle, then get measurements.
r50_hi, r50_lo = smua.contact.r()
smua.contact.calibratelo(r0_lo, Z_actual_lo, r50_lo, 50_ohm_actual_lo)
smua.contact.calibratehi(r0_hi, Z_actual_hi, r50_hi, 50_ohm_actual_hi)
```

The instrument performs a contact check.
Install and measure two resistors.
The user sends the contact check LO calibration command.
The user sends the contact check HI calibration command.

Also see

Adjustment (on page 12-15)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.unlock() (on page 7-236)
smuX.contact.calibratehi() (on page 7-236)

smuX.contact.check()

This function determines if contact resistance is lower than the threshold.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.contact.check()
```

Details

This function returns `true` if the contact resistance is below the threshold; this function returns `false` if it is above the threshold. The threshold value is set by the `smuX.contact.threshold` attribute.

An error is generated when the output is on and:

- SMU is a current source with current range set to less than 1 mA (error code 5065, "I range too low for contact check")
- SMU is a voltage source with current limit set to less than 1 mA (error code 5050, "I limit too low for contact check")

An error is generated when the output is off and:

- The output off mode is High-Z (error code 5048, "Contact check not valid with HIGH-Z OUTPUT off")
- The output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCVOLTS` and the off current limit set to less than 1 mA (error code 5066, "source.offlimiti too low for contact check")
- The output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCAMPS` and the source range is less than 1 mA (error code 5065, "I range too low for contact check")

Example

`if not smua.contact.check() then`
`  -- take action`
`end`
	Takes action if contact check fails.

Also see

Contact check connections (on page 2-49)
Contact check measurements (on page 2-40)
smuX.contact.speed (on page 7-241)
smuX.contact.threshold (on page 7-242)
smuX.source.offfunc (on page 7-275)

smuX.contact.r()

This function measures aggregate contact resistance.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
rhi, rlo = smuX.contact.r()
```

rhi	The measured aggregate contact resistance on the HI/sense HI side
rlo	The measured aggregate contact resistance on the LO/sense LO side

Details

If you attempt to perform a contact resistance measurement when any of the following conditions exist, an error is generated.

- When the output is on and SMU is a current source with current range set to less than 1 mA
- When the output is on and SMU is a voltage source with current limit set to less than 1 mA
- When the output is off and the output off mode is High-Z
- When the output is off and the output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCVOLTS` and the off current limit set to less than 1 mA
- When the output is off and the output off mode is Normal with the `smuX.source.offfunc` attribute set to `smuX.OUTPUT_DCAMPS` and the source range is less than 1 mA

Example

```
if not smua.contact.check() then
   smua.contact.speed = smua.CONTACT_SLOW
   rhi, rlo = smua.contact.r()
   print(rhi, rlo)
   exit()
end
```

Check contacts against threshold.
Set speed to slow.
Get resistance readings.
Output contact resistances to the host.
Terminate execution.

Also see

Contact check connections (on page 2-49)
Contact check measurements (on page 2-40)
smuX.contact.check() (on page 7-239)
smuX.contact.speed (on page 7-241)

smuX.contact.speed

This attribute stores the speed setting for contact check measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	0 (smu*X*.CONTACT_FAST)

Usage

```
speedSetting = smuX.contact.speed
smuX.contact.speed = speedSetting
```

speedSetting	The speed setting. Set to one of the following: ■ 0 or smu*X*.CONTACT_FAST ■ 1 or smu*X*.CONTACT_MEDIUM ■ 2 or smu*X*.CONTACT_SLOW

Details

This setting controls the aperture of measurements made for contact check. It does not affect the smu*X*.measure.nplc aperture setting.

The speed setting can have a dramatic effect on the accuracy of the measurement (see specifications).

Example

```
smua.contact.speed = smua.CONTACT_SLOW
```

Configure contact check for higher accuracy.

Also see

Contact check connections (on page 2-49)
Contact check measurements (on page 2-40)
reset() (on page 7-197)
smuX.contact.check() (on page 7-239)
smuX.contact.r() (on page 7-240)
smuX.reset() (on page 7-264)

smuX.contact.threshold

This attribute stores the resistance threshold for the `smuX.contact.check()` function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	50 (50 Ω)

Usage

```
rValue = smuX.contact.threshold
smuX.contact.threshold = rValue
```

rValue	The resistance above which contact check fails (measured in ohms)

Details

Set the threshold to less than 1 kΩ.

Example

`smua.contact.threshold = 5`	Set the contact check threshold to 5 Ω.

Also see

Contact check connections (on page 2-49)
Contact check measurements (on page 2-40)
reset() (on page 7-197)
smuX.contact.check() (on page 7-239)
smuX.reset() (on page 7-264)

smuX.makebuffer()

This function creates a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar = smuX.makebuffer(bufferSize)
```

bufferVar	The created reading buffer
bufferSize	Maximum number of readings that can be stored

Details

You can use this function to create and dynamically allocate reading buffers. Use *bufferSize* to designate the number of readings the buffer can store.

You can use dynamically allocated reading buffers interchangeably with the smu*X*.nvbuffer*Y* buffers.

To delete a reading buffer, set all references to the reading buffer equal to `nil`, then run the garbage collector (see the `collectgarbage()` function in Standard libraries (on page 6-30)).

Example

```
mybuffer2 = smua.makebuffer(200)
```
Creates a 200 element reading buffer (`mybuffer2`).

Also see

`collectgarbage()` in Base library functions (on page 6-31)
Remote reading buffer programming (on page 3-11)
savebuffer() (on page 7-198)
smuX.nvbufferY (on page 7-263)

smuX.measure.adc

This attribute contains the analog-to-digital converter selection.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	0 (smuX.ADC_INTEGRATE)

Usage

```
adc = smuX.measure.adc
smuX.measure.adc = adc
```

adc	The analog-to-digital (A/D) converter used for measurements; set to one of the following: `0` or smu*X*.`ADC_INTEGRATE`: Integrating A/D converter `1` or smu*X*.`ADC_FAST`: Fast A/D converter
X	Source-measure unit (SMU) channel (for example, `smua.measure.adc` applies to SMU channel A).

Details

When making measurements, the SMU uses one of two types of analog-to-digital converters. This attribute controls which analog to converter is used.

Example

`smua.measure.adc = smua.ADC_FAST`	Selects the fast A/D converter for SMU channel A.

Also see

Analog-to-digital converter (on page 4-1)
smuX.measure.nplc (on page 7-256)

smuX.measure.autorangeY

This attribute stores the measurement autorange setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1 (smu*X*.AUTORANGE_ON)

Usage

```
autoRange = smuX.measure.autorangeY
smuX.measure.autorangeY = autoRange
```

autoRange	The state of the measurement autorange setting; set to one of the following values: ■ `0` or `smuX.AUTORANGE_OFF`: Disabled ■ `1` or `smuX.AUTORANGE_ON`: Enabled ■ `2` or `smuX.AUTORANGE_FOLLOW_LIMIT`: Measure range automatically set to the limit range
Y	SMU measure function (`v` = voltage, `i` = current)

Details

This attribute indicates the measurement autorange state. Its value is `smuX.AUTORANGE_OFF` when the SMU measure circuit is on a fixed range and `smuX.AUTORANGE_ON` when it is in autorange mode.

Setting this attribute to `smuX.AUTORANGE_OFF` puts the SMU on a fixed range. The fixed range is the present SMU measure range.

Setting this attribute to `smuX.AUTORANGE_ON` puts the SMU measure circuit in autorange mode. It remains on its present measure range until the next measurement is requested.

If source high capacitance mode is enabled, current autorange is set to `smuX.AUTORANGE_FOLLOW_LIMIT` and cannot be changed.

Example

```
smua.measure.autorangev = smua.AUTORANGE_ON
```

Enables voltage measurement autoranging.

Also see

Autoranging (on page 2-74)
Range (on page 2-71)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.rangeY (on page 7-258)
smuX.reset() (on page 7-264)

smuX.measure.autozero

This attribute enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	2 (smu*X*.AUTOZERO_AUTO)

Usage

```
azMode = smuX.measure.autozero
smuX.measure.autozero = azMode
```

azMode	Indicates status of autozero; set to one of the following values: ■ 0 or smu*X*.AUTOZERO_OFF: Autozero disabled ■ 1 or smu*X*.AUTOZERO_ONCE: Performs autozero once, then disables autozero ■ 2 or smu*X*.AUTOZERO_AUTO: Automatic checking of reference and zero measurements; an autozero is performed when needed

Details

The integrating analog-to-digital converter (ADC) uses a ratiometric A/D conversion technique. To ensure the accuracy of readings, the instrument must periodically obtain new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture being used for measurements. The Model 2651A uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made. If the reference measurements have expired when a signal measurement is made, the instrument automatically takes two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes more time than normal.

This additional time can cause problems in sweeps and other test sequences in which measurement timing is critical. To avoid the time that is needed for the reference measurements in these situations, you can use the smu*X*.measure.autozero attribute to disable the automatic reference measurements.

Disabling automatic reference measurements may allow the instrument to gradually drift out of specification. To minimize the drift, make a reference and zero measurement immediately before any critical test sequences. You can use the `smuX.AUTOZERO_ONCE` setting to force a refresh of the reference and zero measurements that are used for the present aperture setting.

The Model 2651A stores the reference measurements for the last ten NPLC settings that were used in a reference cache. If an NPLC setting is selected and an entry for it is not in the cache, the oldest (least recently used) entry is discarded to make room for the new entry.

The fast ADC (analog-to-digital converter) does not use the reference measurements associated with this attribute. When this attribute is set to `smuX.AUTOZERO_AUTO`, the fast ADC does not acquire these reference measurements. However, if the fast ADC is selected and you set this attribute to `smuX.AUTOZERO_ONCE`, the fast ADC acquires a new set of reference measurements that are used when the integrating ADC is selected.

Example

```
smua.measure.autozero = smua.AUTOZERO_ONCE
```

Performs autozero once.

Also see

Autozero (on page 2-25)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.nplc (on page 7-256)
smuX.reset() (on page 7-264)

smuX.measure.calibrateY()

This function generates and activates new measurement calibration constants.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smuX.measure.calibrateY(range, cp1Measured, cp1Reference, cp2Measured, cp2Reference)`

Y	SMU measurement function (`v` = voltage, `i` = current)
range	The measurement range to adjust
cp1Measured	The value measured by this SMU for point 1
cp1Reference	The reference measurement for point 1 as measured externally
cp2Measured	The value measured by this SMU for point 2
cp2Reference	The reference measurement for point 2 as measured externally

Details

This function generates and activates new calibration constants for the given range.

The positive and negative polarities of the instrument must be adjusted separately. Use a positive value for the *range* parameter to adjust the positive polarity and a negative value for the *range* parameter to adjust the negative polarity.

All four measurements (*cp1Measured*, *cp1Reference*, *cp2Measured*, and *cp2Reference*) must be made with the calibration set that is active. Corruption of the calibration constants may result if this is ignored.

The new calibration constants are activated immediately but they are not written to nonvolatile memory. Use the `smuX.cal.save()` function to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is only available when calibration is unlocked using `smuX.cal.unlock()`.

Example

`smua.measure.calibratev(1, 1e-4, 1e-5, 0.92, 0.903)`	SMU channel A adjusts voltage measurement using following values: 1 V calibration range, 1e−4 for +zero measurement reading, 1e−5 for +zero DMM measurement reading, 0.92 for +FS measurement reading, and 0.903 for the +FS DMM measurement reading.

Also see

Adjustment (on page 12-15)
smuX.cal.lock() (on page 7-230)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.unlock() (on page 7-236)
smuX.source.calibrateY() (on page 7-268)

smuX.measure.count

This attribute sets the number of measurements made when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1

Usage

```
count = smuX.measure.count
smuX.measure.count = count
```

count	Number of measurements

Details

This attribute controls the number of measurements made any time a measurement is requested. When using a reading buffer with a measure command, this attribute also controls the number of readings to be stored.

If the count is set to a value greater than 1, any measurement delay set by `smuX.measure.delay` only occurs before the first measurement, while the `smuX.measure.interval` controls the interval between successive measurements.

Example

`smua.measure.count = 10`	Sets the measure count to 10.

Also see

reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.delay (on page 7-248)
smuX.measure.interval (on page 7-254)
smuX.measure.overlappedY() (on page 7-257)
smuX.measure.Y() (on page 7-261)
smuX.reset() (on page 7-264)

smuX.measure.delay

This attribute controls the measurement delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	smuX.DELAY_OFF

Usage

```
mDelay = smuX.measure.delay
smuX.measure.delay = mDelay
```

mDelay	Set to the measurement delay value in seconds (for example, to specify an additional 10 ms measurement delay, set the value to `0.010`) You can also set it to one of the following values: ■ `0` or `smua.DELAY_OFF`: No delay ■ `-1` or `smua.DELAY_AUTO`: Automatic delay value

Details

This attribute allows for additional delay (settling time) before making a measurement. If you define the value instead of using the automatic delay value, the delay you set is used regardless of the range.

The `smua.DELAY_AUTO` setting causes a current range-dependent delay to be inserted when a current measurement is requested. This happens when a current measurement command is executed, when the measure action is being performed in a sweep, or after changing ranges during an autoranged measurement.

If `smua.measure.count` is greater than 1, the measurement delay is only inserted before the first measurement.

Example

`smua.measure.delay = 0.010`	Sets a 10 ms measurement delay.

Also see

Measure auto delay (on page 2-73)
reset() (on page 7-197)
smuX.measure.count (on page 7-247)
smuX.measure.delayfactor (on page 7-249)
smuX.source.delay (on page 7-270)
smuX.reset() (on page 7-264)

smuX.measure.delayfactor

This attribute stores a multiplier to the delays that are used when `smuX.measure.delay` is set to `smuX.DELAY_AUTO`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	1

Usage

```
delayFactor = smuX.measure.delayfactor
smuX.measure.delayfactor = delayFactor
```

delayFactor	The delay factor multiplier

Details

The delay factor is only applied when `smuX.measure.delay = smuX.DELAY_AUTO`.

This attribute can be set to a value less than 1 (for example, 0.5) to decrease the automatic delay.

This attribute can be set to a value greater than 1 (for example, 1.5 or 2.0) to increase the automatic delay.

Setting this attribute to zero disables delays when `smuX.measure.delay = smuX.DELAY_AUTO`.

Example

`smua.measure.delayfactor = 2.0`	Doubles the measure delay.

Also see

Measure auto delay (on page 2-73)
reset() (on page 7-197)
smuX.measure.delay (on page 7-248)
smuX.reset() (on page 7-264)

smuX.measure.filter.count

This command sets the number of measured readings that are required to yield one filtered measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1

Usage

```
filterCount = smuX.measure.filter.count
smuX.measure.filter.count = filterCount
```

filterCount	The number of readings required for each filtered measurement (1 to 100)

Details

This attribute sets the size of the stack used for filtered measurements.

Example

```
smua.measure.filter.count = 10
smua.measure.filter.type = smua.FILTER_MOVING_AVG
smua.measure.filter.enable = smua.FILTER_ON
```

Sets the filter count to 10.
Sets the filter type to moving average.
Enables the filter.

Also see

Filters (on page 3-3)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.filter.enable (on page 7-251)
smuX.measure.filter.type (on page 7-252)
smuX.reset() (on page 7-264)

smuX.measure.filter.enable

This command enables or disables filtered measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	0 (smu*X*.FILTER_OFF)

Usage

```
filterState = smuX.measure.filter.enable
smuX.measure.filter.enable = filterState
```

filterState	The filter status; set to one of the following values: ■ 0 or smu*X*.FILTER_OFF: Disables the filter ■ 1 or smu*X*.FILTER_ON: Enables the filter

Details

This command enables or disables the filter.

Example

```
smua.measure.filter.count = 10
smua.measure.filter.type = smua.FILTER_MOVING_AVG
smua.measure.filter.enable = smua.FILTER_ON
```

Sets the filter count to 10.
Sets the filter type to moving average.
Enables the filter.

Also see

Filters (on page 3-3)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.filter.count (on page 7-250)
smuX.measure.filter.type (on page 7-252)
smuX.reset() (on page 7-264)

smuX.measure.filter.type

This command sets the type of filter used for measurements when the measurement filter is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1 (smu*X*.FILTER_REPEAT_AVG)

Usage

```
filterType = smuX.measure.filter.type
smuX.measure.filter.type = filterType
```

filterType	The filter type to use when filtering is enabled. Set to one of the following values: ■ 0 or smu*X*.FILTER_MOVING_AVG: Selects the moving filter ■ 1 or smu*X*.FILTER_REPEAT_AVG: Selects the repeat filter ■ 2 or smu*X*.FILTER_MEDIAN: Selects the median filter

Details

The Model 2651A provides a moving average, repeating average, and median filter type.

For the repeating filter, the stack (filter count) is filled, and the conversions are averaged to yield a reading. The stack is then cleared, and the process starts over.

The moving average filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the measurement conversions are averaged, yielding a reading. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The stack is re-averaged, yielding a new reading.

The median filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the reading nearest to the middle is returned. For each subsequent conversion placed into the stack, the oldest reading is discarded. The stack is then re-sorted, yielding a new reading. If the filter count is an even number, the reading returned is the average of the two middle readings.

Example

```
smua.measure.filter.count = 10
smua.measure.filter.type = smua.FILTER_MOVING_AVG
smua.measure.filter.enable = smua.FILTER_ON
```

Sets the filter count to 10.
Sets the filter type to moving average.
Enables the filter.

Also see

Filters (on page 3-3)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.filter.count (on page 7-250)
smuX.measure.filter.enable (on page 7-251)
smuX.reset() (on page 7-264)

smua.measure.highcrangedelayfactor

This attribute contains a delay multiplier that is only used during range changes when the high-capacitance mode is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	10

Usage

```
delayFactor = smua.measure.highcrangedelayfactor
smua.measure.highcrangedelayfactor = delayFactor
```

delayFactor	The delay factor; set to a value between 1 and 99

Details

This delay multiplier is only active when the high-capacitance mode is active.

Example

`smua.measure.highcrangedelayfactor = 5`	Increases the delay used during range changes by a factor of 5.

Also see

High-capacitance mode (on page 3-73)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.reset() (on page 7-264)
smuX.source.highc (on page 7-271)

smuX.measure.interval

This attribute sets the interval between multiple measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	0 (0 s)

Usage

```
interval = smuX.measure.interval
smuX.measure.interval = interval
```

interval	The interval value (in seconds); set to a value between 0 and 1

Details

This attribute sets the time interval between measurements when `smuX.measure.count` is set to a value greater than 1. The SMU attempts to start each measurement when scheduled. If the SMU cannot keep up with the interval setting, measurements are made as quickly as possible.

If filtered measurements are being made, the time interval is from the start of the first measurement for the filtered reading to the first measurement for a subsequent filtered reading. Extra measurements made to satisfy a filtered reading are not paced by this interval.

Example

```
smua.measure.interval = 0.5
```

Sets the measure interval for SMU channel A to 0.5 s.

Also see

reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.count (on page 7-247)
smuX.reset() (on page 7-264)

smuX.measure.lowrangeY

This attribute sets the lowest measurement range that is used when the instrument is autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	Current: 100e-9 (100 nA) Voltage: 100e-3 (100 mV)

Usage

```
lowRange = smuX.measure.lowrangeY
smuX.measure.lowrangeY = lowRange
```

lowRange	The lowest voltage or current measurement range used during autoranging
Y	SMU measure function (`v` = voltage, `i` = current)

Details

This attribute is used with autoranging to put a lower bound on the range used. Since lower ranges generally require greater settling times, setting a lowest range limit might make measurements require less settling time.

If the instrument is set to autorange and it is on a range lower than the one specified, the range is changed to the *lowRange* range value.

Example

`smua.measure.lowrangev = 1`	Sets voltage low range for SMU channel A to 1 V.

Also see

Range (on page 2-71)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.autorangeY (on page 7-244)
smuX.reset() (on page 7-264)

smuX.measure.nplc

This command sets the integration aperture for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	1.0

Usage

```
nplc = smuX.measure.nplc
smuX.measure.nplc = nplc
```

nplc	The integration aperture; set from 0.001 to 25

Details

When making measurements, the SMU uses one of two types of analog-to-digital converters (ADC): integrating or fast. This attribute controls the integration aperture for the integrating ADC. This attribute is not used when the fast ADC is selected.

The integration aperture is based on the number of power line cycles (NPLC), where 1 PLC for 60 Hz is 16.67 ms (1/60) and 1 PLC for 50 Hz is 20 ms (1/50).

Example

`smua.measure.nplc = 0.5`	Sets the integration time to 0.5.

Also see

reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.adc (on page 7-243)
smuX.reset() (on page 7-264)
Speed (on page 2-78)

smua.measure.overlappedY()

This function starts an asynchronous (background) measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
smua.measure.overlappedY(rbuffer)
smua.measure.overlappediv(ibuffer, vbuffer)
```

Y	SMU measurement type (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
rbuffer	A reading buffer object where the readings are stored
ibuffer	A reading buffer object where current readings are stored
vbuffer	A reading buffer object where voltage readings are stored

Details

This function starts a measurement and returns immediately. The measurements, as they are performed, are stored in a reading buffer (along with any other information that is being acquired). If the instrument is configured to return multiple readings where one is requested, the readings are available as they are made. Measurements are in the following units of measure: `v` = volts, `i` = amperes, `r` = ohms, `p` = watts.

The second form of this function, `smua.measure.overlappediv()`, stores current readings in *ibuffer* and voltage readings in *vbuffer*.

This function is an overlapped command. Script execution continues while the measurements are made in the background. Attempts to access result values that have not yet been generated cause the script to block and wait for the data to become available. The `waitcomplete()` function can also be used to wait for the measurements to complete before continuing.

If a given reading buffer contains any data, it is cleared before making any measurements, unless the reading buffer has been configured to append data.

Example

```
smua.measure.overlappedv(smua.nvbuffer1)
```

Starts background voltage measurements.

Also see

Reading buffers (on page 3-6)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
waitcomplete() (on page 7-459)

smuX.measure.rangeY

This attribute contains the positive full-scale value of the measurement range for voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Saved setup	Current: 100e-9 (100 nA) Voltage: 100e-3 (100 mV)

Usage

```
rangeValue = smuX.measure.rangeY
smuX.measure.rangeY = rangeValue
```

rangeValue	Set to the maximum expected voltage or current to be measured
X	Source-measure unit (SMU) channel (for example, `smua.measure.rangev` applies to SMU channel A)
Y	SMU measurement function (`v` = voltage, `i` = current)

Details

Reading this attribute returns the positive full-scale value of the measurement range that the SMU is currently using. Assigning a value to this attribute sets the SMU on a fixed range large enough to measure the assigned value. The instrument selects the best range for measuring a value of *rangeValue*.

This attribute is primarily intended to eliminate the time that is required by the automatic range selection performed by a measuring instrument. Because selecting a fixed range prevents autoranging, an overrange condition can occur. For example, measuring 3.0 V on the Model 2651A 1 V range causes an overrange. The value 9.91000E+37 is returned when this occurs.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is locked to be the same as the source range. However, the setting for the measure range is retained. If the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

Model 2651A example: Assume the source function is voltage. The source range is 1 V and you set the measure range for 10 V. Since the source range is 1 V, the SMU performs voltage measurements on the 1 V range. If you now change the source function to current, voltage measurements are performed on the 10 V range.

Explicitly setting a measure range disables measure autoranging for that function. Autoranging is controlled separately for each source and measurement function: source voltage, source current, measure voltage and measure current. Autoranging is enabled for all four by default.

Changing the range while the output is off does not update the hardware settings, but querying returns the range setting that is used when the output is turned on. Setting a range while the output is on takes effect immediately.

With measure autoranging enabled, the range is changed only when a measurement is made. Querying the range after a measurement returns the range selected for that measurement.

Example

`smua.measure.rangev = 0.5`	Selects the 1 V measurement range.

Also see

Range (on page 2-71)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.autorangeY (on page 7-244)
smuX.reset() (on page 7-264)
smuX.source.rangeY (on page 7-280)

smua.measure.rel.enableY

This attribute turns relative measurements on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	0 (smua.REL_OFF)

Usage

```
relEnable = smua.measure.rel.enableY
smua.measure.rel.enableY = relEnable
```

relEnable	Relative measurement control. Set *relEnable* to one of the following values: ■ 0 or smua.REL_OFF: Disables relative measurements ■ 1 or smua.REL_ON: Enables relative measurements
Y	SMU measurement function (v = voltage, i = current, r = resistance, p = power)

Details

This attribute enables or disables relative measurements. When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value specified by smua.measure.rel.levelY. Each returned measured relative reading is the result of the following calculation:

Relative reading = Actual measured reading − Relative offset value

Example

`smua.measure.rel.enablev = smua.REL_ON`	Enables relative voltage measurements.

Also see

Relative offset (on page 3-1)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.rel.levelY (on page 7-260)
smuX.reset() (on page 7-264)

smua.measure.rel.levelY

This attribute sets the offset value for relative measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset SMU reset Recall setup	Not saved	0

Usage

```
relValue = smua.measure.rel.levelY
smua.measure.rel.levelY = relValue
```

relValue	Relative measurement offset value
Y	SMU measurement function (`v` = voltage, `i` = current, `r` = resistance, `p` = power)

Details

This attribute specifies the offset value used for relative measurements. When relative measurements are enabled (see `smua.measure.rel.enableY`), all subsequent measured readings are offset by the value of this attribute. Each returned measured relative reading is the result of the following calculation:

Relative reading = Actual measured reading - Relative offset value

Example

```
smua.measure.rel.levelv = smua.measure.v()
```

Performs a voltage measurement and then uses it as the relative offset value.

Also see

Relative offset (on page 3-1)
reset() (on page 7-197)
smuX.measure.rel.enableY (on page 7-259)
smuX.reset() (on page 7-264)

smuX.measure.Y()

This function makes one or more measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = smuX.measure.Y()
reading = smuX.measure.Y(readingBuffer)
iReading, vReading = smuX.measure.iv()
iReading, vReading = smuX.measure.iv(iReadingBuffer)
iReading, vReading = smuX.measure.iv(iReadingBuffer, vReadingBuffer)
```

reading	Returned value of the last (or only) reading of the measurement process
Y	SMU measurement function (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
readingBuffer	A reading buffer object where all readings are stored
iReading	The last reading of the current measurement process
vReading	The last reading of the voltage measurement process
iReadingBuffer	A reading buffer object where current readings are stored
vReadingBuffer	A reading buffer object where voltage readings are stored

Details

If you use this function without specifying a reading buffer, it makes one measurement and returns that measurement as *reading*. To use the additional information that is acquired while making a measurement or to return multiple readings, specify a reading buffer. If the instrument is configured to return multiple readings for a measurement and *readingBuffer* is specified, all readings are available in *readingBuffer*, but only the last measurement is returned as *reading*.

Measurements are in the following units of measure:

- `v` = volts
- `i` = amperes
- `r` = ohms
- `p` = watts

The `smuX.measure.iv()` function returns the last actual current measurement and voltage measurement as *iReading* and *vReading*, respectively. Additionally, it can store current and voltage readings if buffers are provided (*iReadingBuffer* and *vReadingBuffer*).

The `smuX.measure.count` attribute determines how many measurements are performed. When using a reading buffer, it also determines the number of readings to store in the buffer. If a reading buffer is not specified, the SMU ignores the `smuX.measure.count` attribute and only makes one measurement.

The *readingBuffer* is cleared before making any measurements unless the buffer is configured to append data.

Example

`smua.measure.count = 10` `smua.measure.v(smua.nvbuffer1)`	Makes 10 voltage measurements using SMU channel A and stores them in a buffer.

Also see

Reading buffers (on page 3-6)
smuX.measure.count (on page 7-247)
smuX.measure.overlappedY() (on page 7-257)
smuX.nvbufferY (on page 7-263)

smuX.measureYandstep()

This function makes one or two measurements and then steps the source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = smuX.measureYandstep(sourceValue)
iReading, vReading = smuX.measureivandstep(sourceValue)
```

reading	The measured reading before stepping the source
Y	SMU measurement function (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
sourceValue	Source value to be set after the measurement is made
iReading	The current reading before stepping the source
vReading	The voltage reading before stepping the source

Details

The `smuX.measureYandstep()` function makes a measurement and then sets the source to *sourceValue*. Usage of the `smuX.measureivandstep()` function is similar, but makes two measurements simultaneously, one for current (`i`) and one for voltage (`v`).

Measurements are in the following units of measure: `v` = volts, `i` = amperes, `r` = ohms, `p` = watts.

Make sure the specified source value is appropriate for the selected source function. For example, if the source voltage function is selected, then *sourceValue* is expected to be a new voltage level.

Both source and measure autorange must be disabled before using this function. This function cannot be used if source high capacitance mode is enabled (high capacitance mode requires autoranging to be enabled).

This function is provided for very fast execution of source-measure loops. The measurement is made before the source is stepped. Before using this function, and before any loop this function may be used in, set the source value to its initial level.

Example

```
local ivalues = {}
smua.source.rangev = 1
smua.source.levelv = 0
smua.measure.rangei = 0.01
smua.source.output = smua.OUTPUT_ON
for index = 1, 10 do
   ivalues[index] = smua.measureiandstep(index / 10)
end
ivalues[11] = smua.measure.i()
```

This use of the SMU channel A measure and step function measures current starting at a source value of 0 V. After each current measurement, the source is stepped 100 mV for the next current measurement. The final source level is 1 V, where current is again measured.

Also see

smuX.measure.autorangeY (on page 7-244)
smuX.measure.Y() (on page 7-261)
smuX.source.autorangeY (on page 7-267)
smuX.trigger.source.limitY (on page 7-301)
smuX.trigger.source.linearY() (on page 7-302)
smuX.trigger.source.listY() (on page 7-303)
smuX.trigger.source.logY() (on page 7-304)
Sweep Operation (on page 3-21)

smuX.nvbufferY

This attribute contains a dedicated reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	See **Details**	Not applicable

Usage

bufferVar = smu*X*.nvbuffer*Y*

bufferVar	The dedicated reading buffer
Y	SMU nonvolatile buffer (1 or 2)

Details

Each SMU channel contains two dedicated reading buffers: smu*X*.nvbuffer1 and smu*X*.nvbuffer2.

All routines that return measurements can also store them in either reading buffer. Overlapped measurements are always stored in a reading buffer. Synchronous measurements return either a single-point measurement or can be stored in a reading buffer if passed to the measurement command.

The dedicated reading buffers can be saved to internal nonvolatile memory (to retain data between power cycles) using the smu*X*.savebuffer() function.

Example

`smua.measure.overlappedv(smua.nvbuffer1)`	Store voltage readings from SMU channel A into SMU channel A dedicated reading buffer 1.

Also see

Configuring and running sweeps (on page 3-31)
Reading buffers (on page 3-6)
savebuffer() (on page 7-198)
smuX.makebuffer() (on page 7-243)
smuX.measure.overlappedY() (on page 7-257)
smuX.savebuffer() (on page 7-265)
smuX.trigger.measure.action (on page 7-295)
smuX.trigger.measure.set() (on page 7-296)
smuX.trigger.measure.stimulus (on page 7-297)
smuX.trigger.measure.Y() (on page 7-298)

smuX.reset()

This function turns off the output and resets the commands that begin with `smuX.` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.reset()
```

Details

This function turns off the output and returns the specified SMU to its default settings.

Example

`smua.reset()`	Turns off the output and resets SMU to its default settings.

Also see

reset() (on page 7-197)

smuX.savebuffer()

This function saves one source-measure unit (SMU) dedicated reading buffer to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.savebuffer(smuX.nvbufferY)
```

Y	SMU dedicated reading buffer (1 or 2)

Details

When the instrument is turned off and back on, the dedicated reading buffers are restored from nonvolatile memory to their last saved values.

Example

`smua.savebuffer(smua.nvbuffer1)`	Saves buffer 1 to internal memory.

Also see

Reading buffers (on page 3-6)
savebuffer() (on page 7-198)
smuX.nvbufferY (on page 7-263)

smuX.sense

This attribute contains the state of the sense mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smu*X*.SENSE_LOCAL)

Usage

```
senseMode = smuX.sense
smuX.sense = senseMode
```

senseMode	The sense mode; set to one of the following: ■ 0 or smu*X*.SENSE_LOCAL: Selects local sense (2-wire) ■ 1 or smu*X*.SENSE_REMOTE: Selects remote sense (4-wire) ■ 3 or smu*X*.SENSE_CALA: Selects calibration sense mode

Details

Source-measure operations are performed using either 2-wire local sense connections or 4-wire remote sense connections. Writing to this attribute selects the sense mode.

The smu*X*.SENSE_CALA mode is only used for calibration and may only be selected when calibration is enabled.

The sense mode can be changed between local and remote while the output is on.

The calibration sense mode cannot be selected while the output is on.

Resetting the instrument selects the local sense mode.

Example

`smua.sense = smua.SENSE_REMOTE`	Selects remote sensing.

Also see

2-wire local sensing connections (on page 2-47)
4-wire remote sensing connections (on page 2-47)
Sense mode selection (on page 2-64)

smuX.source.autorangeY

This attribute contains the state of the source autorange control (on/off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	1 (smu*X*.AUTORANGE_ON)

Usage

```
sourceAutorange = smuX.source.autorangeY
smuX.source.autorangeY = sourceAutorange
```

sourceAutorange	The state of the source autorange control. Set to one of the following: ■ `0` or `smuX.AUTORANGE_OFF`: Disables source autorange ■ `1` or `smuX.AUTORANGE_ON`: Enables source autorange
Y	SMU source function (`v` = voltage, `i` = current)

Details

This attribute indicates the source autorange state. Its value is `smuX.AUTORANGE_OFF` when the SMU source circuit is on a fixed range and `smuX.AUTORANGE_ON` when it is in autorange mode.

Setting this attribute to `smuX.AUTORANGE_OFF` puts the SMU on a fixed source range. The fixed range used is the present SMU source circuit range.

Setting this attribute to `smuX.AUTORANGE_ON` puts the SMU source circuit into autorange mode. If the source output is on, the SMU immediately changes range to the range most appropriate for the value being sourced if that range is different than the present SMU range.

Autorange is disabled if the source level is edited from the front panel. Setting the source range also turns off autorange when set by using the `smuX.source.rangeY` attribute.

Resetting the instrument selects the `smuX.AUTORANGE_ON`.

Example

`smua.source.autorangev = smua.AUTORANGE_ON`	Enables volts source autorange.

Also see

smuX.measure.autorangeY (on page 7-244)
smuX.source.rangeY (on page 7-280)

smuX.source.calibrateY()

This function generates and activates new source calibration constants.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.source.calibrateY(range, cp1Expected, cp1Reference, cp2Expected, cp2Reference)
```

Y	SMU source function (`v` = voltage, `i` = current)
range	The measurement range to adjust
cp1Expected	The source value set for point 1
cp1Reference	The reference measurement for point 1 as measured externally
cp2Expected	The source value set for point 2
cp2Reference	The reference measurement for point 2 as measured externally

Details

This function generates and activates new calibration constants for the given range.

The positive and negative polarities of the source must be adjusted separately. Use a positive value for *range* to adjust the positive polarity and a negative value for range to adjust the negative polarity. Do not use 0.0 for a negative point; 0.0 is considered to be a positive number.

Typically, the two points that are used are near zero for point 1 and 90% of full scale for point 2. Full scale for point 2 should be avoided if the source of the SMU is substantially out of calibration.

The two reference measurements must be made with the source using the active calibration set. For example, source a value, measure it, and do not change the active calibration set before issuing this command.

The new calibration constants are activated immediately but they are not written to nonvolatile memory. Use the `smuX.cal.save()` function to save the new constants to nonvolatile memory. The active calibration constants stay in effect until the instrument is power cycled or a calibration set is loaded from nonvolatile memory with the `smuX.cal.restore()` function.

This function is only available when calibration is unlocked using `smuX.cal.unlock()`.

Example

`smua.source.calibratev(1, 1e-10, 1e-5, 0.9, 0.903)`	SMU channel A adjust the voltage source using the following values: adjust the 1 V range, 1e−10 for +zero source output value, 1e−5 for +zero DMM measurement reading, 0.9 for +FS source output value, and 0.903 for the +FS DMM measurement reading.

Also see

Calibration (on page 12-1)
smuX.cal.restore() (on page 7-233)
smuX.cal.save() (on page 7-234)
smuX.cal.unlock() (on page 7-236)
smuX.measure.calibrateY() (on page 7-246)

smuX.source.compliance

This attribute contains the state of source compliance.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
compliance = smuX.source.compliance
```

compliance	The state of source compliance: ■ `true`: Indicates that the limit function is in control of the source (source in compliance) ■ `false`: Indicates that the source function is in control of the output (source not in compliance)

Details

Reading this attribute updates the status model and the front panel with generated compliance information. See Current Limit (ILMT) in the status model diagram for the Measurement event registers (on page 15-7). The Voltage Limit (VLMT) is not shown in the status model diagram for the Measurement event registers but is similar to the Current Limit (ILMT).

Example

```
compliance = smua.source.compliance

print(compliance)
```

Reads the source compliance state.
Output:
`true`
This output indicates that a configured limit has been reached (voltage, current, or power limit).

Also see

smuX.source.limitY (on page 7-273)

smuX.source.delay

This attribute contains the source delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smu*X*.DELAY_OFF)

Usage

```
sDelay = smuX.source.delay
smuX.source.delay = sDelay
```

sDelay	Set to the source delay value (for example, to specify an additional 10 ms source delay, set the value to `0.010`); you can also set it one of the following values: ■ `0` or `smuX.DELAY_OFF`: No delay ■ `-1` or `smuX.DELAY_AUTO`: Automatic delay value

Details

This attribute allows for additional delay (settling time) after an output step.

The `smuX.DELAY_AUTO` setting causes a range-dependent delay to be inserted when the source is changed. Range-dependent delays are based on the output settling time values as defined in the Model 2651A specifications.

Example

```
smua.source.delay = smua.DELAY_AUTO
```

Sets the delay to automatic (a range-dependent delay is inserted whenever the source is changed).

Also see

reset() (on page 7-197)
smuX.measure.count (on page 7-247)
smuX.measure.delay (on page 7-248)
smuX.reset() (on page 7-264)

smuX.source.func

This attribute contains the source function, which can be voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	1 (smu*X*.OUTPUT_DCVOLTS)

Usage

```
sFunction = smuX.source.func
smuX.source.func = sFunction
```

sFunction	The source function; set to one of the following values: ■ 0 or smu*X*.OUTPUT_DCAMPS: Selects the current source function ■ 1 or smu*X*.OUTPUT_DCVOLTS: Selects the voltage source function

Details

Reading this attribute indicates the output function of the source. Setting this attribute configures the SMU as either a voltage source or a current source.

Example

```
smua.source.func = smua.OUTPUT_DCAMPS
```

Sets the source function to be a current source.

Also see

smuX.source.levelY (on page 7-272)
smuX.source.output (on page 7-278)

smuX.source.highc

This attribute enables or disables high-capacitance mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smu*X*.DISABLE)

Usage

```
highC = smuX.source.highc
smuX.source.highc = highc
```

highC	The state of the high-capacitance mode; set to one of the following values: ■ 0 or smu*X*.DISABLE: Disables high-capacitance mode ■ 1 or smu*X*.ENABLE: Enables high-capacitance mode

Details

When enabled, the high-capacitance mode has the following effects on the SMU settings:

- `smuX.measure.autoranei` is set to `smuX.AUTORANGE_FOLLOW_LIMIT` and cannot be changed
- Current ranges below 1 µA are not accessible
- If `smuX.source.limiti` is less than 1 µA, it is raised to 1 µA
- If `smuX.source.rangei` is less than 1 µA, it is raised to 1 µA
- If `smuX.source.lowrangei` is less than 1 µA, it is raised to 1 µA
- If `smuX.measure.lowrangei` is less than 1 µA, it is raised to 1 µA

Example

`smua.source.highc = smua.ENABLE`	Activates high-capacitance mode.

Also see

High-capacitance mode (on page 3-73)

smuX.source.levelY

This attribute sets the source level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0

Usage

```
sourceLevel = smuX.source.levelY
smuX.source.levelY = sourceLevel
```

sourceLevel	The source value; set to one of the following values: Voltage: 0 V to ±40 V Current: 0 A to ±20 A
X	Source-measure unit (SMU) channel (for example, `smua.source.levelv` applies to SMU channel A)
Y	SMU source function (`v` = voltage, `i` = current)

Details

This attribute configures the output level of the voltage or current source.

If the source is configured as a voltage source and the output is on, the new `smuX.source.levelv` setting is sourced immediately. If the output is off or the source is configured as a current source, the voltage level is sourced when the source is configured as a voltage source and the output is turned on.

If the source is configured as a current source and the output is on, the new `smuX.source.leveli` setting is sourced immediately. If the output is off or the source is configured as a voltage source, the current level is sourced when the source is configured as a current source and the output is turned on.

The sign of *`sourceLevel`* dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

The `reset()` function sets the source levels to 0 V and 0 A.

Example

`smua.source.levelv = 1`	Sets voltage source of SMU channel A to 1 V.

Also see

smuX.source.compliance (on page 7-269)
smuX.source.func (on page 7-271)
smuX.source.output (on page 7-278)
Source-measure concepts (on page 4-2)

smuX.source.limitY

This attribute sets compliance limits.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Limit voltage: 20 (20 V) Limit current: 100e-3 (100 mA) Limit power: 0 (disabled)

Usage

```
limit = smuX.source.limitY
smuX.source.limitY = limit
```

`limit`	The compliance limit value; set to one of the following values: Voltage compliance: 0 V to 40 V Current compliance: 0 A to 20 A
`X`	Source-measure unit (SMU) channel (for example, `smua.source.limitv` applies to SMU channel A)
`Y`	SMU function (`v` = voltage, `i` = current, `p` = power)

Details

Use the `smuX.source.limiti` attribute to limit the current output of the voltage source. Use `smuX.source.limitv` to limit the voltage output of the current source. The SMU always uses autoranging for the limit setting. Use the `smuX.source.limitp` attribute to limit the output power of the source.

Set this attribute in the test sequence before the turning the source on.

Using a limit value of 0 results in error code 1102, "Parameter too small," for `v` and `i`. Setting this attribute to zero disables power compliance for `p`. When setting the power compliance limit to a nonzero value, the SMU adjusts the source limit where appropriate to limit the output to the specified power. The SMU uses the lower of the programmed compliance value (the compliance level that is if power compliance is disabled) or the limit calculated from the power compliance setting.

Reading this attribute indicates the presently set compliance value. Use `smuX.source.compliance` to read the state of source compliance.

Example

`smua.source.limitv = 15`	Sets the voltage limit to 15 V.

Also see

DUT test connections (on page 2-45)
smuX.source.compliance (on page 7-269)
smuX.source.func (on page 7-271)
smuX.source.output (on page 7-278)

smuX.source.lowrangeY

This attribute sets the lowest source range that is used during autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Voltage: 100e-3 (100 mV) Current: 100e-9 (100 nA)

Usage

```
sourceRangeLow = smuX.source.lowrangeY
smuX.source.lowrangeY = sourceRangeLow
```

sourceRangeLow	Set to the lowest voltage (in volts) or current (in amperes) range to be used
X	Source-measure unit (SMU) channel (for example, `smua.source.lowrangev` applies to SMU channel A)
Y	SMU source function (v = voltage, i = current)

Details

This attribute is used with source autoranging to put a lower bound on the range that is used.

If the instrument is set to autorange and it is on a range lower than the one specified by *sourceRangeLow*, the source range is changed to the range specified by *sourceRangeLow*.

Example

`smua.source.lowrangev = 1`	Sets volts low range for Model 2651A SMU channel A to 1 V. This prevents the source from using the 100 mV range when sourcing voltage.

Also see

smuX.source.autorangeY (on page 7-267)

smuX.source.offfunc

This attribute sets the source function that is used (source 0 A or 0 V) when the output is turned off and the source-measure unit (SMU) is in normal output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	1 (smuX.OUTPUT_DCVOLTS)

Usage

```
offfunc = smuX.source.offfunc
smuX.source.offfunc = offfunc
```

offfunc	Set to the source function to be used when the output is off and the SMU is in normal output-off mode. Set to one of the following values: ■ `0` or `smuX.OUTPUT_DCAMPS`: Source 0 A ■ `1` or `smuX.OUTPUT_DCVOLTS`: Source 0 V

Details

This attribute controls the source function used when the output is turned off and `smuX.source.offmode` is set to `smuX.OUTPUT_NORMAL`.

Set this attribute to `smuX.OUTPUT_DCVOLTS` for the source to be a 0 V source when the output is off (`smuX.source.offlimiti` is used).

Set it to `smuX.OUTPUT_DCAMPS` for the source to be a 0 A source when the output is off (`smuX.source.offlimitv` is used).

Example

```
smua.source.offmode = smua.OUTPUT_NORMAL
smua.source.offfunc = smua.OUTPUT_DCVOLTS
```

Sets the normal output-off mode to source 0 V when the output is turned off.

Also see

Output-off states (on page 2-65)
smuX.source.offlimitY (on page 7-276)
smuX.source.offmode (on page 7-277)
smuX.source.output (on page 7-278)

smuX.source.offlimitY

This attribute sets the limit (current or voltage) used when the source-measure unit (SMU) is in normal output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Current: 1e-3 (1 mA) Voltage: 40 (40 V)

Usage

```
sourceLimit = smuX.source.offlimitY
smuX.source.offlimitY = sourceLimit
```

sourceLimit	Set to the limit to be used when the SMU is in normal output-off mode
Y	SMU source function (`v` = voltage, `i` = current)

Details

Setting the current limit to lower than 1 mA may interfere with operation of the contact check function. See `smuX.contact.check()` and `smuX.contact.r()` for details.

Example

```
smua.source.offlimiti = 10e-3
```

Changes the normal output-off mode limit to 10 mA.

Also see

smuX.contact.check() (on page 7-239)
smuX.contact.r() (on page 7-240)
smuX.source.offfunc (on page 7-275)
smuX.source.offmode (on page 7-277)

smuX.source.offmode

This attribute sets the source output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smuX.OUTPUT_NORMAL)

Usage

```
sourceOffMode = smuX.source.offmode
smuX.source.offmode = sourceOffMode
```

sourceOffMode	The output-off setting; set to one of the following values: ■ 0 or smuX.OUTPUT_NORMAL: Configures the source function according to smuX.source.offfunc attribute ■ 1 or smuX.OUTPUT_ZERO: Configures source to output 0 V as smuX.OUTPUT_NORMAL with different compliance handling (see the **Details** below) ■ 2 or smuX.OUTPUT_HIGH_Z: Opens the output relay when the output is turned off

Details

Reading this attribute returns the output-off mode of the source. Setting this attribute configures the SMU output-off mode.

The default *sourceOffMode* is smuX.OUTPUT_NORMAL. In this mode, the source function is configured according to the smuX.source.offfunc attribute. The smuX.source.offfunc attribute controls whether the SMU is configured as a 0 V voltage source or a 0 A current source. When the SMU is operating as a 0 A current source, the smuX.source.offlimitv attribute sets the voltage limit (similar to how the smuX.source.offlimiti attribute sets the current limit when the SMU is operating as a 0 V voltage source).

When the *sourceOffMode* is set to smuX.OUTPUT_ZERO, the source is configured to output 0 V just as smuX.OUTPUT_NORMAL mode with smuX.source.offfunc = smuX.OUTPUT_DCVOLTS. If the source function is voltage, the current limit is not changed. If the source function is current, the current limit is set to the current source level or 10 percent of the current source range, whichever is greater.

When *sourceOffMode* is set to smuX.OUTPUT_HIGH_Z, the SMU opens the output relay when the output is turned off.

Example

```
smua.source.offmode = smua.OUTPUT_HIGH_Z
```

Sets the output-off mode to open the output relay when the output is turned off.

Also see

Output-off states (on page 2-65)
smuX.source.offfunc (on page 7-275)
smuX.source.offlimitY (on page 7-276)
smuX.source.output (on page 7-278)

smuX.source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.OUTPUT_OFF)

Usage

```
sourceOutput = smuX.source.output
smuX.source.output = sourceOutput
```

sourceOutput	The output state setting of the source; set to one of the following values: ■ `0` or `smuX.OUTPUT_OFF`: Turns off the source output ■ `1` or `smuX.OUTPUT_ON`: Turns on the source output ■ `2` or `smuX.OUTPUT_HIGH_Z`: Turns off the output in high Z mode (allows you to go to high Z mode without first setting the `smuX.source.offmode` attribute to `smuX.OUTPUT_HIGH_Z`)

Details

Reading this attribute returns the output state of the source. Setting this attribute switches the output of the source on or off.

When the output is switched on, the SMU sources either voltage or current, as set by `smuX.source.func`.

Setting this attribute to `smuX.OUTPUT_HIGH_Z` causes the output to turn off and go to the High Z mode. If the `smuX.source.output` is read after setting this attribute to `smuX.OUTPUT_HIGH_Z`, it returns `0`.

Example

`smua.source.output = smua.OUTPUT_ON`	Turns on the source output.

Also see

DUT test connections (on page 2-45)
smuX.source.func (on page 7-271)
smuX.source.offmode (on page 7-277)

smuX.source.outputenableaction

This attribute controls output enable action of the source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smu*X*.OE_NONE)

Usage

```
outputAction = smuX.source.outputenableaction
smuX.source.outputenableaction = outputAction
```

outputAction	The output enable action of the source; set to one of the following values: ■ `0` or `smuX.OE_NONE`: No action ■ `1` or `smuX.OE_OUTPUT_OFF`: Turns the source output off
X	Source-measure unit (SMU) channel (for example, `smua.source.outputenableaction` applies to SMU channel A)

Details

This attribute controls the action the SMU takes when the output enable line is deasserted.

When set to `smuX.OE_NONE`, the SMU takes no action when the output enable line goes low (deasserted).

When set to `smuX.OE_OUTPUT_OFF` and the output enable line is de-asserted, the SMU turns its output off as if the `smuX.source.output = smuX.OUTPUT_OFF` command had been received.

The SMU does not automatically turn its output on when the output enable line returns to the high state.

If the output enable line is not asserted when this attribute is set to `smuX.OE_OUTPUT_OFF` and the output is on, the output turns off immediately.

Detection of the output enable line going low does not abort any running scripts. This may cause execution errors.

Example

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

Sets SMU channel A to turn off the output if the output enable line goes low (deasserted).

Also see

smuX.source.offmode (on page 7-277)
smuX.source.output (on page 7-278)

smuX.source.rangeY

This attribute contains the source range.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	Voltage: 100e-3 (100 mV) Current: 100e-9 (100 nA)

Usage

```
rangeValue = smuX.source.rangeY
smuX.source.rangeY = rangeValue
```

rangeValue	Set to the maximum expected voltage or current to be sourced
Y	SMU source function (`v` = voltage, `i` = current)

Details

This attribute contains a value that sets the source-measure unit (SMU) to a fixed range large enough to source the value. When read, the attribute contains the range the instrument is presently on when in autorange.

Assigning a value to this attribute sets the SMU to a fixed range large enough to source the assigned value. The instrument selects the best range for sourcing a value of *rangeValue*.

Reading this attribute returns the positive full-scale value of the source range the SMU is currently using. With source autoranging enabled, the output level controls the range. Querying the range after the level is set returns the range the instrument chose as appropriate for that source level.

This attribute is primarily intended to eliminate the time required by the automatic range selection performed by a sourcing instrument. Because selecting a fixed range prevents autoranging, an overrange condition can occur. For example, sourcing 3.0 V on the Model 2651A 1 V range causes an overrange condition.

Example

```
smua.source.rangev = 1
```

Selects the 1 V source range for SMU channel A.

Also see

Range (on page 2-71)
reset() (on page 7-197)
setup.recall() (on page 7-221)
smuX.measure.rangeY (on page 7-258)
smuX.reset() (on page 7-264)
smuX.source.autorangeY (on page 7-267)

smuX.source.settling

This attribute contains the source settling mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.SETTLE_SMOOTH)

Usage

```
settleOption = smuX.source.settling
smuX.source.settling = settleOption
```

settleOption	Set to the source settling mode. Set to one of the following values: 0 or smuX.SETTLE_SMOOTH: Turns off additional settling operations (default) 1 or smuX.SETTLE_FAST_RANGE: Instructs the source-measure unit (SMU) to use a faster procedure when changing ranges 2 or smuX.SETTLE_FAST_POLARITY: Instructs the SMU to change polarity without going to zero 3 or smuX.SETTLE_DIRECT_IRANGE: Instructs the SMU to change the current range directly 128 or smuX.SETTLE_FAST_ALL: Enables all smuX.SETTLE_FAST_* operations
X	SMU channel (for example, smua.source.settling applies to SMU channel A)

Details

Using smuX.SETTLE_FAST_RANGE may cause the SMU to exceed the range change overshoot specification.

smuX.SETTLE_FAST_POLARITY does not go to zero when changing polarity and may create inconsistencies at the zero crossing.

smuX.SETTLE_DIRECT_IRANGE switches the SMU directly to the target range instead of the default "range-by-range" method. This option is mutually exclusive of any other smuX.SETTLE_FAST_* commands.

Example

`smua.source.settling = smua.SETTLE_FAST_POLARITY`	Selects fast polarity changing for SMU channel A.

Also see

Range (on page 2-71)

smuX.source.sink

This attribute turns sink mode on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Saved setup	0 (smuX.DISABLE)

Usage

```
sinkMode = smuX.source.sink
smuX.source.sink = sinkMode
```

sinkMode	Sets the sink mode on or off; set to one of the following values: ■ `0` or `smuX.DISABLE`: Turns off sink mode ■ `1` or `smuX.ENABLE`: Turns on sink mode

Details

This attribute enables or disables sink mode. When sink mode is enabled, it reduces the source limit inaccuracy that occurs when operating in quadrants II and IV (quadrants I and III show this source limit inaccuracy).

Example

```
smua.source.sink = smua.ENABLE
```

Enables sink mode.

Also see

Operating boundaries (on page 4-5)

smuX.trigger.arm.count

This attribute sets the arm count in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	1

Usage

```
triggerArmCount = smuX.trigger.arm.count
smuX.trigger.arm.count = triggerArmCount
```

triggerArmCount	The arm count in the trigger model

Details

During a sweep, the SMU iterates through the arm layer of the trigger model this many times. After performing this many iterations, the SMU returns to an idle state.

If this count is set to zero, the SMU stays in the trigger model indefinitely until aborted.

Example

```
smua.trigger.arm.count = 5
```

Sets the SMU to iterate through the arm layer of the trigger model five times and then return to the idle state.

Also see

smuX.trigger.count (on page 7-287)

smuX.trigger.arm.set()

This function sets the arm event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.arm.set()
```

Details

The SMU automatically clears all the event detectors when the `smuX.trigger.initiate()` function is executed. Call this function after the sweep is initiated.

A typical example that uses this function is when you want the SMU to immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur.

This function start actions on the SMU if a missed trigger event is missed.

Example

```
smua.trigger.arm.set()
```

Sets the arm event detector to the detected state.

Also see

smuX.trigger.initiate() (on page 7-294)
smuX.trigger.measure.set() (on page 7-296)
smuX.trigger.source.set() (on page 7-306)

smuX.trigger.arm.stimulus

This attribute selects the event that causes the arm event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.arm.stimulus
smuX.trigger.arm.stimulus = eventID
```

eventID	Event that triggers the arm detector

Details

Set this attribute to the event ID of any trigger event generator to wait for that event.

Set this attribute to zero to bypass waiting for events at the arm event detector (the SMU continues uninterrupted through the remote trigger model). Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*

Event ID	Event description
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[N].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[N].EVENT_ID	Occurs when a delay expires

Example

```
smua.trigger.arm.stimulus = trigger.timer[1].EVENT_ID
```

An event on trigger timer 1 causes the arm event detector to enter the detected state.

Also see

Triggering (on page 3-36)

smuX.trigger.ARMED_EVENT_ID

This constant contains the number of the armed event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = smu*X*.trigger.ARMED_EVENT_ID

eventID	The armed event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to armed events from this SMU.

Example

```
trigger.timer[1].stimulus = smua.trigger.ARMED_EVENT_ID
```

Trigger timer when the SMU passes through the arm layer.

Also see

Triggering (on page 3-36)

smuX.trigger.autoclear

This attribute turns automatic clearing of the event detectors on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smu*X*.DISABLE)

Usage

```
autoClear = smuX.trigger.autoclear
smuX.trigger.autoclear = autoClear
```

autoClear	Auto clear setting; set to one of the following values: ■ 0 or smu*X*.DISABLE: Turns off automatic clearing of the event detectors ■ 1 or smu*X*.ENABLE: Turns on automatic clearing of the event detectors

Details

This attribute enables or disables automatic clearing of the trigger model state machine event detectors when the SMU transitions from the arm layer to the trigger layer.

Only the detected states of the event detectors are cleared.

The overrun statuses of the event detectors are not automatically cleared when the SMU transitions from the arm layer to the trigger layer.

The event detectors are always cleared when a sweep is initiated.

Example

```
smua.trigger.autoclear = smua.ENABLE
```

Automatically clear the event detectors for the trigger mode state.

Also see

Triggering (on page 3-36)

smuX.trigger.count

This attribute sets the trigger count in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	1

Usage

```
triggerCount = smuX.trigger.count
smuX.trigger.count = triggerCount
```

triggerCount	The trigger count is the number of times the source-measure unit (SMU) iterates in the trigger layer for any given sweep

Details

During a sweep, the SMU iterates through the trigger layer of the trigger model the number of times set by this attribute. After performing the iterations, the SMU returns to the arm layer.

If this count is set to zero (0), the SMU stays in the trigger model indefinitely until aborted.

Example

```
reset()
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
smua.trigger.source.listv( {5} )
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
pulse_timer.delay = 0.0006
pulse_timer.stimulus = period_timer.EVENT_ID
pulse_timer.count = 1
period_timer.delay = 0.005
period_timer.count = 9
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
period_timer.passthrough = true
smua.trigger.source.stimulus = period_timer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
smua.trigger.count = 1
smua.trigger.arm.count = 10
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
```

Generate a 10-point pulse train where each pulse has a width of 600 µs and a pulse period of 5 ms.
Alias the trigger timers to use for pulse width and period.
Create a fixed level voltage sweep.
Set the pulse width and trigger the pulse width timer with a period timer.
Set the pulse period to output one pulse per period and the count to generate 10 pulses.
Trigger the pulse period timer when a sweep is initiated.
Configure the timer to output a trigger event when it starts the first delay.
Trigger the SMU source action using pulse period timer.
Trigger the SMU end pulse action using pulse width timer.
Set the trigger model counts.
Configure the SMU to execute a 10-point pulse train.
Start the trigger model.
Wait for the sweep to complete.

Also see

Triggering (on page 3-36)

smuX.trigger.endpulse.action

This attribute enables or disables pulse mode sweeps.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	1 (smu*X*.SOURCE_HOLD)

Usage

```
pulseAction = smuX.trigger.endpulse.action
smuX.trigger.endpulse.action = pulseAction
```

pulseAction	The pulse mode setting; set to one of the following values (see **Details** for definition): ■ 0 or smu*X*.SOURCE_IDLE ■ 1 or smu*X*.SOURCE_HOLD

Details

When set to smu*X*.SOURCE_HOLD, this attribute disables pulse mode sweeps, holding the source level for the remainder of the step.

When set to smu*X*.SOURCE_IDLE, this attribute enables pulse mode sweeps, setting the source level to the programmed (idle) level at the end of the pulse.

Example

```
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
```

Configure the end pulse action to achieve a pulse and configure trigger timer 1 to control the end of pulse.

Also see

Triggering (on page 3-36)

smuX.trigger.endpulse.set()

This function sets the end pulse event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.endpulse.set()
```

Details

This function sets the end pulse event detector to the detected state.

The SMU automatically clears all the event detectors when the `smuX.trigger.initiate()` function is executed. Therefore, call `smuX.trigger.endpulse.set()` after the sweep is initiated. If the event detectors are configured to clear automatically because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`, make sure that `smuX.trigger.endpulse.set()` is issued after the SMU has entered the trigger layer.

Example

```
reset()
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
smua.trigger.source.listv( {5} )
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
pulse_timer.delay = 0.0006
pulse_timer.stimulus = period_timer.EVENT_ID
pulse_timer.count = 1
period_timer.delay = 0.005
period_timer.count = 9
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
period_timer.passthrough = true
smua.trigger.source.stimulus = period_timer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
smua.trigger.count = 1
smua.trigger.arm.count = 10
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
```

Generate a 10-point pulse train where each pulse has a width of 600 µs and a pulse period of 5 ms.
Alias the trigger timers to use for pulse width and period.
Create a fixed level voltage sweep.
Set the pulse width and trigger the pulse width timer with a period timer.
Set the pulse period to output one pulse per period and the count to generate 10 pulses.
Trigger the pulse period timer when a sweep is initiated.
Configure the timer to output a trigger event when it starts the first delay.
Trigger the SMU source action using pulse period timer.
Trigger the SMU end pulse action using pulse width timer.
Set the trigger model counts.
Configure the SMU to execute a 10-point pulse train.
Start the trigger model.
Wait for the sweep to complete.

Also see

smuX.trigger.autoclear (on page 7-287)
smuX.trigger.initiate() (on page 7-294)
Triggering (on page 3-36)

smuX.trigger.endpulse.stimulus

This attribute defines which event causes the end pulse event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.endpulse.stimulus
smuX.trigger.endpulse.stimulus = eventID
```

eventID	Set to the event that triggers the end pulse action of the source

Details

Set this attribute to the event ID of any trigger event generator to wait for that event. To bypass waiting for an event, set this value of this attribute to `0`. Set *eventID* to one of the existing trigger event IDs, which are shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[N].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[N].EVENT_ID	Occurs when a delay expires

Example

```
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = trigger.timer[1].EVENT_ID
```

Configure the end pulse action to achieve a pulse and select the event, `trigger.timer[1].EVENT_ID`, that causes the arm event detector to enter the detected state.

Also see

Triggering (on page 3-36)

smuX.trigger.endsweep.action

This attribute sets the action of the source at the end of a sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smu*X*.SOURCE_IDLE)

Usage

```
action = smuX.trigger.endsweep.action
smuX.trigger.endsweep.action = action
```

action	The source action at the end of a sweep; set to one of the following values: ■ `0` or `smuX.SOURCE_IDLE`: Sets the source level to the programmed (idle) level at the end of the sweep ■ `1` or `smuX.SOURCE_HOLD`: Sets the source level to stay at the level of the last step

Details

Use this attribute to configure the source action at the end of the sweep. The SMU can be programmed to return to the idle source level or hold the last value of the sweep.

Example

```
smua.trigger.endsweep.action = smua.SOURCE_IDLE
```

Sets SMU return the source back to the idle source level at the end of a sweep.

Also see

Triggering (on page 3-36)

smuX.trigger.IDLE_EVENT_ID

This constant contains the idle event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.IDLE_EVENT_ID
```

eventID	The idle event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to idle events from this SMU.

Example

```
trigger.timer[1].stimulus = smua.trigger.IDLE_EVENT_ID
```

Trigger timer 1 when the SMU returns to the idle layer.

Also see

Triggering (on page 3-36)

smuX.trigger.initiate()

This function initiates a sweep operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.initiate()
```

Details

This function causes the SMU to clear the four trigger model event detectors and enter its trigger model (moves the SMU from the idle state into the arm layer).

To perform source actions during the sweep, before calling this function, it is necessary to configure and enable one of the following sweep source actions:

- `smuX.trigger.sourceY()`
- `smuX.trigger.sourceY()`
- `smuX.trigger.sourceY()`

To make measurements during the sweep, you must also configure and enable the measure action using `smuX.trigger.measure.Y()`.

If you run this function more than once without reconfiguring the sweep measurements, the caches on the configured measurement reading buffers hold stale data. Use the `bufferVar.clearcache()` function to remove stale values from the reading buffer cache.

This function initiates an overlapped operation.

Example

```
smua.trigger.initiate()
```

Starts a preconfigured sweep and clears the event detectors.

Also see

bufferVar.clearcache() (on page 7-22)
Configuring and running sweeps (on page 3-31)
smuX.trigger.measure.action (on page 7-295)
smuX.trigger.measure.Y() (on page 7-298)
smuX.trigger.source.action (on page 7-300)
smuX.trigger.source.linearY() (on page 7-302)
smuX.trigger.source.listY() (on page 7-303)
smuX.trigger.source.logY() (on page 7-304)
Triggering (on page 3-36)

smuX.trigger.measure.action

This attribute controls measurement actions during a sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smuX.DISABLE)

Usage

```
action = smuX.trigger.measure.action
smuX.trigger.measure.action = action
```

action	The sweep measurement action; set to one of the following values: ■ 0 or smuX.DISABLE: Do not make measurements during the sweep ■ 1 or smuX.ENABLE: Make measurements during the sweep ■ 2 or smuX.ASYNC: Make measurements during the sweep, but asynchronously with the source area of the trigger model

Details

With this attribute enabled (setting *action* to smuX.ENABLE or smuX.ASYNC), configure the measurement with one of the smuX.trigger.measure.Y() functions.

If this attribute is set to smuX.ASYNC:

- Asynchronous sweep measurements can only be used with measure autoranging turned off. To turn measure autoranging off for all measurements during the sweep, set the smuX.measure.autorangeY attribute to smuX.AUTORANGE_OFF.
- If the integrating ADC is selected (smuX.measure.adc attribute is set to smuX.ADC_INTEGRATE): Autozero must also be turned off. To turn off autozero, set the smuX.measure.autozero attribute to smuX.AUTOZERO_OFF or smuX.AUTOZERO_ONCE.
- The reading buffer used by smuX.trigger.measure.Y() must have *bufferVar*.collectsourcevalues set to 0.

If any of the above items is incorrectly configured, the smuX.trigger.initiate() function generates an error.

Example

```
smua.trigger.measure.v(smua.nvbuffer1)
smua.trigger.measure.action = smua.ENABLE
```

Configure sweep voltage measurements.
Enable voltage measurements during the sweep.

Also see

bufferVar.collectsourcevalues (on page 7-23)
smuX.measure.adc (on page 7-243)
smuX.trigger.autoclear (on page 7-287)
smuX.trigger.measure.Y() (on page 7-298)
Triggering (on page 3-36)

smuX.trigger.measure.set()

This function sets the measurement event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.measure.set()
```

Details

This function is useful whenever you want the SMU to continue operation without waiting for a programmed trigger event. When called, this function immediately satisfies the event detector, allowing the SMU to continue through the trigger model.

For example, you might use this function to have the SMU immediately perform an action the first time through the trigger model, even if a programmed trigger event does not occur.

If the event detectors are configured to clear automatically because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`, make sure that `smuX.trigger.measure.set()` is issued after the SMU has entered the trigger layer. This function can also be used to start actions on the SMU in case of a missed trigger event.

The SMU automatically clears all event detectors when the `smuX.trigger.initiate()` function is executed. Call this function after the sweep is initiated.

Example

`smua.trigger.measure.set()`	Sets the measure event detector.

Also see

smuX.trigger.arm.set() (on page 7-284)
smuX.trigger.autoclear (on page 7-287)
smuX.trigger.endpulse.set() (on page 7-289)
smuX.trigger.source.set() (on page 7-306)

smua.trigger.measure.stimulus

This attribute selects the event that causes the measure event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.measure.stimulus
smuX.trigger.measure.stimulus = eventID
```

eventID	Event that triggers the measurement detector

Details

Set this attribute to the event ID of any trigger event generator to wait for that event. When set, the SMU waits for the event at the measurement event detector portion of the trigger model.

Set this attribute to zero to bypass waiting for an event (the SMU continues uninterrupted through the remote trigger model). Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smu*X*.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[*N*].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[*N*].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[*N*].EVENT_ID	Occurs when a delay expires

Example

```
smua.trigger.measure.stimulus = trigger.timer[1].EVENT_ID
```

Sets delay before measurement begins.

Also see

Triggering (on page 3-36)

smuX.trigger.measure.Y()

This function configures the measurements that are to be made in a subsequent sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.measure.Y(rbuffer)
smuX.trigger.measure.iv(ibuffer, vbuffer)
```

Y	SMU measurement type (`v` = voltage, `i` = current, `r` = resistance, `p` = power)
rbuffer	A reading buffer object where the readings are stored
ibuffer	A reading buffer object where current readings are stored
vbuffer	A reading buffer object where voltage readings are stored

Details

As measurements are made, they are stored in a reading buffer. If the instrument is configured to return multiple readings where one is requested, the readings are available as they are made. Measurements are in the following units of measure: `v` = volts, `i` = amperes, `r` = ohms, `p` = watts.

The `smuX.trigger.measure.iv()` function stores current readings in *ibuffer* and voltage readings in *vbuffer*.

If a given reading buffer contains any data, it is cleared before making any measurements, unless the reading buffer has been configured to append data.

The SMU only retains the last call to any one of these functions and only that measurement action is performed.

After configuring the measurements to make with this function, enable the measurement action.

Example

```
smua.trigger.measure.v(vbuffername)
smua.trigger.measure.action = smua.ENABLE
```

Stores voltage readings during the sweep in buffer `vbuffername`.

Also see

Reading buffers (on page 3-6)
smuX.measure.Y() (on page 7-261)
smuX.nvbufferY (on page 7-263)
smuX.trigger.measure.action (on page 7-295)
Sweep Operation (on page 3-21)
Triggering (on page 3-36)
waitcomplete() (on page 7-459)

smuX.trigger.MEASURE_COMPLETE_EVENT_ID

This constant contains the measurement complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID

eventID	The measurement complete event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to measure complete events from this SMU.

Example

```
trigger.timer[1].stimulus = smua.trigger.MEASURE_COMPLETE_EVENT_ID
```

Trigger the timer when the SMU completes a measurement.

Also see

Triggering (on page 3-36)

smuX.trigger.PULSE_COMPLETE_EVENT_ID

This constant contains the pulse complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = smu*X*.trigger.PULSE_COMPLETE_EVENT_ID

eventID	The pulse complete event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to pulse complete events.

Example

```
trigger.timer[1].stimulus = smua.trigger.PULSE_COMPLETE_EVENT_ID
```

Trigger a timer when the SMU completes a pulse.

Also see

Triggering (on page 3-36)

smuX.trigger.source.action

This attribute enables or disables sweeping the source (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smu*X*.DISABLE)

Usage

```
action = smuX.trigger.source.action
smuX.trigger.source.action = action
```

action	Sweep source action. Set to one of the following values: ■ 0 or smu*X*.DISABLE: Do not sweep the source ■ 1 or smu*X*.ENABLE: Sweep the source

Details

This attribute is used to enable or disable source level changes during a sweep. In addition to enabling the action before initiating the sweep, make sure to configure it using smu*X*.trigger.source.linear*Y*(), smu*X*.trigger.source.list*Y*(), or smu*X*.trigger.source.log*Y*().

Example

```
smua.trigger.source.listv({3, 1, 4, 5, 2})
smua.trigger.source.action = smua.ENABLE
```

Configure list sweep (sweep through 3 V, 1 V, 4 V, 5 V, and 2 V).
Enable the source action.

Also see

smuX.trigger.source.linearY() (on page 7-302)
smuX.trigger.source.listY() (on page 7-303)
smuX.trigger.source.logY() (on page 7-304)
Triggering (on page 3-36)

smuX.trigger.source.limitY

This attribute sets the sweep source limit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0 (smu*X*.LIMIT_AUTO)

Usage

```
sweepSourceLimit = smuX.trigger.source.limitY
smuX.trigger.source.limitY = sweepSourceLimit
```

sweepSourceLimit	The source limit that is used during triggered operation; set to: ■ A user-defined value ■ `smuX.LIMIT_AUTO` ■ `smuX.LIMIT_OFF` (for `smuX.trigger.source.limiti`)
Y	SMU output function (`v` = voltage, `i` = current)

Details

Use this attribute to perform extended operating area pulse mode sweeps.

If this attribute is set to `smuX.LIMIT_AUTO` (or `0`), the SMU uses the normal limit setting during sweeping. If this attribute is set to any other numeric value, the SMU switches in this limit at the start of the source action and returns to the normal limit setting at the end of the end pulse action.

Normally, the limit range is automatically adjusted in accordance with the limit value. During sweeping, however, the limit range is fixed to avoid the delays associated with changing range. This fixed limit range is determined by the maximum limit value needed during the sweep; that is, the greater of either the normal limit value (as specified by `smuX.source.limitY`) or the sweep limit value (as specified by `smuX.trigger.source.limitY`). The minimum limit value that can be enforced during the sweep is equal to 10% of the full-scale value of the fixed limit range. If the smaller limit value (normal or sweep) falls below this 10% threshold, the 10% value is enforced instead. Likewise, if the limit value falls below the 10% threshold as a result of power compliance, the 10% value is enforced instead.

To disable the source current limit during triggered operation, set the `smuX.trigger.source.limiti` attribute equal to `smuX.LIMIT_OFF` (or `"off"`).

When using the extended operating area, the SMU automatically starts the end pulse action if the SMU is not triggered before its maximum pulse width. It also delays the source action if necessary to limit the pulse duty cycle to stay within the capabilities of the SMU.

Example

`smua.trigger.source.limitv = 10`	Sets the voltage sweep limit of SMU channel A to 10 V.

Also see

Configuring and running sweeps (on page 3-31)
smuX.source.limitY (on page 7-273)
Triggering (on page 3-36)

smuX.trigger.source.linearY()

This function configures a linear source sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.source.linearY(startValue, endValue, points)
```

Y	SMU source function (`v` = voltage, `i` = current)
startValue	Source value of the first point
endValue	Source value of the last point
points	The number of points used to calculate the step size

Details

This function configures the source action to be a linear source sweep in a subsequent sweep. During the sweep, the source generates a uniform series of ascending or descending voltage or current changes called steps. The number of source steps is one less than the number of sourced *points*.

The *points* parameter does not set the number of steps in a sweep. Instead, it is used to calculate source values within a subsequent sweep. If the subsequent sweep has more points than specified in *points*, the source restarts at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU satisfies the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *points*, *endValue* is not reached during the sweep. This means that if the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

In cases where the first sweep point is a nonzero value, it may be necessary to pre-charge the circuit so that the sweep returns a stable value for the first measured point without penalizing remaining points in the sweep.

With linear sweeps, it is acceptable to maintain a fixed source resolution over the entire sweep. To prevent source range changes during the sweep (especially when sweeping through 0.0), set the source range to a fixed range appropriate for the larger of either *startValue* or *endValue*.

The SMU only stores the most recent configured source action. The last call to `smuX.trigger.source.linearY()`, `smuX.trigger.source.listY()`, or `smuX.trigger.source.logY()` is used for the source action.

Source functions cannot be changed within a sweep.

After configuring the sweep source values, enable the source action by setting `smuX.trigger.source.action`.

Example

`smua.trigger.source.linearv(0, 10, 11)`	Sweeps SMU channel A from 0 V to 10 V in 1 V steps.

Also see

smuX.trigger.source.action (on page 7-300)
smuX.trigger.source.listY() (on page 7-303)
smuX.trigger.source.logY() (on page 7-304)
Sweep Operation (on page 3-21)

smuX.trigger.source.listY()

This function configures an array-based source sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smu`*`X`*`.trigger.source.list`*`Y`*`(`*`sweepList`*`)`

`Y`	SMU source function (`v` = voltage, `i` = current)
`sweepList`	An array of source values

Details

This function configures the source action to be a list sweep in a subsequent sweep. During the sweep, the source outputs the sequence of source values given in the *`sweepList`* array.

If the subsequent sweep has more points than specified in *`sweepList`*, the source restarts at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU satisfies the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *`sweepList`*, the extra values are ignored. This means that if the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

In cases where the first sweep point is a nonzero value, it may be necessary to pre-charge the circuit so that the sweep returns a stable value for the first measured point without penalizing remaining points in the sweep.

The SMU only stores the most recent configured source action. The last call to `smu`*`X`*`.trigger.source.linear`*`Y`*`()`, `smu`*`X`*`.trigger.source.list`*`Y`*`()`, or `smu`*`X`*`.trigger.source.log`*`Y`*`()` is used for the source action.

Source functions cannot be changed within a sweep.

After configuring the sweep source values, enable the source action by setting `smu`*`X`*`.trigger.source.action`.

Example

`smua.trigger.source.listv({3, 1, 4, 5, 2})`	Sweeps through 3 V, 1 V, 4 V, 5 V, and 2 V.

Also see

smuX.trigger.source.action (on page 7-300)
smuX.trigger.source.linearY() (on page 7-302)
smuX.trigger.source.logY() (on page 7-304)
Sweep operation (on page 3-21)

smuX.trigger.source.logY()

This function configures an exponential (geometric) source sweep.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`smuX.trigger.source.logY(startValue, endValue, points, asymptote)`

Y	SMU source function (`v` = voltage, `i` = current)
startValue	Source value of the first point
endValue	Source value of the last point
points	The number of points used to calculate the step size
asymptote	The asymptotic offset value

Details

This function configures the source action to be a geometric source sweep in a subsequent sweep. During the sweep, the source generates a geometric series of ascending or descending voltage or current changes called steps. Each step is larger or smaller than the previous step by a fixed proportion. The constant of proportionality is determined by the starting value, the ending value, the asymptote, and the number of steps in the sweep. The number of source steps is one less than the number of sourced *points*.

The *points* parameter does not set the number of steps in a sweep, but rather is used to calculate source values within a subsequent sweep. If the subsequent sweep has more points than specified in *points*, the source restarts at the beginning. This means that if the trigger count is greater than the number of points in a sweep as configured, the SMU satisfies the trigger count by restarting the sweep values from the beginning.

If the subsequent sweep has fewer points than specified in *points*, *endValue* is not reached during the sweep. This means that if the trigger count is less than the number of source values configured, the SMU satisfies the trigger count and ignores the remaining source values.

In cases where the first sweep point is nonzero, it may be necessary to pre-charge the circuit so that the sweep returns a stable value for the first measured point without penalizing remaining points in the sweep.

With logarithmic sweeps, it is usually necessary to allow the source to autorange to maintain good source accuracy when sweeping over more than one decade or across range boundaries.

The *asymptote* parameter customizes the inflection and offset of the source value curve. This allows log sweeps to cross zero. Setting this parameter to zero provides a conventional logarithmic sweep. The *asymptote* value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep.

The *asymptote* value must not be equal to or between the starting and ending values. It must be outside the range defined by the starting and ending values.

The SMU stores only the most recent configured source action. The last call to `smuX.trigger.source.linearY()`, `smuX.trigger.source.listY()`, or `smuX.trigger.source.logY()` is used for the source action.

Source functions cannot be changed within a sweep.

After configuring the sweep source values, enable the source action by setting `smuX.trigger.source.action`.

Example

`smua.trigger.source.logv(1, 10, 11, 0)`	Sweeps from 1 V to 10 V in 10 steps with an asymptote of 0 V.

Also see

smuX.trigger.source.action (on page 7-300)
smuX.trigger.source.linearY() (on page 7-302)
smuX.trigger.source.listY() (on page 7-303)
Sweep operation (on page 3-21)

smuX.trigger.source.set()

This function sets the source event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
smuX.trigger.source.set()
```

Details

This function sets the source event detector to the detected state.

The SMU automatically clears all event detectors when the `smuX.trigger.initiate()` function is executed. Call this function after the sweep is initiated. If the event detectors are configured to clear automatically because the `smuX.trigger.autoclear` attribute is set to `smuX.ENABLE`, make sure that `smuX.trigger.source.set()` is issued after the SMU has entered the trigger layer.

Example

```
reset()
smua.trigger.source.listv({5})
smua.trigger.source.stimulus = display.trigger.EVENT_ID
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
delay(1)
-- Continue even if the display trigger key was not pressed.
smua.trigger.source.set()
waitcomplete()
```

Sets the source event detector.

Also see

smuX.trigger.arm.set() (on page 7-284)
smuX.trigger.autoclear (on page 7-287)
smuX.trigger.endpulse.set() (on page 7-289)
smuX.trigger.initiate() (on page 7-294)
smuX.trigger.measure.set() (on page 7-296)
Triggering (on page 3-36)

smuX.trigger.source.stimulus

This attribute defines which event causes the source event detector to enter the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	SMU reset Instrument reset Recall setup	Not saved	0

Usage

```
eventID = smuX.trigger.source.stimulus
smuX.trigger.source.stimulus = eventID
```

eventID	Set to the event that triggers the end-pulse source off action

Details

Set this attribute to the event ID of any trigger event generator to wait for that event. When set, the SMU waits for the event at the source event detector portion of the trigger model. To bypass waiting for an event, set the value of this attribute to zero (`0`). Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smuX.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smuX.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smuX.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smuX.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smuX.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smuX.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smuX.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[N].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[N].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[N].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a `*TRG` command is received on the remote interface GPIB only: Occurs when a `GET` bus command is received VXI-11 only: Occurs with the VXI-11 command `device_trigger`; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[N].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[N].EVENT_ID	Occurs when a delay expires

Example

```
smua.trigger.source.stimulus = digio.trigger[2].EVENT_ID
```

Configure to start its source action when a trigger event occurs on digital I/O line 2.

Also see

Triggering (on page 3-36)

smuX.trigger.SOURCE_COMPLETE_EVENT_ID

This constant contains the source complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID

eventID	The source action complete event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to source complete events from this source-measure unit (SMU).

Example

```
trigger.timer[1].stimulus = smua.trigger.SOURCE_COMPLETE_EVENT_ID
```

Trigger the timer when the SMU updates the source level or starts a pulse.

Also see

Triggering (on page 3-36)

smuX.trigger.SWEEP_COMPLETE_EVENT_ID

This constant contains the sweep complete event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smuX.trigger.SWEEP_COMPLETE_EVENT_ID
```

eventID	The sweep complete event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to sweep complete events from this SMU.

Example

```
digio.trigger[2].mode = digio.TRIG_RISINGA
digio.trigger[2].clear()
smua.trigger.source.stimulus = digio.trigger[2].EVENT_ID
digio.trigger[4].mode = digio.TRIG_RISINGM
digio.trigger[4].pulsewidth = 0.001
digio.trigger[4].stimulus = smua.trigger.SWEEP_COMPLETE_EVENT_ID
```

Configure the Model 2651A to detect a rising edge on digital I/O line 2.
Configure SMU A to start its source action when a trigger event occurs on digital I/O line 2.
Configure digital I/O line 4 to output a 1 ms rising-edge trigger pulse at the completion of the SMU sweep.

Also see

Triggering (on page 3-36)

smua.trigger.SWEEPING_EVENT_ID

This constant contains the sweeping event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = smua.trigger.SWEEPING_EVENT_ID
```

eventID	The sweeping event number

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to sweeping events from this SMU.

Example

```
reset()
period_timer = trigger.timer[1]
pulse_timer = trigger.timer[2]
smua.trigger.source.listv( {5} )
smua.trigger.source.action = smua.ENABLE
smua.source.rangev = 5
smua.trigger.measure.action = smua.DISABLE
pulse_timer.delay = 0.0006
pulse_timer.stimulus = period_timer.EVENT_ID
pulse_timer.count = 1
period_timer.delay = 0.005
period_timer.count = 9
period_timer.stimulus = smua.trigger.SWEEPING_EVENT_ID
period_timer.passthrough = true
smua.trigger.source.stimulus = period_timer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.endpulse.stimulus = pulse_timer.EVENT_ID
smua.trigger.count = 1
smua.trigger.arm.count = 10
smua.source.output = smua.OUTPUT_ON
smua.trigger.initiate()
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
```

Generate a 10-point pulse train where each pulse has a width of 600 µs and a pulse period of 5 ms.
Alias the trigger timers to use for pulse width and period.
Create a fixed level voltage sweep.
Set the pulse width and trigger the pulse width timer with a period timer.
Set the pulse period to output one pulse per period and the count to generate 10 pulses.
Trigger the pulse period timer when a sweep is initiated.
Configure the timer to output a trigger event when it starts the first delay.
Trigger the SMU source action using pulse period timer.
Trigger the SMU end pulse action using pulse width timer.
Set the trigger model counts.
Configure the SMU to execute a 10-point pulse train.
Start the trigger model.
Wait for the sweep to complete.

Also see

Triggering (on page 3-36)

status.condition

This attribute stores the status byte condition register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
statusByte = status.condition
```

statusByte	The status byte; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the status byte, which is returned as a numeric value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred. When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence.

The individual bits of this register have the meanings described in the following table.

Bit	Value and description

B0	`status.MEASUREMENT_SUMMARY_BIT`
	`status.MSB`
	Set summary bit indicates that an enabled measurement event has occurred.
	Bit B0 decimal value: 1

B1	`status.SYSTEM_SUMMARY_BIT`
	`status.SSB`
	Set summary bit indicates that an enabled system event has occurred.
	Bit B1 decimal value: 2

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the error queue.
	Bit B2 decimal value: 4

B3	`status.QUESTIONABLE_SUMMARY_BIT`
	`status.QSB`
	Set summary bit indicates that an enabled questionable event has occurred.
	Bit B3 decimal value: 8

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Request Service (RQS)/Master Summary Status (MSS). Depending on how it is used, bit B6 of the status byte register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit:
	■ When using the GPIB or VXI-11 serial poll sequence of the Model 2651A to obtain the status byte (serial poll byte), B6 is the RQS bit. The set bit indicates that the Request Service (RQS) bit of the status byte (serial poll byte) is set and a serial poll (SRQ) has occurred.
	■ When using the `status.condition` register command or the *STB? common command to read the status byte, B6 is the MSS bit. Set bit indicates that an enabled summary bit of the status byte register is set.
	Bit B6 decimal value: 64

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

In addition to the above constants, when more than one bit of the register is set, *`statusByte`* equals the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

```
statusByte = status.condition
print(statusByte)
```

Returns `statusByte`.
Sample output:
`1.29000e+02`
Converting this output (129) to its binary equivalent yields 1000 0001
Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSS) and B7 (OSB).

Also see

Status model overview (on page 15-1)
Status byte and service request (SRQ) (on page 15-15)

status.measurement.*

These attributes contain the measurement event register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	10,627 (All bits set)

Usage

```
measurementRegister = status.measurement.condition
measurementRegister = status.measurement.enable
measurementRegister = status.measurement.event
measurementRegister = status.measurement.ntr
measurementRegister = status.measurement.ptr
status.measurement.enable = measurementRegister
status.measurement.ntr = measurementRegister
status.measurement.ptr = measurementRegister
```

measurementRegister	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes read or write the measurement event registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, assume value 257 is returned for the enable register. The binary equivalent is 0000 0001 0000 0001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.measurement.VOLTAGE_LIMIT` `status.measurement.VLMT`	Set bit is a summary of the `status.measurement.voltage_limit` register. Bit B0 decimal value: 1
B1	`status.measurement.CURRENT_LIMIT` `status.measurement.ILMT`	Set bit is a summary of the `status.measurement.current_limit` register. Bit B1 decimal value: 2
B2-B6	Not used	Not applicable
B7	`status.measurement.READING_OVERFLOW` `status.measurement.ROF`	Set bit is a summary of the `status.measurement.reading_overflow` register. Bit B7 decimal value: 128
B8	`status.measurement.BUFFER_AVAILABLE` `status.measurement.BAV`	Set bit is a summary of the `status.measurement.buffer_available` register. Bit B8 decimal value: 256
B9-B10	Not used	Not applicable
B11	`status.measurement.OUTPUT_ENABLE` `status.measurement.OE`	Set bit indicates that output enable has been asserted. Bit B11 decimal value: 2,048
B12	Not used	Not applicable
B13	`status.measurement.INSTRUMENT_SUMMARY` `status.measurement.INST`	Set bit indicates that a bit in the measurement instrument summary register is set. Bit B13 decimal value: 8,192
B14-B15	Not used	Not applicable

As an example, to set bit B8 of the measurement event enable register, set `status.measurement.enable = status.measurement.BAV`. For example, assume value 257 is returned for the enable register. The binary equivalent is 0000 0001 0000 0001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B8, set *measurementRegister* to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
status.measurement.enable = status.measurement.BAV
```

Sets the BAV bit of the measurement event enable register.

Also see

Measurement summary bit (Measurement event register)

status.measurement.buffer_available.*

This attribute contains the measurement event buffer available summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
measurementRegister = status.measurement.buffer_available.condition
measurementRegister = status.measurement.buffer_available.enable
measurementRegister = status.measurement.buffer_available.event
measurementRegister = status.measurement.buffer_available.ntr
measurementRegister = status.measurement.buffer_available.ptr
status.measurement.buffer_available.enable = measurementRegister
status.measurement.buffer_available.ntr = measurementRegister
status.measurement.buffer_available.ptr = measurementRegister
```

measurementRegister	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the measurement event buffer available summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 (SMUA) is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.buffer_available.SMUA`	Set bit indicates that there is at least one reading stored in either or both of the dedicated reading buffers. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event buffer available summary enable register, set `status.measurement.buffer_available.enable = status.measurement.buffer_available.SMUA`.

In addition to the above constant, *`measurementRegister`* can be set to the decimal equivalent of the bit to set.

Example

`status.measurement.buffer_available.enable =` `  status.measurement.buffer_available.SMUA`	Sets the SMUA bit of the measurement event buffer available summary enable register.

Also see

Measurement event registers (on page 15-7)

status.measurement.current_limit.*

This attribute contains the measurement event current limit summary registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
measurementRegister = status.measurement.current_limit.condition
measurementRegister = status.measurement.current_limit.enable
measurementRegister = status.measurement.current_limit.event
measurementRegister = status.measurement.current_limit.ntr
measurementRegister = status.measurement.current_limit.ptr
status.measurement.current_limit.enable = measurementRegister
status.measurement.current_limit.ntr = measurementRegister
status.measurement.current_limit.ptr = measurementRegister
```

measurementRegister	The status of the measurement event current limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the measurement event current limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 (SMUA) is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.current_limit.SMUA`	Set bit indicates that the current limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event current limit summary enable register, set `status.measurement.current_limit.enable = status.measurement.current_limit.SMUA`.

In addition to the above constant, *measurementRegister* can be set to the decimal equivalent of the bit to set.

Example

`status.measurement.current_limit.enable =` `  status.measurement.current_limit.SMUA`	Sets the SMUA bit of the Measurement Event Current Limit Summary Enable Register.

Also see

Measurement event registers (on page 15-7)
status.measurement.instrument.smuX.* (on page 7-319)

status.measurement.instrument.*

This attribute contains the registers of the measurement event instrument summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
measurementRegister = status.measurement.instrument.condition
measurementRegister = status.measurement.instrument.enable
measurementRegister = status.measurement.instrument.event
measurementRegister = status.measurement.instrument.ntr
measurementRegister = status.measurement.instrument.ptr
status.measurement.instrument.enable = measurementRegister
status.measurement.instrument.ntr = measurementRegister
status.measurement.instrument.ptr = measurementRegister
```

measurementRegister	The status of the measurement event instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the measurement event instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 (SMUA) is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.instrument.SMUA`	Set bit indicates one or more enabled bits of the measurement event SMU A summary register is set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event instrument summary enable register, set `status.measurement.instrument.enable = status.measurement.instrument.SMUA`.

In addition to the above constant, *`measurementRegister`* can be set to the decimal equivalent of the bit to set.

Example

`status.measurement.instrument.enable =` `  status.measurement.instrument.SMUA`	Sets the SMU A bit of the measurement event instrument summary enable register using a constant.

Also see

Measurement event registers (on page 15-7)

status.measurement.instrument.smuX.*

This attribute contains the registers of the measurement event SMU X summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	387 (All bits set)

Usage

```
measurementRegister = status.measurement.instrument.smuX.condition
measurementRegister = status.measurement.instrument.smuX.enable
measurementRegister = status.measurement.instrument.smuX.event
measurementRegister = status.measurement.instrument.smuX.ntr
measurementRegister = status.measurement.instrument.smuX.ptr
status.measurement.instrument.smuX.enable = measurementRegister
status.measurement.instrument.smuX.ntr = measurementRegister
status.measurement.instrument.smuX.ptr = measurementRegister
```

`measurementRegister`	The status of the instrument measurement status SMU X summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
`X`	Source-measure unit (SMU) channel (for example `status.measurement.instrument.smua.enable` applies to SMU channel A)

Details

These attributes are used to read or write to the measurement event SMU X summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 257 is returned for the enable register. The binary equivalent is 0000 0001 0000 0001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15 **	B14 >	B13 >	B12 >	B11 >	B10 >	B9 >	B8 >	B7 >	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0*	`status.measurement.instrument.smuX.VOLTAGE_LIMIT` `status.measurement.instrument.smuX.VLMT`	Set bit indicates that the voltage limit was exceeded. Bit B0 decimal value: 1
B1*	`status.measurement.instrument.smuX.CURRENT_LIMIT` `status.measurement.instrument.smuX.ILMT`	Set bit indicates that the current limit was exceeded. Bit B1 decimal value: 2
B2-B6	Not used	Not applicable.
B7	`status.measurement.instrument.smuX.READING_OVERFLOW` `status.measurement.instrument.smuX.ROF`	Set bit indicates that an overflow reading has been detected. Bit B7 decimal value: 128
B8	`status.measurement.instrument.smuX.BUFFER_AVAILABLE` `status.measurement.instrument.smuX.BAV`	Set bit indicates that there is at least one reading stored in either or both of the dedicated reading buffers. Bit B8 decimal value: 256
B9-B15	Not used	Not applicable.

* This bit will be updated only when a measurement is taken or `smuX.source.compliance` is invoked.

As an example, to set bit B0 of the measurement event SMU X summary enable register, set `status.measurement.instrument.smua.enable = status.measurement.instrument.smua.VLMT`.

In addition to the above constants, *`measurementRegister`* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *`measurementRegister`* to the sum of their decimal weights. For example, to set bits B1 and B8, set *`measurementRegister`* to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

`status.measurement.instrument.smua.enable = status.measurement.instrument.smua.VLMT`	Sets the VLMT bit of the measurement event SMU A summary enable register using a constant.

Also see

Measurement event registers (on page 15-7)

status.measurement.reading_overflow.*

This attribute contains the measurement event reading overflow summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
measurementRegister = status.measurement.reading_overflow.condition
measurementRegister = status.measurement.reading_overflow.enable
measurementRegister = status.measurement.reading_overflow.event
measurementRegister = status.measurement.reading_overflow.ntr
measurementRegister = status.measurement.reading_overflow.ptr
status.measurement.reading_overflow.enable = measurementRegister
status.measurement.reading_overflow.ntr = measurementRegister
status.measurement.reading_overflow.ptr = measurementRegister
```

measurementRegister	The status of the measurement reading overflow summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the measurement event reading overflow summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 (SMUA) is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.reading_overflow.SMUA`	Set bit indicates that an overflow reading has been detected. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event reading overflow summary enable register, set `status.measurement.reading_overflow.enable = status.measurement.reading_overflow.SMUA`.

In addition to the above constant, *`measurementRegister`* can be set to the numeric equivalent of the bit to set.

Example

`status.measurement.reading_overflow.enable =` `  status.measurement.reading_overflow.SMUA`	Sets the SMU A bit of the measurement reading overflow summary enable register using a constant.

Also see

Measurement event registers (on page 15-7)

status.measurement.sink_limit.*

This attribute contains the measurement sink limit summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
measurementRegister = status.measurement.sink_limit.condition
measurementRegister = status.measurement.sink_limit.enable
measurementRegister = status.measurement.sink_limit.event
measurementRegister = status.measurement.sink_limit.ntr
measurementRegister = status.measurement.sink_limit.ptr
status.measurement.sink_limit.enable = measurementRegister
status.measurement.sink_limit.ntr = measurementRegister
status.measurement.sink_limit.ptr = measurementRegister
```

measurementRegister	The status of the measurement sink limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the measurement event sink limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 (SMUA) is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.sink_limit.SMUA`	Set bit indicates that the sink limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event sink limit summary enable register, set `status.measurement.sink_limit.enable = status.measurement.sink_limit.SMUA`.

In addition to the above constants, *measurementRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B1, set *measurementRegister* to 2.

Example

`status.measurement.sink_limit.enable = status.measurement.sink_limit.SMUA`	Sets the SMU A bit of the measurement sink limit summary enable register using a constant.

Also see

Measurement event registers (on page 15-7)

status.measurement.voltage_limit.*

This attribute contains the measurement event voltage limit summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
measurementRegister = status.measurement.voltage_limit.condition
measurementRegister = status.measurement.voltage_limit.enable
measurementRegister = status.measurement.voltage_limit.event
measurementRegister = status.measurement.voltage_limit.ntr
measurementRegister = status.measurement.voltage_limit.ptr
status.measurement.voltage_limit.enable = measurementRegister
status.measurement.voltage_limit.ntr = measurementRegister
status.measurement.voltage_limit.ptr = measurementRegister
```

measurementRegister	The status of the measurement voltage limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes read or write to the measurement event voltage limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.measurement.voltage_limit.SMUA`	Set bit indicates that the voltage limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the measurement event voltage limit summary enable register, set `status.measurement.voltage_limit.enable = status.measurement.voltage_limit.SMUA`.

In addition to the above constant, *`measurementRegister`* can be set to the numeric equivalent of the bit to set.

Example

`status.measurement.voltage_limit.enable =` `  status.measurement.voltage_limit.SMUA`	Sets the SMUA bit of the measurement event voltage limit summary enable register using a constant.

Also see

Measurement event registers (on page 15-7)

status.node_enable

This attribute stores the system node enable register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
nodeEnableRegister = status.node_enable
status.node_enable = nodeEnableRegister
```

`nodeEnableRegister`	The status of the system node enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

This attribute is used to read or write to the system node enable register. Reading the system node enable register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, a summary bit is set in the appropriate system summary register. The register and bit that is set depends on the TSP-Link node number assigned to this instrument.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description

B0	`status.MEASUREMENT_SUMMARY_BIT`
	`status.MSB`
	Set summary bit indicates that an enabled measurement event has occurred.
	Bit B0 decimal value: 1

B1	Not used

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the error queue.
	Bit B2 decimal value: 4

B3	`status.QUESTIONABLE_SUMMARY_BIT`
	`status.QSB`
	Set summary bit indicates that an enabled questionable event has occurred.
	Bit B3 decimal value: 8

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set.
	Bit B6 decimal value: 64

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

As an example, to set the B0 bit of the system node enable register, set `status.node_enable = status.MSB`.

In addition to the above values, *`nodeEnableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`nodeEnableRegister`* to the sum of their decimal weights. For example, to set bits B0 and B7, set *`nodeEnableRegister`* to 129 (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example 1

```
nodeEnableRegister = status.MSB + status.OSB
status.node_enable = nodeEnableRegister
```

Use constants to set the MSB and OSB bits of the system node enable register.

Example 2

```
-- decimal 129 = binary 10000001
nodeEnableRegister = 129
status.node_enable = nodeEnableRegister
```

Sets the MSB and OSB bits of the system node enable register using a decimal value.

Also see

status.condition (on page 7-311)
status.system.* (on page 7-383)
Status model overview (on page 15-1)
Status byte and service request (SRQ) (on page 15-15)

status.node_event

This attribute stores the status node event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

```
nodeEventRegister = status.node_event
```

nodeEventRegister	The status of the node event register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the status node event register, which is returned as a numeric value (reading this register returns a value). The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred.

Bit	Value and description

B0	`status.MEASUREMENT_SUMMARY_BIT`
	`status.MSB`
	Set summary bit indicates that an enabled measurement event has occurred.
	`Bit B0 decimal value: 1`

B1	Not used

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the error queue.
	Bit B2 decimal value: 4

B3	`status.QUESTIONABLE_SUMMARY_BIT`
	`status.QSB`
	Set summary bit indicates that an enabled questionable event has occurred.
	Bit B3 decimal value: 8

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte register is set.
	Bit B6 decimal value: 64

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

In addition to the above constants, *`nodeEventRegister`* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *`nodeEventRegister`* contains the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

```
nodeEventRegister = status.node_event
print(nodeEventRegister)
```

Reads the status node event register.
Sample output:
`1.29000e+02`
Converting this output (129) to its binary equivalent yields `1000 0001`. Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSB) and B7 (OSB).

Also see

Status byte and service request (SRQ) (on page 15-15)
Status model overview (on page 15-1)
status.condition (on page 7-311)
status.system.* (on page 7-383)

status.operation.*

These attributes manage the operation status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	31,769 (All bits set)

Usage

```
operationRegister = status.operation.condition
operationRegister = status.operation.enable
operationRegister = status.operation.event
operationRegister = status.operation.ntr
operationRegister = status.operation.ptr
status.operation.enable = operationRegister
status.operation.ntr = operationRegister
status.operation.ptr = operationRegister
```

operationRegister	The status of the operation status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes read or write the operation status registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `2.04800e+04` (which is 20,480) is read as the value of the condition register, the binary equivalent is 0101 0000 0000 0000. This value indicates that bit B14 (`PROGRAM_RUNNING`) and bit B12 (USER) are set.

B15 **	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0 *
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value

B0	`status.operation.CALIBRATING`
	`status.operation.CAL`
	Set bit indicates that the summary bit of the `status.operation.calibrating` register has been set.
	Bit B0 decimal value: 1

B1 to B2	Not used

B3	`status.operation.SWEEPING`
	`status.operation.SWE`
	Set bit indicates that the summary bit from the `status.operation.sweeping` register is set.
	Bit B3 decimal value: 8

B4	`status.operation.MEASURING`
	`status.operation.MEAS`
	Set bit indicates that the summary bit of the `status.operation.measuring` register is set.
	Bit B4 decimal value: 16

B5 to B9	Not used

B10	`status.operation.TRIGGER_OVERRUN`
	`status.operation.TRGOVR`
	Set bit indicates that the summary bit from the `status.operation.trigger_overrun` register is set.
	Bit B10 decimal value: 1,024

B11	`status.operation.REMOTE_SUMMARY`
	`status.operation.REM`
	Set bit indicates that the summary bit of the `status.operation.remote` register is set.
	Bit B11 decimal value: 2,048

B12	`status.operation.USER` Set bit indicates that the summary bit from the `status.operation.user` register is set. Bit B12 decimal value: 4,096

B13	`status.operation.INSTRUMENT_SUMMARY`
	`status.operation.INST`
	Set bit indicates that the summary bit from the `status.operation.instrument` register is set.
	Bit B13 decimal value: 8,192

B14	`status.operation.PROGRAM_RUNNING`
	`status.operation.PROG`
	Set bit indicates that a command or program is running.
	Bit B14 decimal value: 16,384

B15	Not used

As an example, to set bit B12 of the operation status enable register, set `status.operation.enable = status.operation.USER`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B12 and B14, set *`operationRegister`* to 20,480 (which is the sum of 4,096 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.USER + status.operation.PROG
status.operation.enable = operationRegister
```

Uses constants to set the USER and PROG bits of the operation status enable register.

Example 2

```
-- decimal 20480 = binary 0101 0000 0000 0000
operationRegister = 20480
status.operation.enable = operationRegister
```

Uses a decimal value to set the USER and PROG bits of the operation status enable register.

Also see

Operation Status Registers (on page 15-9)
Operation Status Registers (on page 15-9)

status.operation.calibrating.*

This attribute contains the operation status calibration summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
operationRegister = status.operation.calibrating.condition
operationRegister = status.operation.calibrating.enable
operationRegister = status.operation.calibrating.event
operationRegister = status.operation.calibrating.ntr
operationRegister = status.operation.calibrating.ptr
status.operation.calibrating.enable = operationRegister
status.operation.calibrating.ntr = operationRegister
status.operation.calibrating.ptr = operationRegister
```

operationRegister	The status of the operation calibrating event register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the operation status calibration summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.calibrating.SMUA`	Set bit indicates that SMU A is unlocked for calibration. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status calibration summary enable register, set `status.operation.calibrating.enable = status.operation.calibrating.SMUA`.

In addition to the above constant, *operationRegister* can be set to the decimal equivalent of the bit to set.

Example

`status.operation.calibrating.enable = status.operation.calibrating.SMUA`	Sets the SMUA bit of the operation status calibration summary enable register using a constant.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

status.operation.instrument.*

This attribute contains the operation status instrument summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	31,746 (All bits set)

Usage

```
operationRegister = status.operation.instrument.condition
operationRegister = status.operation.instrument.enable
operationRegister = status.operation.instrument.event
operationRegister = status.operation.instrument.ntr
operationRegister = status.operation.instrument.ptr
status.operation.instrument.enable = operationRegister
status.operation.instrument.ntr = operationRegister
status.operation.instrument.ptr = operationRegister
```

operationRegister	The status of the operation event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02600e+03` (which is 1,026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15 **	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0 *
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.instrument.SMUA`	Set bit indicates one or more enabled bits for the operation status SMU A summary register is set. Bit B1 decimal value: 2
B2-B9	Not used	Not applicable.
B10	`status.operation.instrument.TRIGGER_BLENDER` `status.operation.instrument.TRGBLND`	Set bit indicates one or more enabled bits for the operation status trigger blender summary register is set. Bit B10 decimal value: 1,024.
B11	`status.operation.instrument.TRIGGER_TIMER` `status.operation.instrument.TRGTMR`	Set bit indicates one or more enabled bits for the operation status trigger timer summary register is set. Bit B11 decimal value: 2,048
B12	`status.operation.instrument.DIGITAL_IO` `status.operation.instrument.DIGIO`	Set bit indicates one or more enabled bits for the operation status digital I/O summary register is set. Bit B12 decimal value: 4,096
B13	`status.operation.instrument.TSPLINK`	Set bit indicates one or more enabled bits for the operation status TSP-Link summary register is set. Bit B13 decimal value: 8,192
B14	`status.operation.instrument.LAN`	Set bit indicates one or more enabled bits for the operation status LAN summary register is set. Bit B14 decimal value: 16,384
B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status instrument summary enable register, set `status.operation.instrument.enable = status.operation.instrument.SMUA`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B10, set *`operationRegister`* to 1,026 (which is the sum of 2 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.SMUA +
   status.operation.instrument.TRGBLND
status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B10 of the operation status instrument summary enable register using constants.

Example 2

```
-- 1026 = binary 0000 0100 0000 0010
operationRegister = 1026
status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B10 of the operation status instrument summary enable register using a decimal value.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

Condition register sets of:

• status.operation.instrument.digio.* (on page 7-335)
• status.operation.instrument.lan.* (on page 7-339)
• status.operation.instrument.trigger_blender.* (on page 7-347)
• status.operation.instrument.trigger_timer.* (on page 7-351)
• status.operation.instrument.tsplink.* (on page 7-354)

status.operation.instrument.digio.*

This attribute contains the operation status digital I/O summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.digio.condition
operationRegister = status.operation.instrument.digio.enable
operationRegister = status.operation.instrument.digio.event
operationRegister = status.operation.instrument.digio.ntr
operationRegister = status.operation.instrument.digio.ptr
status.operation.instrument.digio.enable = operationRegister
status.operation.instrument.digio.ntr = operationRegister
status.operation.instrument.digio.ptr = operationRegister
```

operationRegister	The status of the operation status digital I/O summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status digital I/O summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.digio.TRIGGER_OVERRUN`
	`status.operation.instrument.digio.TRGOVR`
	Set bit indicates an enabled bit in the Operation Status Digital I/O Overrun Register is set.
	Bit B10 decimal value: 1,024
	Binary value: 0100 0000 0010

B11 to B15	Not used

In addition to the above constant, *`operationRegister`* can be set to the decimal value of the bit to set.

Example 1

```
status.operation.instrument.digio.enable = status.operation.instrument.digio.TRGOVR
```

Uses a constant to set the TRGOVR bit of the operation status digital I/O summary enable register.

Example 2

```
status.operation.instrument.digio.enable = 1024
```

Uses the decimal value to set the TRGOVR bit of the operation status digital I/O summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.digio.trigger_overrun.* (on page 7-337)

status.operation.instrument.digio.trigger_overrun.*

This attribute contains the operation status digital I/O overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,766 (All bits set)

Usage

```
operationRegister = status.operation.instrument.digio.trigger_overrun.condition
operationRegister = status.operation.instrument.digio.trigger_overrun.enable
operationRegister = status.operation.instrument.digio.trigger_overrun.event
operationRegister = status.operation.instrument.digio.trigger_overrun.ntr
operationRegister = status.operation.instrument.digio.trigger_overrun.ptr
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
status.operation.instrument.digio.trigger_overrun.ntr = operationRegister
status.operation.instrument.digio.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status digio I/O overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status digital I/O overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02600e+03` (which is 1026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates that the specified digital I/O line generated an action overrun when it was triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.digio.trigger_overrun.LINE1`	2
B2	`status.operation.instrument.digio.trigger_overrun.LINE2`	4
B3	`status.operation.instrument.digio.trigger_overrun.LINE3`	8
B4	`status.operation.instrument.digio.trigger_overrun.LINE4`	16
B5	`status.operation.instrument.digio.trigger_overrun.LINE5`	32
B6	`status.operation.instrument.digio.trigger_overrun.LINE6`	64
B7	`status.operation.instrument.digio.trigger_overrun.LINE7`	128
B8	`status.operation.instrument.digio.trigger_overrun.LINE8`	256
B9	`status.operation.instrument.digio.trigger_overrun.LINE9`	512
B10	`status.operation.instrument.digio.trigger_overrun.LINE10`	1,024
B11	`status.operation.instrument.digio.trigger_overrun.LINE11`	2,048
B12	`status.operation.instrument.digio.trigger_overrun.LINE12`	4,096
B13	`status.operation.instrument.digio.trigger_overrun.LINE13`	8,192
B14	`status.operation.instrument.digio.trigger_overrun.LINE14`	16,384
B15	Not used	Not applicable

As an example, to set bit B1 of the operation status digital I/O overrun enable register, set `status.operation.instrument.digio.trigger_overrun.enable = status.operation.instrument.digio.trigger_overrun.LINE1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values. For example, to set bits B1 and B10, set *`operationRegister`* to 1,026 (which is the sum of 2 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.digio.trigger_overrun.LINE1 +
   status.operation.instrument.digio.trigger_overrun.LINE10
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
```

Uses constants to set bit B1 and bit B10 of the operation status digital I/O overrun enable register.

Example 2

```
operationRegister = 1026
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
```

Uses the decimal value to set bit B1 and bit B10 of the operation status digital I/O overrun enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.digio.* (on page 7-335)

status.operation.instrument.lan.*

This attribute contains the operation status LAN summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1027 (All bits set)

Usage

```
operationRegister = status.operation.instrument.lan.condition
operationRegister = status.operation.instrument.lan.enable
operationRegister = status.operation.instrument.lan.event
operationRegister = status.operation.instrument.lan.ntr
operationRegister = status.operation.instrument.lan.ptr
status.operation.instrument.lan.enable = operationRegister
status.operation.instrument.lan.ntr = operationRegister
status.operation.instrument.lan.ptr = operationRegister
```

operationRegister	The status of the operation status LAN summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status LAN summary registers. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02600e+03` (which is 1026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description

B0	`status.operation.instrument.lan.CONNECTION`
	`status.operation.instrument.lan.CON`
	Set bit indicates that the LAN cable is connected and a link has been detected.
	Bit B0 decimal value: 1

B1	`status.operation.instrument.lan.CONFIGURING`
	`status.operation.instrument.lan.CONF`
	Set bit indicates the LAN is performing its configuration sequence.
	Bit B1 decimal value: 2

B2 to B9	Not used

B10	`status.operation.instrument.lan.TRIGGER_OVERRUN`
	`status.operation.instrument.lan.TRGOVR`
	Set bit indicates one or more enabled bits for the operation status LAN trigger overrun register is set.
	Bit B10 decimal value: 1,024

B11 to B15	Not used

As an example, to set bit B0 of the operation status LAN summary enable register, set `status.operation.instrument.lan.enable = status.operation.instrument.lan.CON`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B10, set *`operationRegister`* to 1,026 (which is the sum of 2 + 1024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.lan.CONF +
   status.operation.instrument.lan.TRGOVR
status.operation.instrument.lan.enable = operationRegister
```

Use constants to set bit B1 and bit B10 of the operation status LAN summary enable register.

Example 2

```
operationRegister = 1026
status.operation.instrument.lan.enable = operationRegister
```

Use the decimal value to set bit B1 and bit B10 of the operation status LAN summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.lan.trigger_overrun.* (on page 7-341)

status.operation.instrument.lan.trigger_overrun.*

This attribute contains the operation status LAN trigger overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```
operationRegister = status.operation.instrument.lan.trigger_overrun.condition
operationRegister = status.operation.instrument.lan.trigger_overrun.enable
operationRegister = status.operation.instrument.lan.trigger_overrun.event
operationRegister = status.operation.instrument.lan.trigger_overrun.ntr
operationRegister = status.operation.instrument.lan.trigger_overrun.ptr
status.operation.instrument.lan.trigger_overrun.enable = operationRegister
status.operation.instrument.lan.trigger_overrun.ntr = operationRegister
status.operation.instrument.lan.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status LAN trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status LAN trigger overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `2.58000e+02` (which is 258) is read as the value of the condition register, the binary equivalent is 0000 0001 0000 0010. This value indicates that bit B1 and bit B8 are set.

B15 **	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0 *
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates that the specified LAN trigger generated an action overrun when triggered to generate a trigger packet.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.lan.trigger_overrun.LAN1`	2
B2	`status.operation.instrument.lan.trigger_overrun.LAN2`	4
B3	`status.operation.instrument.lan.trigger_overrun.LAN3`	8
B4	`status.operation.instrument.lan.trigger_overrun.LAN4`	16
B5	`status.operation.instrument.lan.trigger_overrun.LAN5`	32
B6	`status.operation.instrument.lan.trigger_overrun.LAN6`	64
B7	`status.operation.instrument.lan.trigger_overrun.LAN7`	128
B8	`status.operation.instrument.lan.trigger_overrun.LAN8`	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status LAN trigger overrun enable register, set `status.operation.instrument.lan.trigger_overrun.enable = status.operation.instrument.lan.trigger_overrun.LAN1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B8, set *`operationRegister`* to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.lan.trigger_overrun.LAN1 +
   status.operation.instrument.lan.trigger_overrun.LAN8
status.operation.instrument.lan.trigger_overrun.enable = operationRegister
```

Use constants to set bit B1 and bit B8 of the operation status LAN trigger overrun enable register.

Example 2

```
operationRegister = 258
status.operation.instrument.lan.trigger_overrun.enable = operationRegister
```

Use the decimal value to set bit B1 and bit B8 of the operation status LAN trigger overrun enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.lan.* (on page 7-339)

status.operation.instrument.smuX.*

This attribute contains the operation status SMU summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1049 (All bits set)

Usage

```
operationRegister = status.operation.instrument.smuX.condition
operationRegister = status.operation.instrument.smuX.enable
operationRegister = status.operation.instrument.smuX.event
operationRegister = status.operation.instrument.smuX.ntr
operationRegister = status.operation.instrument.smuX.ptr
status.operation.instrument.smuX.enable = operationRegister
status.operation.instrument.smuX.ntr = operationRegister
status.operation.instrument.smuX.ptr = operationRegister
```

operationRegister	The status of the operation status SMU summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status SMU summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02500e+02` (which is 1,025) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B0 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description

B0	`status.operation.instrument.smuX.CALIBRATING`
	`status.operation.instrument.smuX.CAL`
	Set bit indicates that `smuX` is unlocked for calibration.
	Bit B0 decimal value: 1

B1 to B2	Not used

B3	`status.operation.instrument.smuX.SWEEPING`
	`status.operation.instrument.smuX.SWE`
	Set bit indicates that `smuX` is sweeping.
	Bit B3 decimal value: 8

B4	`status.operation.instrument.smuX.MEASURING`
	`status.operation.instrument.smuX.MEAS`
	Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement.
	Bit B4 decimal value: 16

B5 to B9	Not used

B10	`status.operation.instrument.smuX.TRIGGER_OVERRUN`
	`status.operation.instrument.smuX.TRGOVR`
	Set bit indicates an enabled bit has been set in the operation status `smuX` trigger overrun event register.
	Bit B10 decimal value: 1,024

B11 to B15	Not used

As an example, to set bit B0 of the operation status SMU A summary enable register, set `status.operation.instrument.smua.enable = status.operation.instrument.smua.CAL`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B0 and B10, set *`operationRegister`* to 1,025 (which is the sum of 1 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
status.operation.instrument.smua.enable = status.operation.instrument.smua.MEAS
```

Use a constant to set bit B4t of the operation status SMU A summary enable register.

Example 2

```
status.operation.instrument.smua.enable = 1025
```

Use the decimal value to set bits B0 and B10 of the operation status SMU A summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.smuX.trigger_overrrun.* (on page 7-345)

status.operation.instrument.smuX.trigger_overrrun.*

This attribute contains the operation status SMU trigger overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	30 (All bits set)

Usage

```
operationRegister = status.operation.instrument.smuX.trigger_overrun.condition
operationRegister = status.operation.instrument.smuX.trigger_overrun.enable
operationRegister = status.operation.instrument.smuX.trigger_overrun.event
operationRegister = status.operation.instrument.smuX.trigger_overrun.ntr
operationRegister = status.operation.instrument.smuX.trigger_overrun.ptr
status.operation.instrument.smuX.trigger_overrun.enable = operationRegister
status.operation.instrument.smuX.trigger_overrun.ntr = operationRegister
status.operation.instrument.smuX.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status SMU trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status SMU trigger overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 18 is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`status.operation.instrument.smu`*X*`.trigger_overrun.ARM` Set bit indicates that the arm event detector of the SMU was already in the detected state when a trigger was received. Bit B1 decimal value: 2
B2	`status.operation.instrument.smu`*X*`.trigger_overrun.SRC` Set bit indicates that the source event detector of the SMU was already in the detected state when a trigger was received. Bit B2 decimal value: 4
B3	`status.operation.instrument.smu`*X*`.trigger_overrun.MEAS` Set bit indicates that the measurement event detector of the SMU was already in the detected state when a trigger was received. Bit B3 decimal value: 8
B4	`status.operation.instrument.smu`*X*`.trigger_overrun.ENDP` Set bit indicates that the end pulse event detector of the SMU was already in the detected state when a trigger was received. Bit B4 decimal value: 16
B5 to B15	Not used

As an example, to set bit B1 of the operation status SMU A trigger overrun enable register, set `status.operation.instrument.smua.trigger_overrun.enable = status.operation.instrument.smua.trigger_overrun.ARM`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B4, set *`operationRegister`* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
status.operation.instrument.smua.trigger_overrun.enable =
   status.operation.instrument.smua.trigger_overrun.ARM
```

Uses a constant to sets the ARM bit of the operation status SMU A trigger overrun enable register.

Example 2

```
status.operation.instrument.smua.trigger_overrun.enable = 18
```

Uses the decimal value to set bits B1 and B4 of the operation status SMU A trigger overrun enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.smuX.* (on page 7-343)

status.operation.instrument.trigger_blender.*

This attribute contains the operation status trigger blender summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.trigger_blender.condition
operationRegister = status.operation.instrument.trigger_blender.enable
operationRegister = status.operation.instrument.trigger_blender.event
operationRegister = status.operation.instrument.trigger_blender.ntr
operationRegister = status.operation.instrument.trigger_blender.ptr
status.operation.instrument.trigger_blender.enable = operationRegister
status.operation.instrument.trigger_blender.ntr = operationRegister
status.operation.instrument.trigger_blender.ptr = operationRegister
```

operationRegister	The status of the operation status trigger blender summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status trigger blender summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.trigger_blender.TRIGGER_OVERRUN`
	`status.operation.instrument.trigger_blender.TRGOVR`
	Set bit indicates one or more enabled bits for operation status trigger blender overrun register is set.
	Bit B10 decimal value: 1,024
	Binary value:
	0100 0000 0000

B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *operationRegister* to 1024.

Example

```
status.operation.instrument.trigger_blender.enable = 1024
```

Uses a decimal value to set the TRGOVR bit of the operation status trigger blender summary enable.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.trigger_blender.trigger_overrun.* (on page 7-349)

status.operation.instrument.trigger_blender.trigger_overrun.*

This attribute contains the operation status trigger blender overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	30 (All bits set)

Usage

```
operationRegister =
   status.operation.instrument.trigger_blender.trigger_overrun.condition
operationRegister =
   status.operation.instrument.trigger_blender.trigger_overrun.enable
operationRegister =
   status.operation.instrument.trigger_blender.trigger_overrun.event
operationRegister =
   status.operation.instrument.trigger_blender.trigger_overrun.ntr
operationRegister =
   status.operation.instrument.trigger_blender.trigger_overrun.ptr
status.operation.instrument.trigger_blender.trigger_overrun.enable =
   operationRegister
status.operation.instrument.trigger_blender.trigger_overrun.ntr =
   operationRegister
status.operation.instrument.trigger_blender.trigger_overrun.ptr =
   operationRegister
```

operationRegister	The status of the operation status trigger blender overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger blender overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `18` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 are set.

B15 **	B14 >	B13 >	B12 >	B11 >	B10 >	B9 >	B8 >	B7 >	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit
** Most significant bit

A set bit value indicates that the specified trigger blender generated an action overrun.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.operation.instrument.trigger_blender.trigger_overrun.BLND1`	Bit B1 decimal value: 2
B2	`status.operation.instrument.trigger_blender.trigger_overrun.BLND2`	Bit B2 decimal value: 4
B3	`status.operation.instrument.trigger_blender.trigger_overrun.BLND3`	Bit B3 decimal value: 8
B4	`status.operation.instrument.trigger_blender.trigger_overrun.BLND4`	Bit B4 decimal value: 16
B5-B15	Not used	Not applicable

As an example, to set bit B1 of the operation status trigger blender overrun enable register, set `status.operation.instrument.trigger_blender.trigger_overrun.enable = status.operation.instrument.trigger_blender.trigger_overrun.BLND1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B4, set *`operationRegister`* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
status.operation.instrument.trigger_blender.trigger_overrun.enable
   = status.operation.instrument.trigger_blender.trigger_overrun.BLND1
```

Uses a constant to set the bit for blender 1 of the operation status trigger blender overrun enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.trigger_blender.* (on page 7-347)

status.operation.instrument.trigger_timer.*

This attribute contains the operation status trigger timer summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.trigger_timer.condition
operationRegister = status.operation.instrument.trigger_timer.enable
operationRegister = status.operation.instrument.trigger_timer.event
operationRegister = status.operation.instrument.trigger_timer.ntr
operationRegister = status.operation.instrument.trigger_timer.ptr
status.operation.instrument.trigger_timer.enable = operationRegister
status.operation.instrument.trigger_timer.ntr = operationRegister
status.operation.instrument.trigger_timer.ptr = operationRegister
```

operationRegister	The status of the operation status trigger timer summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status trigger timer summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.trigger_timer.TRIGGER_OVERRUN`
	`status.operation.instrument.trigger_timer.TRGOVR`
	Set bit indicates one or more enabled bits for the operation status trigger timer overrun register is set.
	Bit B10 decimal value: 1,024
	Binary value:
	0100 0000 0000

B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *operationRegister* to 1024.

Example

```
status.operation.instrument.trigger_timer.enable = 1024
```

Uses the decimal value to set the TRGOVR bit of the operation status trigger timer summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.trigger_timer.trigger_overrun.* (on page 7-352)

status.operation.instrument.trigger_timer.trigger_overrun.*

This attribute contains the operation status trigger timer overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```
operationRegister =
   status.operation.instrument.trigger_timer.trigger_overrun.condition
operationRegister =
   status.operation.instrument.trigger_timer.trigger_overrun.enable
operationRegister =
   status.operation.instrument.trigger_timer.trigger_overrun.event
operationRegister =
   status.operation.instrument.trigger_timer.trigger_overrun.ntr
operationRegister =
   status.operation.instrument.trigger_timer.trigger_overrun.ptr
status.operation.instrument.trigger_timer.trigger_overrun.enable =
   operationRegister
status.operation.instrument.trigger_timer.trigger_overrun.ntr =
   operationRegister
status.operation.instrument.trigger_timer.trigger_overrun.ptr =
   operationRegister
```

operationRegister	The status of the operation status trigger timer trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger timer overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `18` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates the specified timer generated an action overrun because it was still processing a delay from a previous trigger when a new trigger was received.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.trigger_timer.trigger_overrun.TMR1`	2
B2	`status.operation.instrument.trigger_timer.trigger_overrun.TMR2`	4
B3	`status.operation.instrument.trigger_timer.trigger_overrun.TMR3`	8
B4	`status.operation.instrument.trigger_timer.trigger_overrun.TMR4`	16
B5	`status.operation.instrument.trigger_timer.trigger_overrun.TMR5`	32
B6	`status.operation.instrument.trigger_timer.trigger_overrun.TMR6`	64
B7	`status.operation.instrument.trigger_timer.trigger_overrun.TMR7`	128
B8	`status.operation.instrument.trigger_timer.trigger_overrun.TMR8`	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status trigger timer trigger overrun enable register, set `status.operation.instrument.trigger_timer.trigger_overrun.enable = status.operation.instrument.trigger_timer.trigger_overrun.TMR1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B4, set *`operationRegister`* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
status.operation.instrument.trigger_timer.trigger_overrun.enable =
   status.operation.instrument.trigger_timer.trigger_overrun.TMR3
```

Uses a constant to set the timer 3 bit of the operation status trigger timer overrun enable register.

Example 2

```
status.operation.instrument.trigger_timer.trigger_overrun.enable = 18
```

Uses a constant to set timer bits B1 and B4 of the operation status trigger timer overrun enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.trigger_timer.* (on page 7-351)

status.operation.instrument.tsplink.*

This attribute contains the operation status TSP-Link summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	1024 (All bits set)

Usage

```
operationRegister = status.operation.instrument.tsplink.condition
operationRegister = status.operation.instrument.tsplink.enable
operationRegister = status.operation.instrument.tsplink.event
operationRegister = status.operation.instrument.tsplink.ntr
operationRegister = status.operation.instrument.tsplink.ptr
status.operation.instrument.tsplink.enable = operationRegister
status.operation.instrument.tsplink.ntr = operationRegister
status.operation.instrument.tsplink.ptr = operationRegister
```

operationRegister	The status of the operation status TSP-Link summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status TSP-Link summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.tsplink.TRIGGER_OVERRUN`
	`status.operation.instrument.tsplink.TRGOVR`
	Set bit indicates one or more enabled bits for the operation status TSP-Link overrun register is set.
	Bit B10 decimal value: 1,024
	Binary value:
	0100 0000 0000

B11 to B15	Not used

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *`operationRegister`* to 1024.

Example

```
status.operation.instrument.tsplink.enable = 1024
```

Uses the decimal value to set the trigger overrun bit of the operation status TSP-Link summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.tsplink.trigger_overrun.* (on page 7-356)

status.operation.instrument.tsplink.trigger_overrun.*

This attribute contains the operation status TSP-Link overrun register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	14 (All bits set)

Usage

```
operationRegister =
   status.operation.instrument.tsplink.trigger_overrun.condition
operationRegister = status.operation.instrument.tsplink.trigger_overrun.enable
operationRegister = status.operation.instrument.tsplink.trigger_overrun.event
operationRegister = status.operation.instrument.tsplink.trigger_overrun.ntr
operationRegister = status.operation.instrument.tsplink.trigger_overrun.ptr
status.operation.instrument.tsplink.trigger_overrun.enable = operationRegister
status.operation.instrument.tsplink.trigger_overrun.ntr = operationRegister
status.operation.instrument.tsplink.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status TSP-link overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status TSP-link overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `10` is read as the value of the condition register, the binary equivalent is 0000 0000 0000 1010. This value indicates that bit B1 and bit B3 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates that the specified line generated an action overrun when triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.tsplink.trigger_overrun.LINE1`	2
B2	`status.operation.instrument.tsplink.trigger_overrun.LINE2`	4
B3	`status.operation.instrument.tsplink.trigger_overrun.LINE3`	8
B4 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status TSP-Link overrun enable register, set `status.operation.instrument.tsplink.trigger_overrun.enable = status.operation.instrument.tsplink.trigger_overrun.LINE1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B3, set *`operationRegister`* to 10 (which is the sum of 2 + 8).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
status.operation.instrument.tsplink.trigger_overrun.enable =
   status.operation.instrument.tsplink.trigger_overrun.LINE1
```

Uses a constant to set the line 1 bit of the operation status TSP-Link overrun enable register.

Example 2

```
status.operation.instrument.tsplink.trigger_overrun.enable = 10
```

Uses the decimal value to set bits for lines 1 and 3 of the operation status TSP-Link overrun enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.instrument.trigger_timer.* (on page 7-351)

status.operation.measuring.*

This attribute contains the operation status measuring summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
operationRegister = status.operation.measuring.condition
operationRegister = status.operation.measuring.enable
operationRegister = status.operation.measuring.event
operationRegister = status.operation.measuring.ntr
operationRegister = status.operation.measuring.ptr
status.operation.measuring.enable = operationRegister
status.operation.measuring.ntr = operationRegister
status.operation.measuring.ptr = operationRegister
```

operationRegister	The status of the operation status measuring summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

Details

These attributes are used to read or write to the operation status measuring summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.measuring.SMUA`	Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constant, *operationRegister* can be set to the decimal equivalent of the bit to set.

Example

`status.operation.measuring.enable =` `  status.operation.measuring.SMUA`	Uses a constant to set the SMUA bit of the operation status measuring summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

status.operation.remote.*

This attribute contains the operation status remote summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2050 (All bits set)

Usage

```
operationRegister = status.operation.remote.condition
operationRegister = status.operation.remote.enable
operationRegister = status.operation.remote.event
operationRegister = status.operation.remote.ntr
operationRegister = status.operation.remote.ptr
status.operation.remote.enable = operationRegister
status.operation.remote.ntr = operationRegister
status.operation.remote.ptr = operationRegister
```

operationRegister	The status of the operation status remote summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status remote summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used

B1	`status.operation.remote.COMMAND_AVAILABLE`
	`status.operation.remote.CAV`
	Set bit indicates there is a command available in the execution queue.
	Bit B1 decimal value: 2
	Binary value: 0000 0000 0000 0010

B2 to B10	Not used

B11	`status.operation.remote.PROMPTS_ENABLED`
	`status.operation.remote.PRMPT`
	Set bit indicates command prompts are enabled.
	Bit B11 decimal value: 2,048
	Binary value: 0000 0100 0000 0000

B12 to B15	Not used

As an example, to set bit B1 of the operation status remote summary enable register, set `status.operation.remote.enable = status.operation.remote.CAV`.

In addition to the above constants, *`operationRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values. For example, to set bits B1 and B11, set *`operationRegister`* to 2,050 (which is the sum of 2 + 2,048).

Example 1

```
status.operation.remote.enable = status.operation.remote.CAV
```

Uses a constant to set the CAV bit, B1, of the operation status remote summary enable register.

Example 2

```
status.operation.remote.enable = 2050
```

Uses the decimal value to set bits B1 and B11 of the operation status remote summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

status.operation.sweeping.*

This attribute contains the operation status sweeping summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
operationRegister = status.operation.sweeping.condition
operationRegister = status.operation.sweeping.enable
operationRegister = status.operation.sweeping.event
operationRegister = status.operation.sweeping.ntr
operationRegister = status.operation.sweeping.ptr
status.operation.sweeping.enable = operationRegister
status.operation.sweeping.ntr = operationRegister
status.operation.sweeping.ptr = operationRegister
```

operationRegister	The status of the operation status sweeping summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

Details

These attributes are used to read or write to the operation status sweeping summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.sweeping.SMUA`	Set bit indicates that SMU A is sweeping. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B5	Not used	Not applicable.

In addition to the above constant, *operationRegister* can be set to the decimal equivalent of the bit to set.

Example

`status.operation.sweeping.enable =` `  status.operation.sweeping.SMUA`	Uses a constant to set the SMUA bit of the operation status sweeping summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

status.operation.trigger_overrun.*

This attribute contains the operation status trigger overrun summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	31,746 (All bits set)

Usage

```
operationRegister = status.operation.trigger_overrun.condition
operationRegister = status.operation.trigger_overrun.enable
operationRegister = status.operation.trigger_overrun.event
operationRegister = status.operation.trigger_overrun.ntr
operationRegister = status.operation.trigger_overrun.ptr
status.operation.trigger_overrun.enable = operationRegister
status.operation.trigger_overrun.ntr = operationRegister
status.operation.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status trigger overrun summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger overrun summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02600e+03` (which is 1,026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15 **	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0 *
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

The bits in this register summarize events in other registers. A set bit in this summary register indicates that an enabled event in one of the summarized registers is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.operation.trigger_overrun.SMUA`	Set bit indicates one of the enabled bits in the operation status SMU A trigger overrun event register is set. Bit B1 decimal value: 2
B2-B9	Not used	Not applicable.
B10	`status.operation.trigger_overrun.TRIGGER_BLENDER` `status.operation.trigger_overrun.TRGBLND`	Set bit indicates one of the enabled bits in the operation status trigger blender overrun event register is set. Bit B10 decimal value: 1,024
B11	`status.operation.trigger_overrun.TRIGGER_TIMER` `status.operation.trigger_overrun.TRGTMR`	Set bit indicates one of the enabled bits in the operation status trigger timer overrun event register is set. Bit B11 decimal value: 2,048
B12	`status.operation.trigger_overrun.DIGITAL_IO` `status.operation.trigger_overrun.DIGIO`	Set bit indicates one of the enabled bits in the operation status digital I/O overrun event register is set. Bit B12 decimal value: 4,096
B13	`status.operation.trigger_overrun.TSPLINK`	Set bit indicates one of the enabled bits in the operation status TSP-Link overrun event register is set. Bit B13 decimal value: 8,192
B14	`status.operation.trigger_overrun.LAN`	Set bit indicates one of the enabled bits in the operation status LAN trigger overrun event register is set. Bit B14 decimal value: 16,384
B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status trigger overrun summary enable register, set `status.operation.trigger_overrun.enable = status.operation.trigger_overrun.SMUA`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B10, set *operationRegister* to 1,026 (which is the sum of 2 + 1,024).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
operationRegister =
   status.operation.trigger_overrun.SMUA +
   status.operation.trigger_overrun.TRGBLND
status.operation.trigger_overrun.enable = operationRegister
```

Uses constants to set bit B1 and bit B10 of the operation status trigger overrun summary enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

status.operation.user.*

These attributes manage the operation status user register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (RW)	Yes	Status reset	Not saved	0
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
operationRegister = status.operation.user.condition
operationRegister = status.operation.user.enable
operationRegister = status.operation.user.event
operationRegister = status.operation.user.ntr
operationRegister = status.operation.user.ptr
status.operation.user.condition = operationRegister
status.operation.user.enable = operationRegister
status.operation.user.ntr = operationRegister
status.operation.user.ptr = operationRegister
```

operationRegister	The status of the operation status user register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status user registers. Reading a status register returns a value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bits B0 and B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.operation.user.BIT0`	1
B1	`status.operation.user.BIT1`	2
B2	`status.operation.user.BIT2`	4
B3	`status.operation.user.BIT3`	8
B4	`status.operation.user.BIT4`	16
B5	`status.operation.user.BIT5`	32
B6	`status.operation.user.BIT6`	64
B7	`status.operation.user.BIT7`	128
B8	`status.operation.user.BIT8`	256
B9	`status.operation.user.BIT9`	512
B10	`status.operation.user.BIT10`	1,024
B11	`status.operation.user.BIT11`	2,048
B12	`status.operation.user.BIT12`	4,096
B13	`status.operation.user.BIT13`	8,192
B14	`status.operation.user.BIT14`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the operation status user enable register, set `status.operation.user.enable = status.operation.user.BIT0`.

In addition to the above constants, *`operationRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values. For example, to set bits B11 and B14, set *`operationRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.user.BIT11 + status.operation.user.BIT14
status.operation.user.enable = operationRegister
```

Uses constants to set bits B11 and B14 of the operation status user enable register.

Example 2

```
-- 18432 = binary 0100 1000 0000 0000
operationRegister = 18432
status.operation.enable = operationRegister
```

Uses the decimal value to set bits B11 and B14 of the operation status user enable register.

Also see

Operation Status Registers (on page 15-9)
status.operation.* (on page 7-329)

status.questionable.*

These attributes manage the questionable status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	13,056 (All bits set)

Usage

```
questionableRegister = status.questionable.condition
questionableRegister = status.questionable.enable
questionableRegister = status.questionable.event
questionableRegister = status.questionable.ntr
questionableRegister = status.questionable.ptr
status.questionable.enable = questionableRegister
status.questionable.ntr = questionableRegister
status.questionable.ptr = questionableRegister
```

questionableRegister	The status of the questionable status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status registers. Reading a status register returns a value. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.22880e+04 (which is 12,288) is read as the value of the condition register, the binary equivalent is 0011 0000 0000 0000. This value indicates that bits B12 and B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0-B7	Not used	Not available
B8	`status.questionable.CALIBRATION` `status.questionable.CAL`	An enabled bit in the questionable status calibration summary event register is set. Bit B6 decimal value:256
B9	`status.questionable.UNSTABLE_OUTPUT` `status.questionable.UO`	An enabled bit in the questionable status unstable output summary event register is set. Bit B9 decimal value: 512
B10-B11	Not used	Not available
B12	`status.questionable.OVER_TEMPERATURE` `status.questionable.OTEMP`	An enabled bit in the questionable status over temperature summary event register is set. Bit B12 decimal value: 4,096
B13	`status.questionable.INSTRUMENT_SUMMARY` `status.questionable.INST`	An enabled bit in the questionable status instrument summary event register is set. Bit B13 decimal value: 8,192
B14-B15	Not used	Not available

As an example, to set bit B9 of the questionable status enable register, set `status.questionable.enable = status.questionable.UO`.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`questionableRegister`* to the sum of their decimal weights. For example, to set bits B12 and B13, set *`questionableRegister`* to 12,288 (which is the sum of 4,096 + 8,192).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Also see

Questionable Status Registers (on page 15-13)

status.questionable.calibration.*

This attribute contains the questionable status calibration summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
questionableRegister = status.questionable.calibration.condition
questionableRegister = status.questionable.calibration.enable
questionableRegister = status.questionable.calibration.event
questionableRegister = status.questionable.calibration.ntr
questionableRegister = status.questionable.calibration.ptr
status.questionable.calibration.enable = questionableRegister
status.questionable.calibration.ntr = questionableRegister
status.questionable.calibration.ptr = questionableRegister
```

questionableRegister	The status of the questionable status calibration summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than zero (0) is 2

Details

These attributes are used to read or write to the questionable status calibration summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.questionable.calibration.SMUA	Set bit indicates that the calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B1 decimal value: 2 Binary value: 0000 0010
B2 to B15	Not used	Not applicable.

In addition to the above constant, *questionableRegister* can be set to the decimal equivalent of the bit to set.

Example

`status.questionable.calibration.enable = status.questionable.calibration.SMUA`	Uses a constant to set the SMUA bit of the questionable status calibration summary enable register.

Also see

Questionable Status Registers (on page 15-13)
status.questionable.* (on page 7-367)

status.questionable.instrument.*

This attribute contains the questionable status instrument summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
questionableRegister = status.questionable.instrument.condition
questionableRegister = status.questionable.instrument.enable
questionableRegister = status.questionable.instrument.event
questionableRegister = status.questionable.instrument.ntr
questionableRegister = status.questionable.instrument.ptr
status.questionable.instrument.enable = questionableRegister
status.questionable.instrument.ntr = questionableRegister
status.questionable.instrument.ptr = questionableRegister
```

questionableRegister	The status of the questionable status instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the questionable status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.questionable.instrument.SMUA`	Set bit indicates that one or more enabled bits in the questionable status SMU A summary event register are set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2 to B15	Not used	Not applicable.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set.

Example

```
status.questionable.instrument.enable =
   status.questionable.instrument.SMUA
```

Uses a constant to set the SMUA bit of the questionable status instrument summary enable register.

Also see

Questionable Status Registers (on page 15-13)
status.questionable.* (on page 7-367)

status.questionable.instrument.smuX.*

This attribute contains the questionable status SMU summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	4864 (All bits set)

Usage

```
questionableRegister = status.questionable.instrument.smuX.condition
questionableRegister = status.questionable.instrument.smuX.enable
questionableRegister = status.questionable.instrument.smuX.event
questionableRegister = status.questionable.instrument.smuX.ntr
questionableRegister = status.questionable.instrument.smuX.ptr
status.questionable.instrument.smuX.enable = questionableRegister
status.questionable.instrument.smuX.ntr = questionableRegister
status.questionable.instrument.smuX.ptr = questionableRegister
```

questionableRegister	The status of the questionable status SMU summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status instrument SMU summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `7.68000e+02` (which is 768) is read as the value of the condition register, the binary equivalent is 0000 0011 0000 0000. This value indicates that bit B8 and bit B9 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0 to B7	Not used	Not applicable.
B8	`status.questionable.instrument.smu`*`X`*`.CALIBRATION` `status.questionable.instrument.smu`*`X`*`.CAL`	Set bit indicates that the calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B8 decimal value: 256
B9	`status.questionable.instrument.smu`*`X`*`.UNSTABLE_OUTPUT` `status.questionable.instrument.smu`*`X`*`.UO`	Set bit indicates that an unstable output condition was detected. Bit B9 decimal value: 512
B10 to B11	Not used	Not applicable
B12	`status.questionable.instrument.smu`*`X`*`.OVER_TEMPERATURE` `status.questionable.instrument.smu`*`X`*`.OTEMP`	Set bit indicates that an over temperature condition was detected. Bit B12 decimal value: 4,096
B13 to B15	Not used	Not applicable.

As an example, to set bit B8 of the questionable status SMU A summary enable register, set `status.questionable.instrument.smua.enable = status.questionable.instrument.smua.CAL`.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`questionableRegister`* to the sum of their decimal weights. For example, to set bits B8 and B9, set *`questionableRegister`* to 768 (which is the sum of 256 + 512).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
questionableRegister = status.questionable.instrument.smua.CAL +
   status.questionable.instrument.smua.UO
status.questionable.instrument.smua.enable = questionableRegister
```

Uses constants to set bit B8 and bit B9 of the questionable status SMU A summary enable register.

Also see

Questionable Status Registers (on page 15-13)
status.operation.* (on page 7-329)

status.questionable.over_temperature.*

This attribute contains the questionable status over temperature summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
questionableRegister = status.questionable.over_temperature.condition
questionableRegister = status.questionable.over_temperature.enable
questionableRegister = status.questionable.over_temperature.event
questionableRegister = status.questionable.over_temperature.ntr
questionableRegister = status.questionable.over_temperature.ptr
status.questionable.over_temperature.enable = questionableRegister
status.questionable.over_temperature.ntr = questionableRegister
status.questionable.over_temperature.ptr = questionableRegister
```

operationRegister	The status of the questionable status over temperature summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the questionable status over temperature summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`status.questionable.over_temperature.SMUA`	Set bit indicates that an over temperature condition was detected. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constants, `questionableRegister` can be set to the numeric equivalent of the bit to set.

Example

`status.questionable.over_temperature.enable =` `  status.questionable.over_temperature.SMUA`	Sets the SMU A bit in the questionable status over temperature summary enable register using a constant.

Also see

Questionable Status Registers (on page 15-13)
status.questionable.* (on page 7-367)

status.questionable.unstable_output.*

This attribute contains the questionable status unstable output summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (All bits set)

Usage

```
questionableRegister = status.questionable.unstable_output.condition
questionableRegister = status.questionable.unstable_output.enable
questionableRegister = status.questionable.unstable_output.event
questionableRegister = status.questionable.unstable_output.ntr
questionableRegister = status.questionable.unstable_output.ptr
status.questionable.unstable_output.enable = questionableRegister
status.questionable.unstable_output.ntr = questionableRegister
status.questionable.unstable_output.ptr = questionableRegister
```

operationRegister	The status of the questionable status unstable output summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 2

Details

These attributes are used to read or write to the questionable status unstable output summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights.

Bit	Value	Description
B0	Not used	Not applicable.
B1	status.questionable.unstable_output.SMUA	Set bit indicates that an unstable output condition was detected. Bit B1 decimal value: 2 Binary value: 0000 0010
B2-B15	Not used	Not applicable.

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set.

Example

```
status.questionable.unstable_output.enable =
    status.questionable.unstable_output.SMUA
```

Uses a constant to set the SMU A bit in the questionable status unstable output summary enable register bit.

Also see

Questionable Status Registers (on page 15-13)
status.questionable.* (on page 7-367)

status.request_enable

This attribute stores the service request (SRQ) enable register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
requestSRQEnableRegister = status.request_enable
status.request_enable = requestSRQEnableRegister
```

requestSRQEnableRegister	The status of the service request (SRQ) enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

This attribute is used to read or write to the service request enable register. Reading the service request enable register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value

B0	`status.MEASUREMENT_SUMMARY_BIT`
	`status.MSB`
	Set summary bit indicates that an enabled event in the Measurement Event Register has occurred.
	Bit B0 decimal value: 1

B1	`status.SYSTEM_SUMMARY_BIT`
	`status.SSB`
	Set summary bit indicates that an enabled event in the System Summary Register has occurred.
	Bit B1 decimal value: 2

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the error queue.
	Bit B2 decimal value: 4

B3	`status.QUESTIONABLE_SUMMARY_BIT`
	`status.QSB`
	Set summary bit indicates that an enabled event in the Questionable Status Register has occurred.
	Bit B3 decimal value: 8

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred.
	Bit B5 decimal value: 32

B6	Not used

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled event in the Operation Status Register has occurred.
	Bit B7 decimal value: 128

As an example, to set bit B0 of the service request enable register, set `status.request_enable = status.MSB`.

In addition to the above values, *`requestSRQEnableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`requestSRQEnableRegister`* to the sum of their decimal weights. For example, to set bits B0 and B7, set *`requestSRQEnableRegister`* to 129 (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example 1

```
requestSRQEnableRegister = status.MSB + status.OSB
status.request_enable = requestSRQEnableRegister
```

Uses constants to set the MSB and OSB bits of the service request (SRQ) enable register.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the MSB and OSB bits of the service request (SRQ) enable register.

Also see

Status byte and service request (SRQ) (on page 15-15)
Status model overview (on page 15-1)
status.condition (on page 7-311)
status.system.* (on page 7-383)

status.request_event

This attribute stores the service request (SRQ) event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

```
requestSRQEventRegister = status.request_event
```

requestSRQEventRegister	The status of the request event register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the service request event register, which is returned as a numeric value. Reading this register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value and description

B0	`status.MEASUREMENT_SUMMARY_BIT`
	`status.MSB`
	Set summary bit indicates that an enabled event in the Measurement Event Register has occurred.
	Bit B0 decimal value: 1

B1	`status.SYSTEM_SUMMARY_BIT`
	`status.SSB`
	Set summary bit indicates that an enabled event in the System Summary Register has occurred.
	Bit B1 decimal value: 2

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the error queue.
	Bit B2 decimal value: 4

B3	`status.QUESTIONABLE_SUMMARY_BIT`
	`status.QSB`
	Set summary bit indicates that an enabled event in the Questionable Status Register has occurred.
	Bit B3 decimal value: 8

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred.
	Bit B5 decimal value: 32

B6	Not used

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled event in the Operation Status Register has occurred.
	Bit B7 decimal value: 128

In addition to the above constants, *requestEventRegister* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *requestEventRegister* contains the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

```
requestEventRegister = status.request_event
print(requestEventRegister)
```

Reads the status request event register.
Sample output:

```
1.29000e+02
```

Converting this output (129) to its binary equivalent yields 1000 0001.
Therefore, this output indicates that the set bits of the status request event register are presently B0 (MSB) and B7 (OSB).

Also see

status.condition (on page 7-311)
status.system.* (on page 7-383)
Status model overview (on page 15-5)
Status byte and service request (SRQ) (on page 15-15)

status.reset()

This function resets all bits in the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.reset()
```

Details

This function clears all status data structure registers (enable, event, NTR, and PTR) to their default values. For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19).

Example

```
status.reset()
```

Resets the instrument status model.

Also see

Status model (on page 15-1)

status.standard.*

These attributes manage the standard event status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	253 (All bits set)

Usage

```
standardRegister = status.standard.condition
standardRegister = status.standard.enable
standardRegister = status.standard.event
standardRegister = status.standard.ntr
standardRegister = status.standard.ptr
status.standard.enable = standardRegister
status.standard.ntr = standardRegister
status.standard.ptr = standardRegister
```

standardRegister	The status of the standard event status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the standard event status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value

B0	`status.standard.OPERATION_COMPLETE`
	`status.standard.OPC`
	Set bit indicates that all pending selected instrument operations are completed and the instrument is ready to accept new commands. The bit is set in response to an `*OPC` command. The `opc()` function can be used in place of the `*OPC` command.
	Bit B0 decimal value: 1

B1	Not used

B2	`status.standard.QUERY_ERROR`
	`status.standard.QYE`
	Set bit indicates that you attempted to read data from an empty Output Queue.
	Bit B2 decimal value: 4

B3	`status.standard.DEVICE_DEPENDENT_ERROR`
	`status.standard.DDE`
	Set bit indicates that an instrument operation did not execute properly due to some internal condition.
	Bit B3 decimal value: 8

B4	`status.standard.EXECUTION_ERROR`
	`status.standard.EXE`
	Set bit indicates that the instrument detected an error while trying to execute a command.
	Bit B4 decimal value: 16

B5	`status.standard.COMMAND_ERROR`
	`status.standard.CME`
	Set bit indicates that a command error has occurred. Command errors include:
	IEEE Std 488.2 syntax error: Instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
	Semantic error: Instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented.
	GET error: The instrument received a Group Execute Trigger (GET) inside a program message.
	Bit B5 decimal value: 32

B6	`status.standard.USER_REQUEST`
	`status.standard.URQ`
	Set bit indicates that the LOCAL key on the instrument front panel was pressed.
	Bit B6 decimal value: 64

B7	`status.standard.POWER_ON`
	`status.standard.PON`
	Set bit indicates that the instrument has been turned off and turned back on since the last time this register has been read.
	Bit B7 decimal value: 128

B8 to B15	Not used

As an example, to set bit B0 of the standard event status enable register, set `status.standard.enable = status.standard.OPC`.

In addition to the above constants, *`standardRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`standardRegister`* to the sum of their decimal weights. For example, to set bits B0 and B4, set *`standardRegister`* to 17 (which is the sum of 1 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example 1

```
standardRegister = status.standard.OPC + status.standard.EXE
status.standard.enable = standardRegister
```

Uses constants to set the OPC and EXE bits of the standard event status enable register.

Example 2

```
-- decimal 17 = binary 0001 0001
standardRegister = 17
status.standard.enable = standardRegister
```

Uses the decimal value to set the OPC and EXE bits of the standard event status enable register.

Also see

Standard Event Register (on page 15-21)

status.system.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 1 through 14.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system.condition
enableRegister = status.system.enable
enableRegister = status.system.event
enableRegister = status.system.ntr
enableRegister = status.system.ptr
status.system.enable = enableRegister
status.system.ntr = enableRegister
status.system.ptr = enableRegister
```

enableRegister	The status of the system summary register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system.EXTENSION_BIT` `status.system.EXT`	1
B1	`status.system.NODE1`	2
B2	`status.system.NODE2`	4
B3	`status.system.NODE3`	8
B4	`status.system.NODE4`	16
B5	`status.system.NODE5`	32
B6	`status.system.NODE6`	64
B7	`status.system.NODE7`	128
B8	`status.system.NODE8`	256
B9	`status.system.NODE9`	512
B10	`status.system.NODE10`	1,024
B11	`status.system.NODE11`	2,048
B12	`status.system.NODE12`	4,096
B13	`status.system.NODE13`	8,192
B14	`status.system.NODE14`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary status enable register, set `status.system.enable = status.system.enable.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example 1

```
enableRegister = status.system.NODE11 + status.system.NODE14
status.system.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary enable register.

Also see

status.system2.* (on page 7-385)
System summary and standard event registers (on page 15-7)

status.system2.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 15 through 28.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system2.condition
enableRegister = status.system2.enable
enableRegister = status.system2.event
enableRegister = status.system2.ntr
enableRegister = status.system2.ptr
status.system2.enable = enableRegister
status.system2.ntr = enableRegister
status.system2.ptr = enableRegister
```

enableRegister	The status of the system summary 2 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system2.EXTENSION_BIT` `status.system2.EXT`	1
B1	`status.system2.NODE15`	2
B2	`status.system2.NODE16`	4
B3	`status.system2.NODE17`	8
B4	`status.system2.NODE18`	16
B5	`status.system2.NODE19`	32
B6	`status.system2.NODE20`	64
B7	`status.system2.NODE21`	128
B8	`status.system2.NODE22`	256
B9	`status.system2.NODE23`	512
B10	`status.system2.NODE24`	1,024
B11	`status.system2.NODE25`	2,048
B12	`status.system2.NODE26`	4,096
B13	`status.system2.NODE27`	8,192
B14	`status.system2.NODE28`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 2 enable register, set `status.system2.enable = status.system2.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system2.NODE25 + status.system2.NODE28
status.system2.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 2 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system2.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary 2 enable register.

Also see

status.system.* (on page 7-383)
status.system3.* (on page 7-388)
System summary and standard event registers (on page 15-7)

status.system3.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 29 through 42.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system3.condition
enableRegister = status.system3.enable
enableRegister = status.system3.event
enableRegister = status.system3.ntr
enableRegister = status.system3.ptr
status.system3.enable = enableRegister
status.system3.ntr = enableRegister
status.system3.ptr = enableRegister
```

enableRegister	The status of the system summary 3 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0 and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system3.EXTENSION_BIT` `status.system3.EXT`	1
B1	`status.system3.NODE29`	2
B2	`status.system3.NODE30`	4
B3	`status.system3.NODE31`	8
B4	`status.system3.NODE32`	16
B5	`status.system3.NODE33`	32
B6	`status.system3.NODE34`	64
B7	`status.system3.NODE35`	128
B8	`status.system3.NODE36`	256
B9	`status.system3.NODE37`	512
B10	`status.system3.NODE38`	1,024
B11	`status.system3.NODE39`	2,048
B12	`status.system3.NODE40`	4,096
B13	`status.system3.NODE41`	8,192
B14	`status.system3.NODE42`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 3 enable register, set `status.system3.enable = status.system3.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system3.NODE39 + status.system3.NODE42
status.system3.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 3 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system3.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary 3 enable register.

Also see

status.system2.* (on page 7-385)
status.system4.* (on page 7-390)
System summary and standard event registers (on page 15-7)

status.system4.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 43 through 56.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system4.condition
enableRegister = status.system4.enable
enableRegister = status.system4.event
enableRegister = status.system4.ntr
enableRegister = status.system4.ptr
status.system4.enable = enableRegister
status.system4.ntr = enableRegister
status.system4.ptr = enableRegister
```

enableRegister	The status of the system summary 4 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system4.EXTENSION_BIT` `status.system4.EXT`	1
B1	`status.system4.NODE43`	2
B2	`status.system4.NODE44`	4
B3	`status.system4.NODE45`	8
B4	`status.system4.NODE46`	16
B5	`status.system4.NODE47`	32
B6	`status.system4.NODE48`	64
B7	`status.system4.NODE49`	128
B8	`status.system4.NODE50`	256
B9	`status.system4.NODE51`	512
B10	`status.system4.NODE52`	1,024
B11	`status.system4.NODE53`	2,048
B12	`status.system4.NODE54`	4,096
B13	`status.system4.NODE55`	8,192
B14	`status.system4.NODE56`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 4 enable register, set `status.system4.enable = status.system4.enable.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system4.NODE53 + status.system4.NODE56
status.system2.enable = enableRegister
```

Uses constants to set bit B11 and bit B14 of the system summary 4 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system4.enable = enableRegister
```

Uses a decimal value to set bit B11 and bit B14 of the system summary 4 enable register.

Also see

status.system3.* (on page 7-388)
status.system5.* (on page 7-392)
System summary and standard event registers (on page 15-7)

status.system5.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 57 through 64.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```
enableRegister = status.system5.condition
enableRegister = status.system5.enable
enableRegister = status.system5.event
enableRegister = status.system5.ntr
enableRegister = status.system5.ptr
status.system5.enable = enableRegister
status.system5.ntr = enableRegister
status.system5.ptr = enableRegister
```

enableRegister	The status of the system summary 5 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.30000e+02` (which is 130) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0010. This value indicates that bit B1 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 15-1) and Enable and transition registers (on page 15-19). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.system5.NODE57`	2
B2	`status.system5.NODE58`	4
B3	`status.system5.NODE59`	8
B4	`status.system5.NODE60`	16
B5	`status.system5.NODE61`	32
B6	`status.system5.NODE62`	64
B7	`status.system5.NODE63`	128
B8	`status.system5.NODE64`	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the system summary 5 enable register, set `status.system5.enable = status.system5.NODE57`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights. For example, to set bits B1 and B4, set *`enableRegister`* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system5.NODE57 + status.system5.NODE60
status.system2.enable = enableRegister
```

Uses constants to set bits B1 and B4 of the system summary 5 enable register.

Example 2

```
-- decimal 18 = binary 0000 0000 0001 0010
enableRegister = 18
status.system5.enable = enableRegister
```

Uses the decimal value to set bits B1 and B4 of the system summary 5 enable register.

Also see

status.system4.* (on page 7-390)
System summary and standard event registers (on page 15-7)

SweepILinMeasureV()

This KISweep factory script function performs a linear current sweep with voltage measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepILinMeasureV(smuX, starti, stopi, stime, points)
```

starti	Sweep start current in amperes
stopi	Sweep stop current in amperes
stime	Settling time in seconds; occurs after stepping the source and before making a measurement
points	Number of sweep points (must be ≥ 2)

Details

Data for voltage measurements, current source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

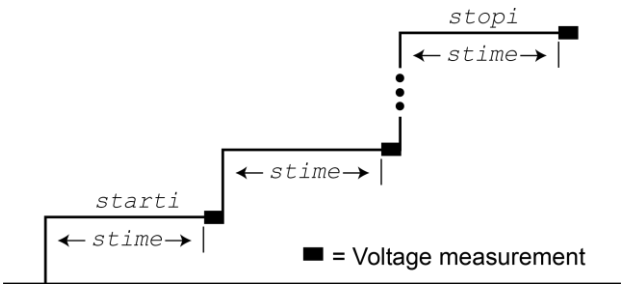
Performs a linear current sweep with voltage measured at every step (point):

1. Sets the SMU to output *starti* amperes, allows the source to settle for *stime* seconds, and then makes a voltage measurement.
2. Sets the SMU to output the next amperes step, allows the source to settle for *stime* seconds, and then makes a voltage measurement.
3. Repeats the above sequence until the voltage is measured on the *stopi* amperes step.

The linear step size is automatically calculated as follows:

```
step = (stopi – starti) / (points – 1)
```

Figure 120: SweepILinMeasureV()



Example

```
SweepILinMeasureV(smua, -1e-3, 1e-3, 0, 100)
```

This function performs a 100-point linear current sweep starting at −1 mA and stopping at +1 mA. Voltage is measured at every step (point) in the sweep. Because *stime* is set for 0 s, voltage is measured as quickly as possible after each current step.

Also see

KISweep factory script (on page 5-22)

SweepIListMeasureV()

This KISweep factory script function performs a current list sweep with voltage measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepIListMeasureV(smuX, ilist, stime, points)
```

ilist	Arbitrary list of current source values; *ilist = {value1, value2, ...valueN}*
stime	Settling time in seconds; occurs after stepping the source and before making a measurement
points	Number of sweep points (must be ≥ 2)

Details

Data for voltage measurements, current source values, and timestamps are stored in smu*X*.nvbuffer1.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a current list sweep with voltage measured at every step (point):

1. Sets the SMU to output *ilist* amperes value, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
2. Sets the SMU to output the next *ilist* step, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
3. Repeats the above sequence until the voltage is measured for the last amperes value. The last point in the list to be measured is *points*.

Example

```
testilist = {-100e-9, 100e-9, -1e-6, 1e-6, -1e-3, 1e-3}

SweepIListMeasureV(smua, testilist, 500e-3, 6)
```

This function performs a six-point current list sweep starting at the first point in testilist. Voltage is measured at every step (point) in the sweep. The source is allowed to settle on each step for 500 ms before a measurement is performed.

Also see

KISweep factory script (on page 5-22)

SweepILogMeasureV()

This KISweep factory script function performs a logarithmic current sweep with voltage measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepILogMeasureV(smuX, starti, stopi, stime, points)
```

starti	Sweep start current in amperes
stopi	Sweep stop current in amperes
stime	Settling time in seconds; occurs after stepping the source and before making a measurement
points	Number of sweep points (must be ≥ 2)

Details

Data for voltage measurements, current source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a logarithmic current sweep with voltage measured at every step (point):

1. Sets the SMU to output *starti* amperes value, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
2. Sets the SMU to output the next amperes step, allows the source to settle for *stime* seconds, and then performs a voltage measurement.
3. Repeats the above sequence until the voltage is measured on the *stopi* amperes step.

The source level at each step (`SourceStepLevel`) is automatically calculated as follows:

MeasurePoint = The step point number for a measurement

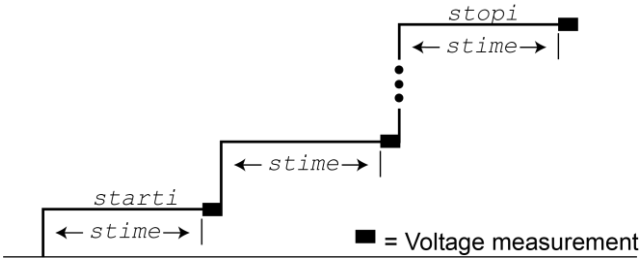
For example, for a five-point sweep (*points* = 5), a measurement is performed at *MeasurePoint* 1, 2, 3, 4, and 5.

LogStepSize = (log10(stopi) – log10(starti)) / (points – 1)

*LogStep = (MeasurePoint – 1) * (LogStepSize)*

*SourceStepLevel = antilog(LogStep) * starti*

Figure 121: SweepILogMeasureV()



Example

```
SweepILogMeasureV(smua, 0.01, 0.1, 0.001, 5)
```

This function performs a five-point linear current sweep starting at 10 mA and stopping at 100 mA. Voltage is measured at every step (point) in the sweep. The source is allowed to settle on each step for 1 ms before a measurement is made.

The following table contains log values and corresponding source levels for the five-point logarithmic sweep:

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
1	0.25	0.0	0.01 A
2	0.25	0.25	0.017783 A
3	0.25	0.5	0.031623 A
4	0.25	0.75	0.056234 A
5	0.25	1.0	0.1 A

Also see

KISweep factory script (on page 5-22)

SweepVLinMeasureI()

This KISweep factory script function performs a linear voltage sweep with current measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepVLinMeasureI(smuX, startv, stopv, stime, points)
```

startv	Sweep start voltage in volts
stopv	Sweep stop voltage in volts
stime	Settling time in seconds; occurs after stepping the source and before making a measurement
points	Number of sweep points (must be ≥ 2)

Details

Data for current measurements, voltage source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

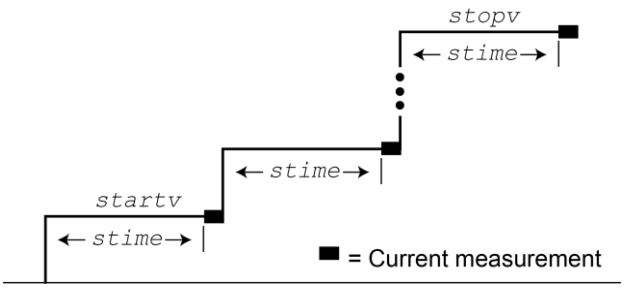
Performs a linear voltage sweep with current measured at every step (point):

1. Sets the SMU to output *startv* amperes, allows the source to settle for *stime* seconds, and then makes a current measurement.
2. Sets the SMU to output the next amperes step, allows the source to settle for *stime* seconds, and then makes a voltage measurement.
3. Repeats the above sequence until the voltage is measured on the *stopv* amperes step.

The linear step size is automatically calculated as follows:

```
step = (stopv – startv) / (points – 1)
```

Figure 122: SweepVLinMeasureI()



Example

`SweepVLinMeasureI(smua, -1, 1, 1e-3, 1000)`	This function performs a 1000-point linear voltage sweep starting at -1 V and stopping at +1 V. Current is measured at every step (point) in the sweep after a 1 ms source settling period.

Also see

KISweep factory script (on page 5-22)

SweepVListMeasureI()

This KISweep factory script function performs a voltage list sweep with current measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`SweepVListMeasureI(smu`*`X`*`, `*`vlist`*`, `*`stime`*`, `*`points`*`)`

`vlist`	Arbitrary list of voltage source values; *`vlist = {value1, value2, ... valueN}`*
`stime`	Settling time in seconds; occurs after stepping the source and before making a measurement
`points`	Number of sweep points (must be ≥ 2)

Details

Data for current measurements, voltage source values, and timestamps are stored in `smu`*`X`*`.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a voltage list sweep with current measured at every step (point):

1. Sets the SMU to output *`vlist`* volts value, allows the source to settle for *`stime`* seconds, and then performs a current measurement.
2. Sets the SMU to output the next *`vlist`* volts value, allows the source to settle for *`stime`* seconds, and then performs a current measurement.
3. Repeats the above sequence until the current is measured for the last volts value. The last point in the list to be measured is *`points`*.

Example

`myvlist = {-0.1, 0.1, -1, 1, -6, 6, -40, 40, 0, 0}`
 	
`SweepVListMeasureI(smua, myvlist, 500E-3, 10)`
	This function performs a 10-point voltage list sweep starting at the first point in `myvlist`. Current is measured at every step (point) in the sweep. The source is allowed to settle on each step for 500 ms before a measurement is performed.

Also see

KISweep factory script (on page 5-22)

SweepVLogMeasureI()

This KISweep factory script function performs a logarithmic voltage sweep with current measured at every step (point).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
SweepVLogMeasureI(smuX, startv, stopv, stime, points)
```

startv	Sweep start voltage in volts
stopv	Sweep stop voltage in volts
stime	Settling time in seconds; occurs after stepping the source and before making a measurement
points	Number of sweep points (must be ≥ 2)

Details

Data for current measurements, voltage source values, and timestamps are stored in `smuX.nvbuffer1`.

If all parameters are omitted when this function is called, this function is executed with the parameters set to the default values.

Performs a logarithmic voltage sweep with current measured at every step (point):

1. Sets the SMU to output *startv* amperes, allows the source to settle for *stime* seconds, and then makes a current measurement.
2. Sets the SMU to output the next volts step, allows the source to settle for *stime* seconds, and then makes a current measurement.
3. Repeats the above sequence until the voltage is measured on the *stopv* volts step.

The source level at each step (`SourceStepLevel`) is automatically calculated as follows:

MeasurePoint = The step point number for a measurement

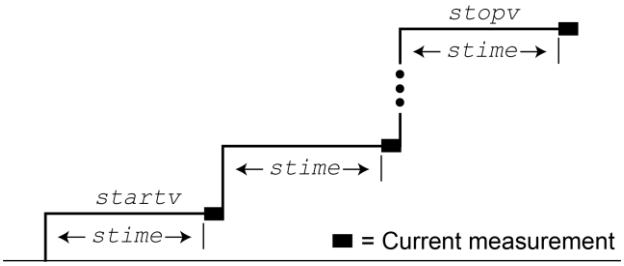
For example, for a five-point sweep (*points* = 5), a measurement is made at `MeasurePoint` 1, 2, 3, 4, and 5.

LogStepSize = (log10(stopi) – log10(starti)) / (points – 1)

*LogStep = (MeasurePoint – 1) * (LogStepSize)*

*SourceStepLevel = antilog(LogStep) * startv*

Figure 123: SweepVLogMeasureI()



Example

```
SweepVLogMeasureI(smua, 1, 10,
   0.001, 5)
```

This function performs a five-point logarithmic voltage sweep starting at 1 V and stopping at 10 V. Current is measured at every step (point) in the sweep after a 1 ms source settling period.

The following table contains log values and corresponding source levels for the five-point logarithmic sweep:

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
1	0.25	0.0	1.0000 V
2	0.25	0.25	1.7783 V
3	0.25	0.5	3.1623 V
4	0.25	0.75	5.6234 V
5	0.25	1.0	10.000 V

Also see

KISweep factory script (on page 5-22)

timer.measure.t()

This function measures the elapsed time since the timer was last reset.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
time = timer.measure.t()
```

time	The elapsed time in seconds (1 µs resolution)

Example 1

```
timer.reset()
-- (intervening code)
time = timer.measure.t()
print(time)
```

This example resets the timer and measures the time since the reset.
Output:

```
1.469077e+01
```

The output varies. The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Example 2

```
beeper.enable = beeper.ON
beeper.beep(0.5, 2400)
print("reset timer")
timer.reset()
delay(0.5)
dt = timer.measure.t()
print("timer after delay:", dt)
beeper.beep(0.5, 2400)
```

Enable the beeper.
Emit a beep and set the beeper.
Reset the timer.
Set a delay.
Verify the duration of the delay before emitting another beep.
Output:

```
reset timer
timer after delay: 5.00e-01
```

Also see

timer.reset() (on page 7-403)

timer.reset()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timer.reset()
```

Example

```
timer.reset()
-- (intervening code)
time = timer.measure.t()
print(time)
```

Resets the timer and then measures the time since the reset.
Output:
`1.469077e+01`
The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Also see

timer.measure.t() (on page 7-402)

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 four)

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

```
trigger.blender[2].clear()
```

Clears the event detector for blender 2.

Also see

None

trigger.blender[N].EVENT_ID

This constant contains the trigger blender event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.blender[N].EVENT_ID
```

eventID	Trigger event number
N	The blender number (up to four)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this trigger blender.

Example

```
digio.trigger[1].stimulus = trigger.blender[2].EVENT_ID
```

Set the trigger stimulus of digital I/O trigger 1 to be controlled by the trigger blender 2 event.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender performs OR operations or AND operations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Trigger blender N reset Recall setup	Not saved	false (AND mode)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The type of operation: ■ `true`: OR operation ■ `false`: AND operation
N	The blender number (up to four)

Details

This command selects whether the blender waits for any one event (`OR`) or waits for all selected events (`AND`) before signaling an output event.

Example

```
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 7-407)

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 four)

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 7-407)

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 four)

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 7-405)
trigger.blender[N].overrun (on page 7-406)
trigger.blender[N].stimulus[M] (on page 7-407)

trigger.blender[N].stimulus[M]

This attribute specifies which events trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger blender N reset	Not saved	0

Usage

```
eventID = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = eventID
```

eventID	The event that triggers the blender action; see **Details**
N	An integer representing the trigger event blender (up to four)
M	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event. The *eventID* parameter can be the event ID of any trigger event.

Use zero to disable the blender input.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smu*X*.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[*N*].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[*N*].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[*N*].EVENT_ID	Occurs when a delay expires

Example

```
digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[5].mode = digio.TRIG_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 7-407)

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 four) on which to wait
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```
digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[5].mode = digio.TRIG_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
print(trigger.blender[1].wait(3))
```

Generate a trigger blender 1 event when a digital I/O trigger happens either on line 3 or 5.
Wait three seconds while checking if trigger blender 1 event has occurred.
If the blender trigger event has happened, then `true` is output. If the trigger event has not happened, then `false` is output after the timeout expires.

Also see

trigger.blender[N].clear() (on page 7-403)

trigger.clear()

This function clears the command interface trigger event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

The trigger event detector indicates if a trigger event has been detected since the last `trigger.wait()` call. `trigger.clear()` clears the trigger event detector and discards the history of command interface trigger events.

Also see

trigger.wait() (on page 7-420)

trigger.EVENT_ID

This constant contains the command interface trigger event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.EVENT_ID
```

eventID	The event ID for the command interface triggers

Details

You can set the stimulus of any trigger object to the value of this constant to have the trigger object respond to command interface trigger events.

Example

```
trigger.timer[1].stimulus = trigger.EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the command interface trigger event.

Also see

None

trigger.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 8)

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

```
trigger.timer[1].clear()
```

Clears trigger timer 1.

Also see

trigger.timer[N].count (on page 7-412)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer *N* reset	Not saved	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

count	Number of times to repeat the trigger (0 to 1,048,575)
N	Trigger timer number (1 to 8)

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example

```
print(trigger.timer[1].count)
```

Read trigger count for timer number 1.

Also see

trigger.timer[N].clear() (on page 7-411)
trigger.timer[N].delay (on page 7-412)
trigger.timer[N].reset() (on page 7-417)

trigger.timer[N].delay

This attribute sets and reads the timer delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Not saved	10e-6 (10 µs)

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

interval	Delay interval in seconds (8 µs to 100 ks)
N	Trigger timer number (1 to 8)

Details

Once the timer is enabled, each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

```
trigger.timer[N].delaylist = {interval}
```

This creates a delay list of one value.

Reading this attribute returns the delay interval that is used the next time the timer is triggered.

Example

```
trigger.timer[1].delay = 50e-6
```

Set the trigger timer 1 to delay for 50 µs.

Also see

trigger.timer[N].reset() (on page 7-417)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer *N* reset	Not saved	10e-6 (10 µs)

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

intervals	Table of delay intervals in seconds
N	Trigger timer number (1 to 8)

Details

Each time the timer is triggered after it is enabled, it uses the next delay period from the array. The default value is an array with one value of 10 µs.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be ≥ 50 µs.

Example

```
trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}
DelayList = trigger.timer[3].delaylist
for x = 1, table.getn(DelayList) do
   print(DelayList[x])
end
```

Set a delay list on trigger timer 3 with three delays (50 µs, 100 µs, and 150 µs).
Read the delay list on trigger timer 3.
Output:

```
5e-05
0.0001
0.00015
```

Also see

trigger.timer[N].reset() (on page 7-417)

trigger.timer[N].EVENT_ID

This constant specifies the trigger timer event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = trigger.timer[*N*].EVENT_ID

eventID	The trigger event number
N	Trigger timer number (1 to 8)

Details

This constant is an identification number that identifies events generated by this timer.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to events from this timer.

Example

```
trigger.timer[1].stimulus = tsplink.trigger[2].EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the TSP-Link trigger 2 event.

Also see

None

trigger.timer[N].overrun

This attribute indicates if an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup Trigger timer N clear Trigger timer N reset	Not applicable	false

Usage

```
overrun = trigger.timer[N].overrun
```

overrun	Trigger overrun state (`true` or `false`)
N	Trigger timer number (1 to 8)

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

```
print(trigger.timer[1].overrun)
```

If an event was ignored, the output is `true`.
If the event was not ignored, the output is `false`.

Also see

trigger.timer[N].reset() (on page 7-417)

trigger.timer[N].passthrough

This attribute enables or disables the timer trigger pass-through mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Not saved	false (disabled)

Usage

```
passthrough = trigger.timer[N].passthrough
trigger.timer[N].passthrough = passthrough
```

passthrough	The state of pass-through mode; set to one of the following values: ■ `true`: Enabled ■ `false`: Disabled
N	Trigger timer number (1 to 8)

Details

When pass-through mode is enabled, triggers are passed through immediately and initiate the delay. When disabled, a trigger only initiates a delay.

Example

```
trigger.timer[1].passthrough = true
```

Enables pass-through mode on trigger timer 1.

Also see

trigger.timer[N].reset() (on page 7-417)

trigger.timer[N].reset()

This function resets some of the trigger timer settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].reset()
```

N	Trigger timer number (1 to 8)

Details

The `trigger.timer[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].passthrough`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

```
trigger.timer[1].reset()
```

Resets the attributes associated with timer 1 back to factory default values.

Also see

trigger.timer[N].count (on page 7-412)
trigger.timer[N].delay (on page 7-412)
trigger.timer[N].delaylist (on page 7-413)
trigger.timer[N].overrun (on page 7-415)
trigger.timer[N].passthrough (on page 7-416)
trigger.timer[N].stimulus (on page 7-418)

trigger.timer[N].stimulus

This attribute specifies which event starts the timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Not saved	0

Usage

```
eventID = trigger.timer[N].stimulus
trigger.timer[N].stimulus = eventID
```

eventID	The event that triggers the timer delay
N	Trigger timer number (1 to 8)

Details

The *eventID* parameter may be one of the trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
smu*X*.trigger.SWEEPING_EVENT_ID	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
smu*X*.trigger.ARMED_EVENT_ID	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
smu*X*.trigger.SOURCE_COMPLETE_EVENT_ID	Occurs when the SMU completes a source action
smu*X*.trigger.MEASURE_COMPLETE_EVENT_ID	Occurs when the SMU completes a measurement action
smu*X*.trigger.PULSE_COMPLETE_EVENT_ID	Occurs when the SMU completes a pulse
smu*X*.trigger.SWEEP_COMPLETE_EVENT_ID	Occurs when the SMU completes a sweep
smu*X*.trigger.IDLE_EVENT_ID	Occurs when the SMU returns to the idle state
digio.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a digital I/O line
tsplink.trigger[*N*].EVENT_ID	Occurs when an edge is detected on a TSP-Link line
lan.trigger[*N*].EVENT_ID	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
display.trigger.EVENT_ID	Occurs when the TRIG key on the front panel is pressed
trigger.EVENT_ID	Occurs when a *TRG command is received on the remote interface GPIB only: Occurs when a GET bus command is received VXI-11 only: Occurs with the VXI-11 command device_trigger; reference the VXI-11 standard for additional details on the device trigger operation
trigger.blender[*N*].EVENT_ID	Occurs after a collection of events is detected
trigger.timer[*N*].EVENT_ID	Occurs when a delay expires

Set this attribute to the *eventID* of any trigger event to cause the timer to start when that event occurs.

Set this attribute to zero (0) to disable event processing.

Example

`print(trigger.timer[1].stimulus)`	Prints the event that starts a trigger 1 timer action.

Also see

trigger.timer[N].reset() (on page 7-417)

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 8)
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.timer[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on timer 3.
If `false` is returned, no trigger was detected during the 10 s timeout.
If `true` is returned, a trigger was detected.

Also see

trigger.timer[N].clear() (on page 7-411)

trigger.wait()

This function waits for a command interface trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
triggered = trigger.wait(timeout)
```

triggered	`true`: A trigger was detected during the timeout period `false`: No triggers were detected during the timeout period
timeout	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A `*TRG` message is received

If one or more of these trigger events were previously detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.wait(10)
print(triggered)
```

Waits up to 10 seconds for a trigger.
If `false` is returned, no trigger was detected during the 10-second timeout.
If `true` is returned, a trigger was detected.

Also see

trigger.clear() (on page 7-410)

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 6-64)

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

Details

After doing a TSP-Link reset (`tsplink.reset()`), use this attribute to access the node number of the master in a set of instruments connected over TSP-Link.

Example

```
LinkMaster = tsplink.master
```

Store the TSP-Link master node number in a variable called LinkMaster.

Also see

tsplink.reset() (on page 7-426)

tsplink.node

This attribute defines the node number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	1

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

nodeNumber	The node number of the instrument or enclosure (1 to 64)

Details

This command sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.reset()` from an earlier TSP-Link instrument is executed on any node in the system.

Each node connected to the TSP-Link system must be assigned a different node number.

Example

```
tsplink.node = 3
```

Sets the TSP-Link node for this instrument to number 3.

Also see

tsplink.reset() (on page 7-426)
tsplink.state (on page 7-427)

tsplink.readbit()

This function reads the state of a TSP-Link synchronization line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readbit(N)
```

data	The state of the synchronization line
N	The trigger line (1 to 3)

Details

Returns a value of zero (0) if the line is low and 1 if the line is high.

Example

```
data = tsplink.readbit(3)
print(data)
```

Assume line 3 is set high, and it is then read.
Output:

```
1.000000e+00
```

Also see

tsplink.readport() (on page 7-425)
tsplink.writebit() (on page 7-439)

tsplink.readport()

This function reads the TSP-Link trigger lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readport()
```

data	Numeric value that indicates which lines are set

Details

The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and the value of bit 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. This indicates that line 2 is high (1), and that the other two lines are low (0).

Example

```
data = tsplink.readport()
print(data)
```

Reads state of all three TSP-Link lines.
Assuming line 2 is set high, the output is:

```
2.000000e+00
```

(binary 010)
The format of the output may vary depending on the ASCII precision setting.

Also see

TSP-Link trigger lines (on page 3-98)
tsplink.readbit() (on page 7-424)
tsplink.writebit() (on page 7-439)
tsplink.writeport() (on page 7-440)

tsplink.reset()

This function initializes (resets) all nodes (instruments) in the TSP-Link system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.reset()
nodesFound = tsplink.reset(expectedNodes)
```

nodesFound	The number of nodes actually found on the system
expectedNodes	The number of nodes expected on the system (1 to 64)

Details

This function erases all information regarding other nodes connected on the TSP-Link system and regenerates the system configuration. This function must be called at least once before any remote nodes can be accessed. If the node number for any instrument is changed, the TSP-Link must be reset again.

If *expectedNodes* is not given, this function generates an error if no other nodes are found on the TSP-Link network.

If *nodesFound* is less than *expectedNodes*, an error is generated. Note that the node on which the command is running is counted as a node. For example, giving an expected node count of 1 does not generate any errors, even if there are no other nodes on the TSP-Link network.

Also returns the number of nodes found.

Example

```
nodesFound = tsplink.reset(2)
print("Nodes found = " .. nodesFound)
```

Perform a TSP-Link reset and indicate how many nodes are found.
Sample output if two nodes are found:

```
Nodes found = 2
```

Sample output if fewer nodes are found and if `localnode.showerrors = 1`:

```
1219, TSP-Link found fewer nodes than expected
Nodes found = 1
```

Also see

localnode.showerrors (on page 7-181)
tsplink.node (on page 7-423)
tsplink.state (on page 7-427)

tsplink.state

This attribute describes the TSP-Link online state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
state = tsplink.state
```

state	TSP-Link state (`online` or `offline`)

Details

When the instrument power is first turned on, the state is `offline`. After `tsplink.reset()` is successful, the state is `online`.

Example

```
state = tsplink.state
print(state)
```

Read the state of the TSP-Link system. If it is online, the output is:

```
online
```

Also see

tsplink.node (on page 7-423)
tsplink.reset() (on page 7-426)

tsplink.trigger[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event ID.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].assert()
```

N	The trigger line (1 to 3)

Details

The set pulse width determines how long the trigger is asserted.

Example

```
tsplink.trigger[2].assert()
```

Asserts trigger on trigger line 2.

Also see

tsplink.trigger[N].clear() (on page 7-429)
tsplink.trigger[N].mode (on page 7-431)
tsplink.trigger[N].overrun (on page 7-433)
tsplink.trigger[N].pulsewidth (on page 7-434)
tsplink.trigger[N].release() (on page 7-435)
tsplink.trigger[N].stimulus (on page 7-437)
tsplink.trigger[N].wait() (on page 7-438)

tsplink.trigger[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].clear()
```

N	The trigger line (1 to 3) to clear

Details

The trigger event detector enters the detected state when an event is detected. `tsplink.trigger[N].clear()` clears a trigger event detector, discards the history of the trigger line, and clears the `tsplink.trigger[N].overrun` attribute.

Example

```
tsplink.trigger[2].clear()
```

Clears trigger event on synchronization line 2.

Also see

tsplink.trigger[N].mode (on page 7-431)
tsplink.trigger[N].overrun (on page 7-433)
tsplink.trigger[N].release() (on page 7-435)
tsplink.trigger[N].stimulus (on page 7-437)
tsplink.trigger[N].wait() (on page 7-438)

tsplink.trigger[N].EVENT_ID

This constant identifies the number that is used for the trigger events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = tsplink.trigger[N].EVENT_ID
```

eventID	The trigger event number
N	The trigger line (1 to 3)

Details

This number is used by the TSP-Link trigger line when it detects an input trigger.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this line.

Example

```
trigger.timer[1].stimulus = tsplink.trigger[2].EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the TSP-Link trigger 2 event.

Also see

None

tsplink.trigger[N].mode

This attribute defines the trigger operation and detection mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup TSP-Link trigger N reset	Not saved	0 (tsplink.TRIG_BYPASS)

Usage

```
mode = tsplink.trigger[N].mode
tsplink.trigger[N].mode = mode
```

mode	The trigger mode
N	The trigger line (1 to 3)

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger line.

The setting for the *mode* parameter can be one of the values shown in the following table.

Mode	Number value	Description
tsplink.TRIG_BYPASS	0	Allows direct control of the line as a digital I/O line.
tsplink.TRIG_FALLING	1	Detects falling-edge triggers as input. Asserts a TTL-low pulse for output.
tsplink.TRIG_RISING	2	If the programmed state of the line is high, the tsplink.TRIG_RISING mode behaves similarly to tsplink.TRIG_RISINGA. If the programmed state of the line is low, the tsplink.TRIG_RISING mode behaves similarly to tsplink.TRIG_RISINGM. Use tsplink.TRIG_RISINGA if the line is in the high output state. Use tsplink.TRIG_RISINGM if the line is in the low output state.
tsplink.TRIG_EITHER	3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
tsplink.TRIG_SYNCHRONOUSA	4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low.
tsplink.TRIG_SYNCHRONOUS	5	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
tsplink.TRIG_SYNCHRONOUSM	6	Detects rising-edge triggers as an input. Asserts a TTL-low pulse for output.
tsplink.TRIG_RISINGA	7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
tsplink.TRIG_RISINGM	8	Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.

When programmed to any mode except tsplink.TRIG_BYPASS, the output state of the I/O line is controlled by the trigger logic and the user-specified output state of the line is ignored.

When the trigger mode is set to `tsplink.TRIG_RISING`, the user-specified output state of the line is examined. If the output state selected when the mode is changed is high, the actual mode that is used is `tsplink.TRIG_RISINGA`. If the output state selected when the mode is changed is low, the actual mode that is used is `tsplink.TRIG_RISINGM`.

The *mode* parameter stores the trigger mode as a numeric value when the attribute is read.

To control the line state, use the `tsplink.TRIG_BYPASS` mode with the `tsplink.writebit()` and the `tsplink.writeport()` commands.

Example

```
tsplink.trigger[3].mode = tsplink.TRIG_RISINGM
```

Sets the trigger mode for synchronization line 3 to `tsplink.TRIG_RISINGM`.

Also see

digio.writebit() (on page 7-69)
digio.writeport() (on page 7-70)
tsplink.trigger[N].assert() (on page 7-428)
tsplink.trigger[N].clear() (on page 7-429)
tsplink.trigger[N].overrun (on page 7-433)
tsplink.trigger[N].release() (on page 7-435)
tsplink.trigger[N].reset() (on page 7-436)
tsplink.trigger[N].stimulus (on page 7-437)
tsplink.trigger[N].wait() (on page 7-438)

tsplink.trigger[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup TSP-Link trigger N clear TSP-Link trigger N reset	Not applicable	Not applicable

Usage

```
overrun = tsplink.trigger[N].overrun
```

overrun	Trigger overrun state
N	The trigger line (1 to 3)

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself.

It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun.

Example

```
print(tsplink.trigger[1].overrun)
```

If an event was ignored, displays `true`; if an event was not ignored, displays `false`.

Also see

tsplink.trigger[N].assert() (on page 7-428)
tsplink.trigger[N].clear() (on page 7-429)
tsplink.trigger[N].mode (on page 7-431)
tsplink.trigger[N].release() (on page 7-435)
tsplink.trigger[N].reset() (on page 7-436)
tsplink.trigger[N].stimulus (on page 7-437)
tsplink.trigger[N].wait() (on page 7-438)

tsplink.trigger[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset TSP-Link trigger N reset Recall setup	Not saved	10e-6 (10 µs)

Usage

```
width = tsplink.trigger[N].pulsewidth
tsplink.trigger[N].pulsewidth = width
```

width	The pulse width (in seconds)
N	The trigger line (1 to 3)

Details

Setting the pulse width to 0 (seconds) asserts the trigger indefinitely.

Example

```
tsplink.trigger[3].pulsewidth = 20e-6
```

Sets pulse width for trigger line 3 to 20 µs.

Also see

tsplink.trigger[N].release() (on page 7-435)

tsplink.trigger[N].release()

This function releases a latched trigger on the given TSP-Link trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].release()
```

N	The trigger line (1 to 3)

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

```
tsplink.trigger[3].release()
```

Releases trigger line 3.

Also see

tsplink.trigger[N].assert() (on page 7-428)
tsplink.trigger[N].clear() (on page 7-429)
tsplink.trigger[N].mode (on page 7-431)
tsplink.trigger[N].overrun (on page 7-433)
tsplink.trigger[N].pulsewidth (on page 7-434)
tsplink.trigger[N].stimulus (on page 7-437)
tsplink.trigger[N].wait() (on page 7-438)

tsplink.trigger[N].reset()

This function resets some of the TSP-Link trigger attributes to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].reset()
```

N	The trigger line (1 to 3)

Details

The `tsplink.trigger[N].reset()` function resets the following attributes to their factory defaults:

- `tsplink.trigger[N].mode`
- `tsplink.trigger[N].stimulus`
- `tsplink.trigger[N].pulsewidth`

This also clears `tsplink.trigger[N].overrun`.

Example

```
tsplink.trigger[3].reset()
```

Resets TSP-Link trigger line 3 attributes back to factory default values.

Also see

tsplink.trigger[N].mode (on page 7-431)
tsplink.trigger[N].overrun (on page 7-433)
tsplink.trigger[N].pulsewidth (on page 7-434)
tsplink.trigger[N].stimulus (on page 7-437)

tsplink.trigger[N].stimulus

This attribute specifies the event that causes the synchronization line to assert a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup TSP-Link trigger N reset	Not saved	0

Usage

```
eventID = tsplink.trigger[N].stimulus
tsplink.trigger[N].stimulus = eventID
```

eventID	The event identifier for the triggering event
N	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to zero (0).

Do not use this attribute when triggering under script control. Use `tsplink.trigger[N].assert()` instead.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

Trigger event IDs*	
Event ID	**Event description**
`smuX.trigger.SWEEPING_EVENT_ID`	Occurs when the source-measure unit (SMU) transitions from the idle state to the arm layer of the trigger model
`smuX.trigger.ARMED_EVENT_ID`	Occurs when the SMU moves from the arm layer to the trigger layer of the trigger model
`smuX.trigger.SOURCE_COMPLETE_EVENT_ID`	Occurs when the SMU completes a source action
`smuX.trigger.MEASURE_COMPLETE_EVENT_ID`	Occurs when the SMU completes a measurement action
`smuX.trigger.PULSE_COMPLETE_EVENT_ID`	Occurs when the SMU completes a pulse
`smuX.trigger.SWEEP_COMPLETE_EVENT_ID`	Occurs when the SMU completes a sweep
`smuX.trigger.IDLE_EVENT_ID`	Occurs when the SMU returns to the idle state
`digio.trigger[N].EVENT_ID`	Occurs when an edge is detected on a digital I/O line
`tsplink.trigger[N].EVENT_ID`	Occurs when an edge is detected on a TSP-Link line
`lan.trigger[N].EVENT_ID`	Occurs when the appropriate LXI trigger packet is received on LAN trigger object *N*
`display.trigger.EVENT_ID`	Occurs when the TRIG key on the front panel is pressed
`trigger.EVENT_ID`	Occurs when a `*TRG` command is received on the remote interface GPIB only: Occurs when a `GET` bus command is received VXI-11 only: Occurs with the VXI-11 command `device_trigger`; reference the VXI-11 standard for additional details on the device trigger operation
`trigger.blender[N].EVENT_ID`	Occurs after a collection of events is detected
`trigger.timer[N].EVENT_ID`	Occurs when a delay expires

Example

```
print(tsplink.trigger[3].stimulus)
```

Prints the event that starts TSP-Link trigger line 3 action.

Also see

tsplink.trigger[N].assert() (on page 7-428)
tsplink.trigger[N].reset() (on page 7-436)

tsplink.trigger[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = tsplink.trigger[*N*].wait(*timeout*)

triggered	Trigger detection indication; set to one of the following values: `true`: A trigger is detected during the timeout period `false`: A trigger is not detected during the timeout period
N	The trigger line (1 to 3)
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events were detected since the last time `tsplink.trigger[N].wait()` or `tsplink.trigger[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = tsplink.trigger[3].wait(10)
print(triggered)
```

Waits up to 10 seconds for a trigger on TSP-Link® line 3.
If `false` is returned, no trigger was detected during the 10-second timeout.
If `true` is returned, a trigger was detected.

Also see

tsplink.trigger[N].clear() (on page 7-429)

tsplink.writebit()

This function sets a TSP-Link trigger line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writebit(N, data)
```

N	The trigger line (1 to 3)
data	The value to write to the bit: ■ Low: `0` ■ High: `1`

Details

Use `tsplink.writebit()` and `tsplink.writeport()` to control the output state of the trigger line when trigger operation is set to `tsplink.TRIG_BYPASS`.

If the output line is write-protected by the `tsplink.writeprotect` attribute, this command is ignored.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

```
tsplink.writebit(3, 0)
```

Sets trigger line 3 low (0).

Also see

tsplink.readbit() (on page 7-424)
tsplink.readport() (on page 7-425)
tsplink.writeport() (on page 7-440)
tsplink.writeprotect (on page 7-441)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writeport(data)
```

data	Value to write to the port (0 to 7)

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other two lines are set low (0).

Write-protected lines are not changed.

Use the `tsplink.writebit()` and `tsplink.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `tsplink.TRIG_BYPASS`.

The `reset()` function does not affect the present states of the trigger lines.

Example

```
tsplink.writeport(3)
```

Sets the synchronization lines 1 and 2 high (binary 011).

Also see

tsplink.readbit() (on page 7-424)
tsplink.writebit() (on page 7-439)
tsplink.writeprotect (on page 7-441)

tsplink.writeprotect

This attribute contains the write-protect mask that protects bits from changes by the `tsplink.writebit()` and `tsplink.writeport()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	0

Usage

```
mask = tsplink.writeprotect
tsplink.writeprotect = mask
```

mask	An integer that specifies the value of the bit pattern for write-protect; set bits to `1` to write-protect the corresponding TSP-Link trigger line

Details

The binary equivalent of *mask* indicates the mask to be set for the TSP-Link trigger line. For example, a *mask* value of `5` has a binary equivalent of 101. This *mask* write-protects TSP-Link trigger lines 1 and 3.

Example

```
tsplink.writeprotect = 5
```

Write-protects TSP-Link trigger lines 1 and 3.

Also see

Controlling digital I/O lines (on page 3-94)
tsplink.readbit() (on page 7-424)
tsplink.readport() (on page 7-425)
tsplink.writebit() (on page 7-439)
tsplink.writeport() (on page 7-440)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.clear(connectionID)
```

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

```
tspnet.write(testdevice, "print([[hello]])")
print(tspnet.readavailable(testdevice))
tspnet.clear(testdevice)
print(tspnet.readavailable(testdevice))
```

Write data to a device, then print how much is available.
Output:

```
6.00000e+00
```

Clear data and print how much data is available again.
Output:

```
0.00000e+00
```

Also see

tspnet.connect() (on page 7-443)
tspnet.readavailable() (on page 7-448)
tspnet.write() (on page 7-455)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
connectionID = tspnet.connect("ipAddress")
connectionID = tspnet.connect("ipAddress", portNumber, "initString")
```

connectionID	The connection ID to be used as a handle in all other `tspnet` function calls
ipAddress	IP address to which to connect in a string; accepts IP address or host name when trying to connect
portNumber	Port number (default `5025`)
initString	Initialization string to send to *ipAddress*

Details

This command connects a device to another device through the LAN interface. If the *portNumber* is 23, the interface uses the Telnet protocol and sets appropriate termination characters to communicate with the device.

If a *portNumber* and *initString* are provided, it is assumed that the remote device is not TSP-enabled. The Model 2651A 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 Model 2651A sends an `abort` command to the remote device on connection.

The Model 2651A also enables TSP prompts on the remote device and error management. The Model 2651A places remote errors from the TSP-enabled device in its own error queue and prefaces these errors with `Remote Error`, followed by an error description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show errors by changing `localnode.showerrors` on the remote TSP-enabled device. 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 7-177)
localnode.showerrors (on page 7-181)
tspnet.tsp.abortonconnect (on page 7-452)
tspnet.disconnect() (on page 7-444)

tspnet.disconnect()

This function disconnects a specified TSP-Net session.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.disconnect(`*`connectionID`*`)`

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function disconnects the two devices by closing the connection. The *connectionID* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

```
testID = tspnet.connect("192.0.2.0")
-- Use the connection
tspnet.disconnect(testID)
```

Create a TSP-Net session.
Close the session.

Also see

tspnet.connect() (on page 7-443)

tspnet.execute()

This function sends a command string to the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.execute("connectionID", "commandString")
value1 = tspnet.execute("connectionID", "commandString", formatString)
value1, value2 = tspnet.execute("connectionID", "commandString", formatString)
value1, ..., valueN = tspnet.execute("connectionID", "commandString", formatString)
```

connectionID	The connection ID returned from `tspnet.connect()`
commandString	The command to send to the remote device
value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The *N*th value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
formatString	Format string for the output

Details

This command sends a command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (`tspnet.termination()`). You can also specify a format string, which causes the command to wait for a response from the remote instrument. The Model 2651A 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 Model 2651A 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 Model 2651A places any remotely generated errors into its error queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

```
tspnet.execute(instrumentID, "runScript()")
```

Command the remote device to run a script named `runScript`.

Example 2

```
tspnet.timeout = 5
id_instr = tspnet.connect("192.0.2.23", 23, "*rst\r\n")
tspnet.termination(id_instr, tspnet.TERM_CRLF)
tspnet.execute(id_instr, "*idn?")
print("tspnet.execute returns:", tspnet.read(id_instr))
```

Print the `*idn?` string from the remote device.

Also see

tspnet.connect() (on page 7-443)
tspnet.read() (on page 7-447)
tspnet.termination() (on page 7-449)
tspnet.write() (on page 7-455)

tspnet.idn()

This function retrieves the response of the remote device to `*IDN?`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

idnString = tspnet.idn(*connectionID*)

idnString	The returned `*IDN?` string
connectionID	The connection ID returned from `tspnet.connect()`

Details

This function retrieves the response of the remote device to `*IDN?`.

Example

```
deviceID = tspnet.connect("192.0.2.1")
print(tspnet.idn(deviceID))
tspnet.disconnect(deviceID)
```

Assume the instrument is at IP address 192.0.2.1.
The output that is produced when you connect to the instrument and read the identification string may appear as:

```
Keithley Instruments, Model 2651A, 00000170,
    1.1.0
```

Also see

tspnet.connect() (on page 7-443)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valueN = tspnet.read(connectionID, formatString)
```

value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The nth value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
connectionID	The connection ID returned from `tspnet.connect()`
formatString	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following specifiers:

`%[width]s`	Read data until the specified length
`%[max width]t`	Read data until the specified length or until punctuation is found, whichever comes first
`%[max width]n`	Read data until a newline or carriage return
`%d`	Read a number (delimited by punctuation)

A maximum of 10 format specifiers can be used for a maximum of 10 return values.

If *formatString* is not provided, the command returns a string that contains the data until a new line is reached. If no data is available, the Model 2651A pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When the Model 2651A reads from a TSP-enabled remote instrument, the Model 2651A removes Test Script Processor (TSP®) prompts and places any errors it receives from the remote instrument into its own error queue. The Model 2651A prefaces errors from the remote device with "`Remote Error`," followed by the error number and error description.

Example

```
tspnet.write(deviceID, "*idn?\r\n")

print("write/read returns:", tspnet.read(deviceID))
```

Send the "*idn?\r\n" message to the instrument connected as deviceID.
Display the response that is read from deviceID (based on the *idn? message).

Also see

tspnet.connect() (on page 7-443)
tspnet.readavailable() (on page 7-448)
tspnet.timeout (on page 7-450)
tspnet.write() (on page 7-455)

tspnet.readavailable()

This function checks to see if data is available from the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
bytesAvailable = tspnet.readavailable(connectionID)
```

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from tspnet.connect()

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")
repeat bytes = tspnet.readavailable(ID) until bytes > 0
print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that create data.
Wait for data to be available.

Also see

tspnet.connect() (on page 7-443)
tspnet.read() (on page 7-447)

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	An enumerated value indicating the termination type: ■ 1 or tspnet.TERM_LF ■ 4 or tspnet.TERM_CR ■ 2 or tspnet.TERM_CRLF ■ 3 or tspnet.TERM_LFCR
connectionID	The connection ID returned from tspnet.connect()
termSequence	The termination sequence

Details

This function sets and gets the termination character sequence that is used to indicate the end of a line for a TSP-Net connection.

Using the *termSequence* parameter sets the termination sequence. The present termination sequence is always returned.

For the *termSequence* parameter, use the same values listed in the table above for type. There are four possible combinations, all of which are made up of line feeds (LF or 0x10) and carriage returns (CR or 0x13). For TSP-enabled devices, the default is tspnet.TERM_LF. For devices that are not TSP-enabled, the default is tspnet.TERM_CRLF.

Example

```
deviceID = tspnet.connect("192.0.2.1")
if deviceID then
   tspnet.termination(deviceID, tspnet.TERM_LF)
end
```

Sets termination type for IP address 192.0.2.1 to `TERM_LF`.

Also see

tspnet.connect() (on page 7-443)
tspnet.disconnect() (on page 7-444)

tspnet.timeout

This attribute sets the timeout value for the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset Recall setup	Not saved	20.0 (20 s)

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

value	The timeout duration in seconds (1 ms to 30.0 s)

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution but is only accurate to the nearest 10 ms.

Example

```
tspnet.timeout = 2.0
```

Sets the timeout duration to 2 s.

Also see

tspnet.connect() (on page 7-443)
tspnet.execute() (on page 7-445)
tspnet.read() (on page 7-447)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.abort(connectionID)
```

connectionID	Integer value used as a handle for other `tspnet` commands

Details

This function is appropriate only for TSP-enabled instruments.

Sends an abort command to the remote instrument.

Example

```
tspnet.tsp.abort(testConnection)
```

Stops remote instrument execution on `testConnection`.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset Recall setup	Not saved	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	■ Enable: `1` ■ Disable: `0`

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument does not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

```
tspnet.tsp.abortonconnect = 0
```

Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

tspnet.connect() (on page 7-443)

tspnet.tsp.rbtablecopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
table = tspnet.tsp.rbtablecopy(connectionID, "name")
table = tspnet.tsp.rbtablecopy(connectionID, "name", startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other `tspnet` commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The *startIndex* and *endIndex* parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

```
t = tspnet.tsp.rbtablecopy(testConnection, "testRemotebuffername.readings", 1, 3)
print(t[1], t[2], t[3])
```

Copy the specified readings table for buffer items 1 through 3, then display the first three readings.
Example output:

```
4.56534e-01
4.52675e-01
4.57535e-01
```

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.tsp.runscript(connectionID, "script")
tspnet.tsp.runscript(connectionID, "name", "script")
```

connectionID	Integer value used as an identifier for other `tspnet` commands
name	The name that is assigned to the script
script	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions later.

If no name is specified, the script is loaded as the anonymous script.

Example

```
tspnet.tsp.runscript(myconnection, "mytest",
"print([[start]]) for d = 1, 10 do print([[work]]) end print([[end]])")
```

Load and run a script entitled `mytest` on the TSP-enabled instrument connected with `myconnection`.

Also see

tspnet.execute() (on page 7-445)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.write(connectionID, "inputString")
```

connectionID	The connection ID returned from `tspnet.connect()`
inputString	The string to be written

Details

The `tspnet.write()` function sends *inputString* to the remote instrument. It does not wait for command completion on the remote instrument.

The Model 2651A sends *inputString* to the remote instrument exactly as indicated. The *inputString* must contain any necessary new lines, termination, or other syntax elements needed to complete properly.

Because `tspnet.write()` does not process output from the remote instrument, do not send commands that generate too much output without processing the output. This command can stop executing if there is too much unprocessed output from previous commands.

Example

```
tspnet.write(myID, "runscript()\r\n")
```

Commands the remote instrument to execute a command or script named `runscript()` on a remote device identified in the system as `myID`.

Also see

tspnet.connect() (on page 7-443)
tspnet.read() (on page 7-447)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.add("name", "value")
```

name	The name of the string; the key of the key-value pair
value	The string to associate with *name*; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

Example

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.catalog() do
   print(name .. " = " ..
      userstring.get(name))
end
```

Stores user-defined strings in nonvolatile memory and recalls them from the instrument using a for loop.
Example output:

```
assetnumber = 236
contact = John Doe
product = Widgets
```

Also see

userstring.catalog() (on page 7-457)
userstring.delete() (on page 7-458)
userstring.get() (on page 7-458)

userstring.catalog()

This function creates an iterator for the user-defined string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do body end
```

name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The catalog provides access for user-defined string pairs, allowing you to manipulate all the key-value pairs in nonvolatile memory. The entries are enumerated in no particular order.

Example 1

```
for name in userstring.catalog() do
   userstring.delete(name)
end
```

Deletes all user-defined strings in nonvolatile memory.

Example 2

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.catalog() do
   print(name .. " = " ..
      userstring.get(name))
end
```

Prints all `userstring` key-value pairs.

Output:

```
product = Widgets
assetnumber = 236
contact = John Doe
```

Notice the key-value pairs are not listed in the order they were added.

Also see

userstring.add() (on page 7-456)
userstring.delete() (on page 7-458)
userstring.get() (on page 7-458)

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 7-456)
userstring.catalog() (on page 7-457)
userstring.get() (on page 7-458)

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 7-456)
userstring.catalog() (on page 7-457)
userstring.delete() (on page 7-458)

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 group.

Example 1

```
waitcomplete()
```

Waits for all nodes in the local group.

Example 2

```
waitcomplete(G)
```

Waits for all nodes in group G.

Example 3

```
waitcomplete(0)
```

Waits for all nodes on the TSP-Link network.

Also see

None

Section 8

Troubleshooting guide

In this section:

Introduction

Troubleshooting information includes information on the Keithley Instruments Model 2651A errors (including a complete list of error messages) and LAN troubleshooting suggestions.

Error levels

Error messages are listed in Error summary list (on page 8-3). Errors have one of the following error levels:

Number	Error level	Description
0	NO_SEVERITY	The message is information only. This level is used when the error queue is empty; the message does not represent an error.
10	INFORMATIONAL	The message is information only. This level is used to indicate status changes; the message does not represent an error.
20	RECOVERABLE	The error was caused by improper use of the instrument or by conditions that can be corrected. This message indicates that an error occurred. The instrument is still operating normally.
30	SERIOUS	There is a condition that prevents the instrument from functioning properly. The message indicates that the instrument is presently operating in an error condition. If the condition is corrected, the instrument returns to normal operation.
40	FATAL	There is a condition that cannot be corrected that prevents the instrument from functioning properly. Disconnect the DUT and turn the power off and then on again. If the error is a hardware fault that persists after cycling the power, the instrument must be repaired.

Effects of errors on scripts

Most errors do not abort a running script. The only time a script is aborted is when a Lua runtime error (error code −286, "TSP runtime error") is detected. Runtime errors are caused by actions such as trying to index into a variable that is not a table.

Syntax errors (error code −285, "Program syntax") in a script or command prevent execution of the script or command.

Retrieving errors

When errors occur, the error messages are placed in the error queue. Use `errorqueue` commands to request error message information. For example, the following commands request the complete set of information about the next message in the error queue. They return the code, message, severity, and node for that error:

```
errorCode, message, severity, errorNode = errorqueue.next()
print(errorCode, message, severity, errorNode)
```

The following table lists the commands associated with the error queue.

Remote commands associated with the error queue

Command	Description
errorqueue.clear() (on page 7-98)	Clear error queue of all errors
errorqueue.count (on page 7-98)	Number of messages in the error queue
errorqueue.next() (on page 7-99)	Request next error message from queue

Error summary list

Error summary

Error number	Error level	Error message
-430	RECOVERABLE	Query DEADLOCKED
-420	RECOVERABLE	Query UNTERMINATED
-410	RECOVERABLE	Query INTERRUPTED
-363	RECOVERABLE	Input buffer overrun
-350	RECOVERABLE	Queue overflow
-315	RECOVERABLE	Configuration memory lost
-314	RECOVERABLE	Save/recall memory lost
-292	RECOVERABLE	Referenced name does not exist
-286	RECOVERABLE	TSP Runtime error
-285	RECOVERABLE	Program syntax
-282	RECOVERABLE	Illegal program name
-281	RECOVERABLE	Cannot create program
-225	RECOVERABLE	Out of memory or TSP Memory allocation error
-224	RECOVERABLE	Illegal parameter value
-223	RECOVERABLE	Too much data
-222	RECOVERABLE	Parameter data out of range
-221	RECOVERABLE	Settings conflict
-220	RECOVERABLE	Parameter error
-211	RECOVERABLE	Trigger ignored
-203	RECOVERABLE	Command protected
-154	RECOVERABLE	String too long
-151	RECOVERABLE	Invalid string data
-203	RECOVERABLE	Command protected
-154	RECOVERABLE	String too long
-151	RECOVERABLE	Invalid string data
-144	RECOVERABLE	Character data too long
-141	RECOVERABLE	Invalid character data
-121	RECOVERABLE	Invalid character in number
-120	RECOVERABLE	Numeric data
-109	RECOVERABLE	Missing parameter
-108	RECOVERABLE	Parameter not allowed
-105	RECOVERABLE	Trigger not allowed
-104	RECOVERABLE	Data type error
-101	RECOVERABLE	Invalid character
0	NO_SEVERITY	Queue Is Empty
603	RECOVERABLE	Power on state lost
702	FATAL	Unresponsive digital FPGA
802	RECOVERABLE	OUTPUT blocked by interlock
820	RECOVERABLE	Error parsing value
900	FATAL	Internal system error
1100	RECOVERABLE	Command unavailable
1101	RECOVERABLE	Parameter too big
1102	RECOVERABLE	Parameter too small
1103	RECOVERABLE	Min greater than max
1104	RECOVERABLE	Too many digits for param type

Error summary

Error number	Error level	Error message
1106	RECOVERABLE	Battery not present
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist
1109	RECOVERABLE	Menu name already exists
1110	FATAL	Catastrophic analog supply failure
1122	SERIOUS	Interlock or power supply failure
1200	RECOVERABLE	TSP-Link initialization failed
1201	RECOVERABLE	TSP-Link initialization failed
1202	RECOVERABLE	TSP-Link initialization failed
1203	RECOVERABLE	TSP-Link initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSP-Link initialization failed
1205	RECOVERABLE	TSP-Link initialization failed (no remote nodes found)
1206	RECOVERABLE	TSP-Link initialization failed
1207	RECOVERABLE	TSP-Link initialization failed
1208	RECOVERABLE	TSP-Link initialization failed
1209	RECOVERABLE	TSP-Link initialization failed
1210	RECOVERABLE	TSP-Link initialization failed (node ID conflict)
1211	RECOVERABLE	Node NN is inaccessible
1212	RECOVERABLE	Invalid node ID
1400	RECOVERABLE	Expected at least NN parameters
1401	RECOVERABLE	Parameter NN is invalid
1402	RECOVERABLE	User scripts lost
1403	RECOVERABLE	Factory scripts lost
1404	RECOVERABLE	Invalid byte order
1405	RECOVERABLE	Invalid ASCII precision
1406	RECOVERABLE	Invalid data format
1500	RECOVERABLE	Invalid baud rate setting
1501	RECOVERABLE	Invalid parity setting
1502	RECOVERABLE	Invalid terminator setting
1503	RECOVERABLE	Invalid bits setting

Error summary

Error number	Error level	Error message
1504	RECOVERABLE	Invalid flow control setting
1600	RECOVERABLE	Maximum GPIB message length exceeded
1700	RECOVERABLE	Display area boundary exceeded
1800	RECOVERABLE	Invalid digital trigger mode
1801	RECOVERABLE	Invalid digital I/O line
2000	SERIOUS	Flash download error
2001	RECOVERABLE	Cannot flash with error in queue
2100	FATAL	Could not open socket
2101	FATAL	Could not close socket
2102	RECOVERABLE	Lan configuration already in progress
2103	RECOVERABLE	Lan disabled
2104	RECOVERABLE	Socket error
2105	RECOVERABLE	Unreachable gateway
2106	RECOVERABLE	Could not acquire ip address
2107	RECOVERABLE	Duplicate IP address detected
2108	RECOVERABLE	DHCP lease lost
2109	RECOVERABLE	Lan cable disconnected
2110	RECOVERABLE	Could not resolve hostname
2111	RECOVERABLE	DNS name (FQDN) too long
2112	RECOVERABLE	Connection not established
2200	RECOVERABLE	File write error
2201	RECOVERABLE	File read error
2202	RECOVERABLE	Cannot close file
2203	RECOVERABLE	Cannot open file
2204	RECOVERABLE	Directory not found
2205	RECOVERABLE	File not found
2206	RECOVERABLE	Cannot read current working directory
2207	RECOVERABLE	Cannot change directory
2208	RECOVERABLE	Cannot create directory
2209	RECOVERABLE	Cannot remove directory
2210	RECOVERABLE	File is not a valid script format
2211	RECOVERABLE	File system error
2212	RECOVERABLE	File system command not supported
2213	RECOVERABLE	Too many open files
2214	RECOVERABLE	File access denied
2215	RECOVERABLE	Invalid file handle
2216	RECOVERABLE	Invalid drive
2217	RECOVERABLE	File system busy
2218	RECOVERABLE	Disk full
2219	RECOVERABLE	File corrupt
2220	RECOVERABLE	File already exists
2221	RECOVERABLE	File seek error
2222	RECOVERABLE	End-of-file error
2223	RECOVERABLE	Directory not empty
2401	RECOVERABLE	Invalid specified connection

Error summary

Error number	Error level	Error message
2402	RECOVERABLE	TSPnet remote error: %s, where %s explains the remote error
2403	RECOVERABLE	TSPnet failure
2404	RECOVERABLE	TSPnet read failure
2405	RECOVERABLE	TSPnet read failure, aborted
2406	RECOVERABLE	TSPnet read failure, timeout
2407	RECOVERABLE	TSPnet write failure
2408	RECOVERABLE	TSPnet write failure, aborted
2409	RECOVERABLE	TSPnet write failure, timeout
2410	RECOVERABLE	TSPnet max connections reached
2411	RECOVERABLE	TSPnet connection failed
2412	RECOVERABLE	TSPnet invalid termination
2413	RECOVERABLE	TSPnet invalid reading buffer table
2414	RECOVERABLE	TSPnet invalid reading buffer index range
2415	RECOVERABLE	TSPnet feature only supported on TSP connections
2416	RECOVERABLE	TSPnet must specify both port and init
2417	RECOVERABLE	TSPnet disconnected by other side
4900	RECOVERABLE	Reading buffer index NN is invalid
4901	RECOVERABLE	The maximum index for this buffer is %d
4903	RECOVERABLE	Reading buffer expired
4904	SERIOUS	ICX parameter count mismatch, %s (Line #%d)
4905	SERIOUS	ICX parameter invalid value, %s (Line #%d)
4906	SERIOUS	ICX invalid function id, %s (Line #%d)
5001	FATAL	SMU is unresponsive. Disconnect DUT and cycle power
5003	SERIOUS	Saved calibration constants corrupted
5004	RECOVERABLE	Operation conflicts with CALA sense mode
5005	RECOVERABLE	Value too big for range
5007	RECOVERABLE	Operation would exceed safe operating area of the instrument
5008	RECOVERABLE	Operation not permitted while OUTPUT is on
5009	SERIOUS	Unknown sourcing function
5010	SERIOUS	No such SMU function
5011	RECOVERABLE	Operation not permitted while cal is locked
5012	RECOVERABLE	Cal data not saved - save or restore before lock
5013	RECOVERABLE	Cannot save cal data - unlock before save
5014	RECOVERABLE	Cannot restore cal data - unlock before restore
5015	RECOVERABLE	Save to cal set disallowed
5016	RECOVERABLE	Cannot change cal date - unlock before operation
5017	RECOVERABLE	Cannot change cal constants - unlock before operation
5018	SERIOUS	Cal version inconsistency
5019	RECOVERABLE	Cannot unlock - invalid password
5021	SERIOUS	Cannot restore default calset. Using previous calset
5022	SERIOUS	Cannot restore previous calset. Using factory calset
5023	SERIOUS	Cannot restore factory calset. Using nominal calset
5024	SERIOUS	Cannot restore nominal calset. Using firmware defaults
5025	RECOVERABLE	Cannot set filter.count > 1 when measure.count > 1
5027	RECOVERABLE	Unlock cal data with factory password
5028	RECOVERABLE	Cannot perform requested operation while source autorange is enabled
5029	RECOVERABLE	Cannot save without changing cal adjustment date
5032	RECOVERABLE	Cannot change this setting unless buffer is cleared

Error summary

Error number	Error level	Error message
5033	RECOVERABLE	Reading buffer not found within device
5038	RECOVERABLE	Index exceeds maximum reading
5040	RECOVERABLE	Cannot use same reading buffer for multiple overlapped measurements
5041	SERIOUS	Output Enable not asserted
5042	RECOVERABLE	Cannot perform requested action while an overlapped operation is in progress
5043	RECOVERABLE	Cannot perform requested operation while voltage measure autorange is enabled
5044	RECOVERABLE	Cannot perform requested operation while current measure autorange is enabled
5045	RECOVERABLE	Cannot perform requested operation while filter is enabled
5046	SERIOUS	SMU too hot
5047	RECOVERABLE	Minimum timestamp resolution is 1us
5048	RECOVERABLE	Contact check not valid with HIGH-Z OUTPUT off
5049	RECOVERABLE	Contact check not valid while an active current source
5050	RECOVERABLE	I limit too low for contact check
5051	FATAL	Model number/SMU hardware mismatch. Disconnect DUT and cycle power
5052	RECOVERABLE	Interlock engaged; system stabilizing
5053	RECOVERABLE	Unstable output detected - Measurements may not be valid
5055	RECOVERABLE	Cannot change adjustment date - change cal constants before operation
5059	RECOVERABLE	trigger.source.action enabled without configuration
5060	RECOVERABLE	trigger.measure.action enabled without configuration
5061	RECOVERABLE	Operation not permitted while OUTPUT is off
5062	SERIOUS	SMU overload. Automatic OUTPUT off.
5063	RECOVERABLE	Cannot perform requested operation while measure autozero is on
5064	RECOVERABLE	Cannot use reading buffer that collects source values
5065	RECOVERABLE	I range too low for contact check
5066	RECOVERABLE	source.offlimiti too low for contact check
5067	FATAL	Hardware communication error. Disconnect DUT and cycle power
5069	SERIOUS	Autorange locked for HighC mode

LAN troubleshooting suggestions

If you are unable to connect to the web interface of the instrument, check the following items:

- Verify that the network cable is in the LAN port on the rear panel of the instrument, not one of the TSP-Link® ports (see the description in Rear panel (on page 2-6)).
- Verify that the network cable is in the correct port on the computer. The LAN port of a laptop may be disabled when the laptop is in a docking station.
- Verify that the configuration information for the correct ethernet card was used during the setup procedure.
- Verify that the network card in the computer is enabled.
- Verify that the IP address of the instrument is compatible with the IP address on the computer.
- Verify that the subnet mask address of the instrument is the same as the subnet mask address of the computer.
- Turn the instrument power off, and then on. Wait at least 60 seconds for the network configuration to be completed. Verify that an IP address has been assigned to the instrument:
  1. Press the **MENU** key to display the MAIN MENU.
  2. Use the navigation wheel to select **LAN**. The LAN CONFIG menu is displayed.
  3. Select **STATUS**.
  4. Select **IP-ADDRESS**.
- Restart your computer.
- For more detail on LAN settings, see Connecting to the LAN (on page 13-6).

If the above actions do not correct the problem, contact your system administrator.

Section 9

Frequently asked questions

In this section:

How do I display the instrument's serial number?

The instrument serial number is on a label on the rear panel of the instrument. You can also access the serial number from the front panel using the front-panel keys and menus.

To display the serial number on the front panel:

1. If the Model 2651A is in remote operation, press the **EXIT (LOCAL)** key once to place the instrument in local operation.
2. Press the **MENU** key.
3. Use the navigation wheel to scroll to the **SYSTEM-INFO** menu item.
4. Press the **ENTER** key. The SYSTEM INFORMATION menu is displayed.
5. Scroll to the **SERIAL#** menu item.
6. Press the **ENTER** key. The Model 2651A serial number is displayed.

How do I optimize performance?

The primary factors that affect measurement accuracy and speed are:

- **Warm-up:** For rated measurement accuracy, allow the Model 2651A to warm up for at least two hours before use.
- **Speed setting:** The speed setting affects both speed and accuracy. For more information, see Speed.
- **Autozero:** Autozero can be disabled to increase speed at the expense of accuracy (for more information, see Disabling autozero to increase speed (on page 9-2)).

Disabling autozero to increase speed

Disabling autozero (setting it to OFF) can increase measurement speed. If autozero is disabled, accuracy drifts with time and temperature.

NOTE

Turning autozero OFF disables the autozero function and possibly increases measurement speed. To minimize drift, setting autozero to ONCE performs an autozero operation one time (when it is selected), and then disables the autozero function. For a more detailed discussion of autozero, see Autozero (on page 2-25).

To configure autozero from the front panel:

1. Press the **CONFIG** key, and then select **MEAS** from the menu.
2. Select **AUTO-ZERO**, and then press the **ENTER** key or the navigation wheel.
3. Select the mode (**OFF**, **ONCE**, or **AUTO**), and then press the **ENTER** key or the navigation wheel.
4. Press the **EXIT (LOCAL)** key to the normal display.

Refer to Remote command autozero (on page 2-26) for details about configuring autozero from a remote interface.

How do I upgrade the firmware?

For information on upgrading the firmware, see Upgrading the firmware (on page 11-4).

How do I use the digital I/O port?

You can use the Model 2651A digital input/output with the trigger model or to control an external digital circuit, such as a device handler used to perform binning operations. To control or configure any of the six digital I/O lines, send commands to the Model 2651A over a remote interface.

Use a cable equipped with a male DB-25 connector (L-com part number CSMN25MF-5) to connect the digital I/O port to other Keithley Instruments models equipped with a Trigger Link (TLINK).

For more information about the Model 2651A digital I/O port, see Digital I/O (on page 3-92).

How do I trigger other instruments?

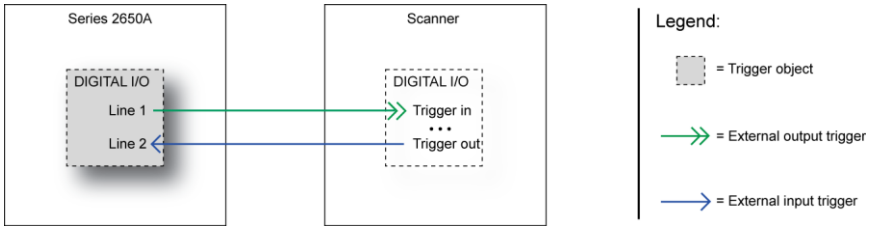
You can use the Model 2651A digital input/output to control an external digital circuit, such as a device handler used to perform binning operations. For more information about the Model 2651A digital I/O port, see Digital I/O (on page 3-92).

Another option is the Keithley Instruments TSP-Link® interface, a high-speed trigger synchronization and communication bus that you can use to connect multiple instruments in a master and subordinate configuration. See TSP-Link System Expansion Interface (on page 6-56) for additional information.

Triggering a scanner

A typical test scenario might call for using the Model 2651A with a scanner to test a number of devices under test (DUTs) in sequence. A basic example of this uses the Model 2651A digital I/O port to trigger a scanner (shown in the figure below). In this example, line 1 of the digital I/O port is used as a trigger output and connected to the scanner mainframe trigger input, and line 2 of the digital I/O port is used as a trigger input.

Figure 124: Triggering a scanner



Interactive trigger programming

The programming example below illustrates how to set up interactive triggering. The example sets the output trigger pulse width on line 1, then programs both lines 1 and 2 for falling edge triggers. Digital I/O line 1 trigger asserts, and then line 2 waits for the input trigger up to the timeout period specified.

```
-- Set line 1 pulse width to 10 us.
digio.trigger[1].pulsewidth = 10e-6
-- Set line 1 mode to falling edge.
digio.trigger[1].mode = digio.TRIG_FALLING
-- Set line 2 mode to falling edge.
digio.trigger[2].mode = digio.TRIG_FALLING
-- Assert trigger on line 1.
digio.trigger[1].assert()
-- When complete, wait for trigger on line 2.
digio.trigger[2].wait(2)
```

More information about triggering

To obtain precise timing and synchronization between instruments, use the remote trigger model. For more information about the remote trigger model and interactive triggering using other trigger objects, see Triggering (on page 3-36).

How do I generate a GPIB service request?

NOTE

For detailed information about this topic, see Status model (on page 15-1).

Setting up a service request

The exact programming steps necessary to generate a GPIB service request (SRQ) vary depending on the events intended to generate the SRQ. In general, these steps are:

1. Clear all status registers to prevent anomalous events from generating an SRQ.
2. Set the appropriate bits in the appropriate status model enable registers.
3. Set the proper bits in the service request enable register. At least one bit in this register must always be set, but the exact bits to be set depend on the desired SRQ events.

Service request programming example

The example below shows how to program the Model 2651A to generate a service request (SRQ) when the current limit is exceeded.

```
-- Clear all registers.
status.reset()
-- Enable the current limit bit in the current limit register.
status.measurement.current_limit.enable = status.measurement.current_limit.SMUA
-- Enable the status measure current limit bit.
status.measurement.enable = status.measurement.ILMT
-- Enable the status SRQ MSB.
status.request_enable = status.MSB
```

Polling for SRQs

To determine if the Model 2651A is the GPIB device that generated the service request (SRQ), serial poll the instrument for the status byte, and test to see if the corresponding summary bits are set.

How do I store measurements in nonvolatile memory?

After the measurements are complete, you can save the reading buffer data to the nonvolatile memory in the instrument.

To save the reading buffer data:

1. From the front panel, press the **STORE** key, and then select **SAVE**.
2. Select **INTERNAL** to save to internal nonvolatile memory.
3. Select one of the following:
   - **SMUA_BUFFER1**
   - **SMUA_BUFFER2**
4. The front panel displays `Saving... This may take awhile`.
5. Press the **EXIT (LOCAL)** key to return to the main menu.

For additional information, see Saving reading buffers (on page 3-9).

When should I change the output-off state?

CAUTION

Carefully consider and configure the appropriate output-off state, source function, and compliance limits before connecting the Model 2651A to a device that can deliver energy (for example, other voltage sources, batteries, capacitors, solar cells, or other Model 2651A instruments). Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and compliance limits may result in damage to the instrument or to the device under test (DUT).

The Model 2651A instrument provides multiple output-off states. The multiple states are required because different types of connected devices (or loads) require different behaviors from the Model 2651A when its output is turned off. Therefore, careful selection of the proper output-off state is important to prevent damage to devices and instruments. This is especially true when the device can deliver energy to the Model 2651A, such as a battery or capacitor or when another SourceMeter instrument is connected across the output terminals. In these situations, you should use an output-off state that isolates the instrument from the device by either setting `smuX.source.offfunc = smuX.OUTPUT_DCAMPS` or `smuX.source.offfunc = smuX.OUTPUT_DCVOLTS`, as applicable.

For example, a passive device such as a diode is not affected by a 0 V source connected across its terminals when the output is turned off. However, connecting a 0 V source to the terminals of a battery causes the battery to discharge.

There are other guidelines to follow when connecting the output of multiple Model 2651A instruments to get a larger current or voltage. Refer to the following references for more information:

- Combining SMU outputs (on page 2-51)
- User's Manual section, "Increasing SMU current sourcing ability"
- Keithley application notes on tek.com/keithley

How do I make contact check measurements?

For information about making contact check measurements, see Contact check measurements (on page 2-40) and Contact check (on page 4-23).

How do I make low-current measurements?

Low-current measurements (<1 mA) are subject to errors caused by leakage currents and leakage resistances in the signal path. Model 2651A instruments are equipped with triaxial connectors to minimize these problems. To assure accurate low-level measurements, the integrity of the signal path must be maintained to the device under test (DUT), including using both low-noise triaxial cables and a suitable test fixture.

Low-current connections

The figure below shows typical connections for low-current measurements. The DUT in this example could be a low-current semiconductor device, a high-megohm resistor, or any other passive or active electronic device requiring low-current measurements. Note that the DUT is enclosed in both a guard shield and a safety shield, which is necessary when hazardous voltages are used.

The inner shield (guard) of the HI triaxial cable is connected to the test fixture guard shield. The guard shield prevents leakage currents from affecting the measurements. The outer cable shield (chassis ground or protective earth (safety ground)) is connected to the safety shield.

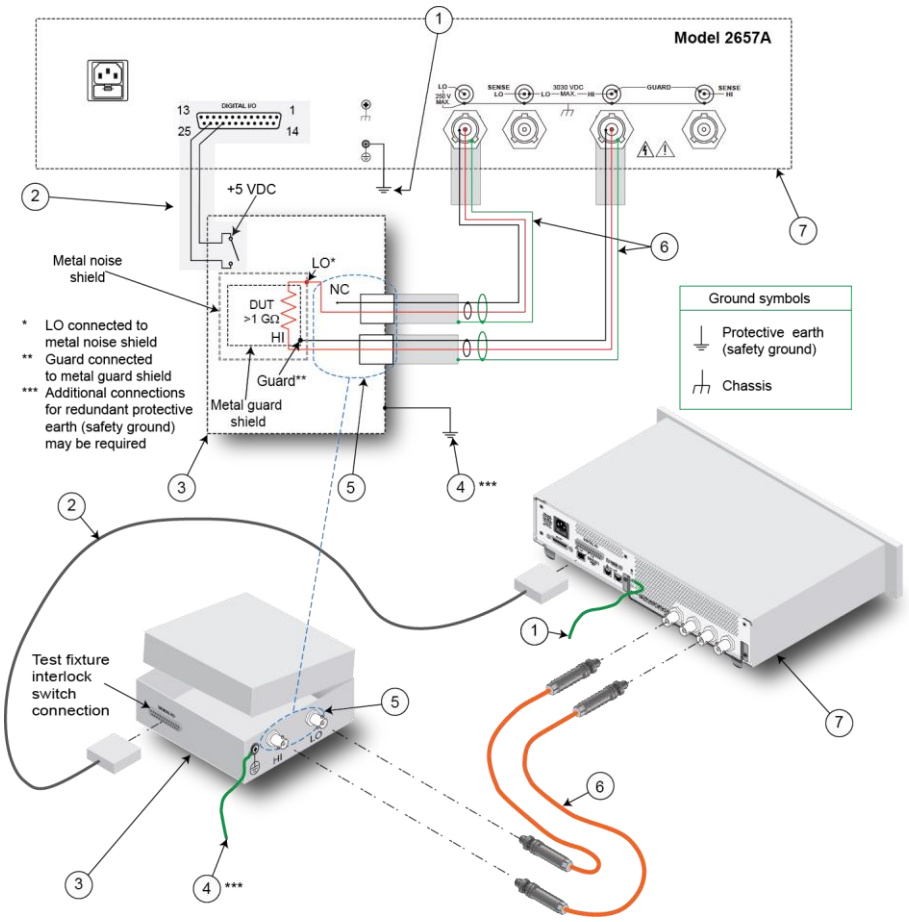
WARNING

A safety shield must be used whenever hazardous voltages (>30 V_{RMS}, 42 V_{PEAK}) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the Model 2651A in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield.

Connect the enclosure of all metal test fixtures to protective earth (safety ground). Nonconductive test fixtures must be rated to double the maximum capability of the test equipment in the system. Failure to attach the ground wires to a known protective earth may result in electric shock.

Additional connections for redundant protective earth (safety ground) may be required.

Figure 125: Typical low-current connections



(1)	Protective earth (safety ground). Keithley Instruments Model CA-568-120 is a protective earth cable assembly. See tek.com/keithley for ordering information.
(2)	Model 2651A interlock digital I/O. Pin 24 (INT) and pin 22 (5 Vdc) are connected to the test fixture lid switch. The interlock switch is shown in the disengaged, or lid open, position.
(3)	Normally-open (NO) interlock metal safety enclosure.
(4)	To protective earth (safety ground) from the test fixture or protection module. Additional connections for redundant protective earth may be required.
(5)	HI and LO connections using Model HV-CA-571-3 high-voltage triaxial female panel mount to unterminated cable assembly. LO is connected to the metal noise shield.
(6)	High-voltage triaxial cable assembly (Model HV-CA-554)
(7)	Model 2651A.

Low-current measurement programming example

Example code for a typical low-current measurement is shown below. This code assumes that a 100 GΩ resistor is being tested.

```
-- Restore defaults.
smua.reset()
-- Set source to DC V.
smua.source.func = smua.OUTPUT_DCVOLTS
-- Select 200 V source range.
smua.source.rangev = 200
-- Output 100 Vdc.
smua.source.levelv = 100
-- Select 1 nA range.
smua.measure.rangei = 1e-9
-- Set current limit to 2 nA.
smua.source.limiti = 2e-9
-- Turn on output.
smua.source.output = smua.OUTPUT_ON
-- Delay 1 second to allow for source and measure settling.
smua.source.delay = 1
-- Returns current reading.
print(smua.measure.i())
-- Returns resistance reading.
print(smua.measure.r())
-- Turn off output.
smua.source.output = smua.OUTPUT_OFF
```

How can I change the line frequency or voltage?

The Model 2651A requires a line voltage of 100 V ac to 240 V ac (±10%) and a line frequency of 50 Hz or 60 Hz. The factory configures the Model 2651A to automatically detect and operate at the appropriate power line frequency each time the instrument power is turned on. In noisy environments, it may be necessary to manually configure the instrument to match the actual line frequency. For more information, see Line frequency configuration.

Where can I get the LabVIEW driver?

The latest NI™ LabVIEW™ driver is available on tek.com/keithley.

What should I do if I get an 802 interlock error?

You receive error code 802, "OUTPUT blocked by interlock," if you:

- Disengage the interlock when the Model 2651A output is already on.
- Attempt to turn on the Model 2651A output when the interlock is disengaged.

To recover from this error, properly engage the interlock using a safe test fixture, and then turn on the Model 2651A output.

Why is the reading value 9.91e37?

This value indicates that there is a measurement overflow error. This error occurs when:

- A measurement performed on a fixed range has a measured value greater than the specified range
- The measured value is larger than the maximum current or voltage range of the instrument (exceeds the instrument rating)

If the instrument displays the overflow message on a particular range, select a higher range until an on-range reading is displayed. To ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow.

Section 10

Next steps

In this section:

Additional Model 2651A information

For additional information about the Model 2651A, refer to the Keithley Instruments website (tek.com/keithley), which contains the most up-to-date information. From the website, you can access:

- *The Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*
- *Semiconductor Device Test Applications Guide*
- Application notes
- Updated drivers
- Information about related products, including:
  - Series 2600B System SourceMeter® Instruments
  - The Model 4200A-SCS Semiconductor Characterization System

In addition, your local Field Applications Engineer can help you with product selection, configuration, and usage. Check the website for contact information.

Section 11

Maintenance

In this section:

Introduction

The information in this section describes routine maintenance of the instrument that the operator can perform.

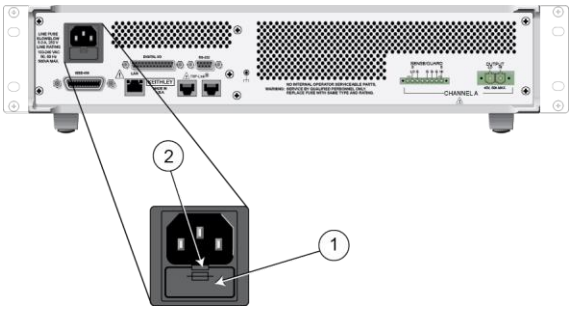
Line fuse replacement

A fuse on the Model 2651A rear panel protects the power line input of the instrument. See the following instructions to replace the fuse. You do not need to return your instrument for service if the fuse is damaged.

WARNING

Disconnect the line cord at the rear panel and remove all test leads connected to the instrument before replacing a line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.

Figure 126: 2651A fuse replacement



WARNING

To prevent injury, death, or instrument damage, use only the correct fuse type (see table).

To replace the line fuse:

1. Power off the instrument and remove the line cord.
2. The fuse drawer (item 1 in the figure) is located below the ac receptacle. A small tab is located on the top of the fuse drawer (item 2). Using a thin-bladed knife or a screwdriver, pry this tab away from the ac receptacle.
3. Slide the fuse drawer out to gain access to the fuse (the fuse drawer does not pull completely out of the power module).
4. Snap the fuse out of the drawer and replace it with the same type (the fuse is specified in the table below).
5. Push the fuse drawer back into the module.

If a fuse continues to become damaged, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

Line fuse

Line voltage	Rating	Keithley part number
100 V to 240 V	5 A, 250 V, FAST-ACTING CARTRIDGE FUSE, 5 x 20 mm	FU-154-5

Front-panel tests

The front-panel tests test the functionality of the front-panel keys and the display.

NOTE

In the following procedures, highlight the menu item and press the **ENTER** key to select it. You can also select a menu item by pressing the navigation wheel.

Keys test

This test checks the functionality of each front-panel key.

Perform the following steps to run the KEYS test:

1. If the instrument is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.

3. Navigate through the menus by turning the navigation wheel. Press the **ENTER** key to select the menu items as follows: **DISPLAY > TEST > DISPLAY-TESTS**.
4. Turn the navigation wheel until the **KEYS** menu item is highlighted.
5. To start the test, press the **ENTER** key. When you press a key while the test is active, the label name for that key is displayed to indicate that it is functioning properly. When you release the key, the message `No keys pressed` is displayed.
6. To test the EXIT (LOCAL) key, press the **EXIT (LOCAL)** key once.
7. To exit the test, press the **EXIT (LOCAL)** key twice consecutively. You exit the test and the instrument returns to the FRONT PANEL TESTS menu.
8. Press the **EXIT (LOCAL)** key multiple times to exit out of the menu structure.

Display patterns test

This test lets you verify that each pixel and indicator in the vacuum fluorescent display is working properly.

Perform the following steps to run the display test:

1. If the instrument is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Navigate through the menus by turning the navigation wheel, and then pressing the **ENTER** key to select the items as follows: **DISPLAY > TEST > DISPLAY-TESTS**.
4. Turn the navigation wheel until the **DISPLAY-PATTERNS** menu item is highlighted.
5. To start the display test, press the **ENTER** key. There are three parts to the display test. Each time the **ENTER** key or the navigation wheel is pressed, the next part of the test sequence is selected. The test sequence is as follows:
   - Checkerboard pattern and the indicators that are on during normal operation
   - Checkerboard pattern (alternate pixels on) and all the numeric indicators (which are not used) are illuminated
   - Each digit and adjacent indicators are sequenced; all the pixels of the selected digit are on
6. When finished, abort the display test by pressing the **EXIT (LOCAL)** key. The instrument returns to the FRONT PANEL TESTS menu. Continue pressing the **EXIT (LOCAL)** key to exit out of the menu structure.

Upgrading the firmware

You can load a newer or older version of firmware to the instrument. The process takes about five minutes.

To load the firmware, you can use a USB flash drive or select a file from a computer.

From the front panel, you must use the USB flash drive. Make sure the USB flash drive is empty except for the firmware file.

From the web interface, Test Script Builder, or remote interface, you can select the file from the computer.

Firmware files are available for download from the Product Support and Downloads web page (tek.com/product-support) in the category "Software." After downloading the file, unzip the file. The file with the extension `.x` is the firmware file.

CAUTION

Disconnect the input and output terminals before you upgrade or downgrade.

Do not remove power from the instrument or remove the USB flash drive while an upgrade or downgrade is in progress. Wait until the instrument completes the procedure and shows the opening display. If you are upgrading an instrument with no front panel (NFP), the LAN and 1588 LEDs on the front panel blink in unison during the upgrade and stop when the upgrade is complete.

Do not initialize or reset TSP-Link before starting the upgrade.

If you are loading an older version, check the release notes and verify that the version of firmware you are loading is compatible with your hardware.

Before starting the upgrade, turn the instrument power off, wait a few seconds, then turn the instrument power on.

To upgrade or downgrade the firmware using the front panel:

1. Turn the instrument power off. Wait a few seconds.
2. Turn the instrument power on.
3. Copy the firmware file to a USB flash drive.
4. Disconnect the input and output terminals to and from the instrument.
5. If the instrument is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
6. Insert the flash drive into the USB port on the front panel of the Model 2651A.

7. From the front panel, press the **MENU** key
8. Turn the navigation wheel to go to the **UPGRADE** menu item, and then press the **ENTER** key.
9. Turn the navigation wheel to select the file on the USB flash drive that contains the appropriate version of firmware.
10. Press the **ENTER** key to upgrade the firmware. The status of the upgrade is displayed.

The instrument reboots automatically when the upgrade is complete.

To upgrade or downgrade the firmware from the web interface:

1. Turn the instrument power off. Wait a few seconds.
2. Turn the instrument power on.
3. Open a web browser on the host computer.
4. Enter the IP address of the instrument in the web browser address box. For example, if the instrument IP address is 192.168.1.101, enter `192.168.1.101` in the browser address box.
5. Press **Enter** on the computer keyboard to open the web interface of the instrument.
6. From the left navigation area, select **Flash Upgrade**.
7. Choose **Select File**. A file selection dialog box is displayed.
8. Select the firmware file.
9. Select **Start**. A progress dialog box is displayed. When the upgrade begins, you can also see progress on the front-panel display.

After the instrument automatically restarts, it is ready for use.

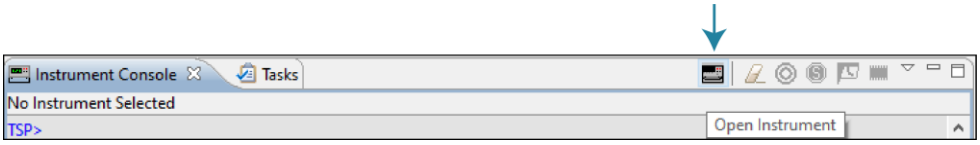
Using TSB to upgrade the firmware

After downloading the flash file from tek.com/keithley, you can use Test Script Builder (TSB) to upgrade the firmware of your Model 2651A.

To upgrade the firmware using Test Script Builder:

1. Start Test Script Builder.
2. On the Instrument Console toolbar, click the **Open Instrument** icon.

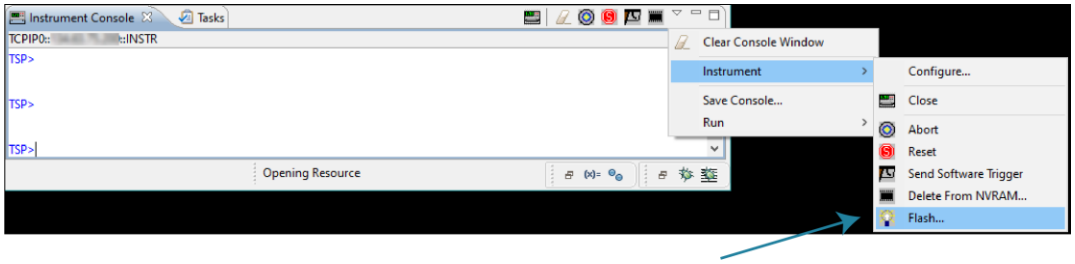
Figure 127: Open Instrument icon



3. Select your instrument from the **Select Instrument** dialog box.

4. On the Instrument Console toolbar, select the down arrow and select **Instrument > Flash**.

Figure 128: Flash menu option



5. For **Select or enter a firmware image file**, use the browser to select the new firmware.
6. For **Select a replacement mode**, select **Upgrade** to install a newer version of firmware or **Downgrade** to install an older of firmware.
7. Wait until the upgrade or downgrade is complete and the instrument shows the opening display.

Section 12

Calibration

In this section:

Verification

WARNING

The information in this section is intended for qualified service personnel only, as described by the types of product users in the Safety precautions pages, provided at the beginning of this document. Do not attempt these procedures unless you are qualified to do so.

Some of these procedures may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

Use the procedures in this section to verify that the Model 2651A accuracy is within the limits stated in the instrument's one-year accuracy specifications. Perform the verification procedures:

- When you first receive the instrument to make sure that it was not damaged during shipment.
- To verify that the instrument meets factory specifications.
- To determine if calibration is required.
- After performing a calibration adjustment to make sure the instrument was adjusted properly.

NOTE

If the instrument is still under warranty and its performance is outside specified limits, contact your Keithley Instruments representative or the factory to determine the correct course of action.

Calibration test requirements

Be sure that you perform the calibration tests:

- Under the proper environmental conditions.
- After the specified warmup period.
- Using the correct line voltage.
- Using the proper test equipment.
- Using the specified output signal and reading limits.

NOTE

Product specifications are subject to change. Listed uncertainties and test limits are provided only as examples. Always verify values against the most recent product specifications.

Environmental conditions

Conduct your performance calibration procedures in a test environment with:

- An ambient temperature of 18 °C to 28 °C.
- A relative humidity of less than 70 percent unless otherwise noted.

NOTE

Product specifications that are listed as 18 °C to 28 °C assume adjustment has been done at 23 °C. If the Model 2651A is adjusted at a different temperature, the specifications apply to ±5 °C of that adjustment temperature.

Line power

The Model 2651A requires a line voltage of 100 V to 240 V and a line frequency of 50 Hz or 60 Hz. Perform calibration tests in this range.

Warmup period

Allow the Model 2651A to warm up for at least two hours before conducting the calibration procedures.

If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the internal temperature of the instrument to stabilize. Typically, allow one extra hour to stabilize an instrument that is 10 °C outside the specified temperature range.

Also allow test equipment to warm up for the minimum time specified by the manufacturer.

Recommended verification equipment

The following table summarizes recommended maximum allowable test equipment uncertainty for verification points. Total test equipment measurement uncertainty should meet or be less than the listed values at each test point. Generally, test equipment uncertainty should be at least four times better than corresponding Model 2651A specifications.

Description	Manufacturer/model	Accuracy	
Digital multimeter		DC voltage	50 mV: ±1500 ppm 90 mV: ±880 ppm 0.5 V: ±200 ppm 0.9 V: ±130 ppm 5 V: ±200 ppm 9 V: ±130 ppm 10 V: ±170 ppm 18 V: ±110 ppm 20 V: ±200 ppm 36 V: ±130 ppm
Digital multimeter (continued)		DC current	50 nA: ±2700 ppm 90 nA: ±1500 ppm 500 nA: ±1200 ppm 900 nA: ±750 ppm 5 μA: ±600 ppm 9 μA: ±420 ppm 50 μA: ±170 ppm 90 μA: ±110 ppm 0.5 mA: ±150 ppm 0.9 mA: ±100 ppm 5 mA: ±170 ppm 9 mA: ±110 ppm 50 mA: ±150 ppm 90 mA: ±100 ppm 500 mA: ±1600 ppm 900 mA: ±950 ppm 2.5 A: ±420 ppm 4.5 A: ±290 ppm 5 A: ±600 ppm 9 A: ±460 ppm 10 A: ±400 ppm 18 A: ±310 ppm
		DC current Pulse only*	20 A: ±750 ppm 45 A: ±620 ppm
Precision resistor: 0.1 Ω, 250 W, 0.1%	Isotek RUG-Z-R100-0.1-TK1	Resistance**	0.1 Ω: ±75 ppm
* To verify this pulse current, DMM must be able to complete its measurement of the shunt voltage within 500 μs of receiving a trigger signal. ** Resistor used to test 1 A, 5 A, 10 A, and 20 A ranges (dc level), and 50 A range (pulse). Before use, characterize the resistor to the uncertainty shown.			

Calibration limits

The calibration limits stated in this section have been calculated using only the Model 2651A one-year accuracy specifications. They do not include test equipment uncertainty. If a particular measurement falls outside the allowable range, recalculate new limits based both on the Model 2651A specifications and corresponding test equipment specifications.

Source limit calculations

As an example of how to calculate source verification limits, assume you are testing the Model 2651A 10 Vdc output range using a 9 V output value. Using the one-year accuracy specification for the 10 V range dc output of ± (0.02% of output + 5 mV offset), the calculated output limits are:

Output limits = 9.0 V ± [(9.0 V × 0.02%) + 5 mV]
Output limits = 9.0 V ± (0.0018 V + 0.005 V)
Output limits = 9.0 V ± 0.0068 V
Output limits = 8.9932 V to 9.0068 V

Source limits for current are calculated in the same way but using the Model 2651A instrument's current source specifications.

Measurement limit calculations

Measurement limits are calculated in the same way as the source limits, except that the limits are calculated with respect to the measurement of the external reference instrument. For example, suppose that the Model 2651A is programmed to source 9.0 V and the external precision DMM measures 8.9993 V. Using the one-year accuracy specification for the 10 Vdc range measurement of ± (0.02% of output + 3 mV offset), the calculated measurement limits are:

Measurement limits = 8.9993 V ± [(8.9993 V × 0.02%) + 3 mV]
Measurement limits = 8.9993 V ± (0.00179986 V + 0.003 V)
Measurement limits = 8.9993 V ± 0.00479986 V
Measurement limits = 8.9945 V to 9.0041 V

Measurement limits for current are calculated in the same way but using the Model 2651A instrument's current measurement specifications.

Restoring factory defaults

Before performing the calibration procedures, restore the instrument to its factory defaults.

To restore the factory defaults:

1. Press the **MENU** key.
2. Scroll to the **SETUP** menu item, and then press the **ENTER** key.
3. Scroll to the **RECALL** menu item, and then press the **ENTER** key.
4. Scroll to the **INTERNAL** menu item, and then press the **ENTER** key.
5. Scroll to the **FACTORY** menu item.
6. Press the **ENTER** key to restore defaults.

Performing the calibration test procedures

Perform the following calibration tests to make sure the instrument is operating within specifications:

- Current source accuracy (on page 12-7)
- Current measurement accuracy (on page 12-9)
- Voltage source accuracy (on page 12-13)
- Voltage measurement accuracy (on page 12-14)

If the Model 2651A is not within specifications and not under warranty, see the procedures in Adjustment (on page 12-15) for information on adjusting the instrument.

Test considerations

When performing the verification procedures:

- Be sure to restore factory front panel defaults as outlined above.
- Make sure that the test equipment is properly warmed up and connected to the Model 2651A output terminals (use 4-wire sensing for voltage).
- Make sure the Model 2651A SMU is set to the correct source range.
- Be sure the Model 2651A SMU output is turned on before making measurements.
- Be sure the test equipment is set up for the proper function and range.
- Allow the Model 2651A SMU output signal to settle before making a measurement.
- Do not connect test equipment to the Model 2651A SMU through a scanner, multiplexer, or other switching equipment.

WARNING

The maximum common-mode voltage (voltage between LO and chassis ground) is 250 V dc. Exceeding this value may cause a breakdown in insulation, creating a shock hazard that could result in personal injury or death.

The input/output terminals of the Model 2651A High Power System SourceMeter® instrument SMU are rated for connection to circuits rated Measurement Category I only, with transients rated less than 1500 V peak above the maximum rated input. Do not connect the Model 2651A terminals to CAT II, CAT III, or CAT IV circuits. Connection of the Model 2651A terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2651A while the instrument is powered on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Setting the source range and output value

Before testing each calibration point, you must properly set the source range and output value.

To set the source range and output value:

1. Press the **SRC** key to select the appropriate source function.
2. Press the navigation wheel to enable the edit mode (EDIT indicator on).
3. When the cursor in the source display field is flashing, set the source range to the range being calibrated. Use the up or down **RANGE** keys to select the range.
4. Use the navigation wheel and **CURSOR** keys to set the source value to the required value.
5. Press the navigation wheel to complete editing.

Setting the measurement range

When simultaneously sourcing and measuring either voltage or current, the measure range is coupled to the source range, and you cannot independently control the measure range. Thus, it is not necessary for you to set the range when testing voltage or current measurement accuracy (on page 12-9).

Current source accuracy

Follow the steps below to verify that the Model 2651A output current accuracy is within specified limits.

1. With the power off, connect the digital multimeter to the Model 2651A terminals as shown in Connections for current verification (100 mA range and below).
2. Select the multimeter dc current measuring function.
3. Press the **SRC** key to source current, and make sure the source output is turned on.
4. Verify output current accuracy for each of the currents for the 100 nA through 100 mA ranges using the values listed in the following table. For each test point:
   - Select the correct source range.
   - Set the Model 2651A output current to the correct value.
   - Verify that the multimeter reading is within the limits given in the table below.

Output current accuracy limits

Source range	Output current setting[1]	Output current limits (1 year, 18° C to 28° C)
100 nA	50.000 nA 90.000 nA	49.450 nA to 50.550 nA 89.410 nA to 90.590 nA
1 μA	500.00 nA 900.00 nA	497.50 nA to 502.50 nA 897.10 nA to 902.90 nA
10 μA	5.0000 μA 9.0000 μA	4.9850 μA to 5.0150 μA 8.9810 μA to 9.0190 μA
100 μA	50.000 μA 90.000 μA	49.925 μA to 50.075 μA 89.913 μA to 90.087 μA
1 mA	0.50000 mA 0.90000 mA	0.49955 mA to 0.50045 mA 0.89943 mA to 0.90057 mA
10 mA	5.0000 mA 9.0000 mA	4.9905 mA to 5.0095 mA 8.9893 mA to 9.0107 mA
100 mA	50.000 mA 90.000 mA	49.955 mA to 50.045 mA 89.943 mA to 90.057 mA
1 A	500.00 mA 900.00 mA	496.10 mA to 503.90 mA 895.78 mA to 904.22 mA
5 A	2.50000 A 4.50000 A	2.4945 A to 2.5055 A 4.4929 A to 4.5071 A
10 A	5.00000 A 9.00000 A	4.9865 A to 5.0135 A 8.9805 A to 9.0195 A
20 A	10.0000 A 18.0000 A	9.977 A to 10.023 A 17.965 A to 18.035 A
50 A (pulse)	20.0000 A (dc level) 45.0000 A (pulse)[2]	19.890 A to 20.110 A 44.853 A to 45.148 A

1. Modify current limits if necessary. See Source limit calculations (on page 12-4).
2. Verification of a pulsed output (levels greater than 20 A on the 50 A range) is accomplished during current measurement verification test procedure. See Current measurement accuracy (on page 12-9).

5. Repeat the procedure for negative output currents with the same magnitudes as those listed.
6. Turn the output off, and change connections as shown in Connections for current verification (1 A range and above) (use the 0.1 Ω resistor as shown).
7. Select the DMM dc volts function.
8. Repeat steps 4 through 6 for the 1 A through 20 A ranges, and the 20.0000 A (dc level) output setting for the 50 A (pulse) range. Calculate the current from the DMM voltage reading and the characterized 0.1 Ω resistance value: *I=V/R*.

Figure 129: Connections for current verification (100 mA range and below)

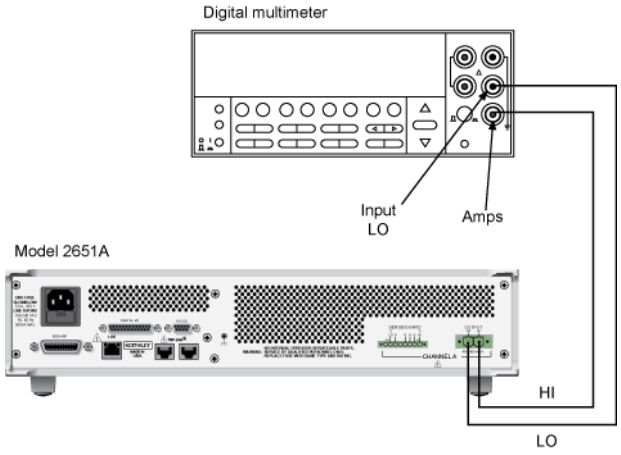
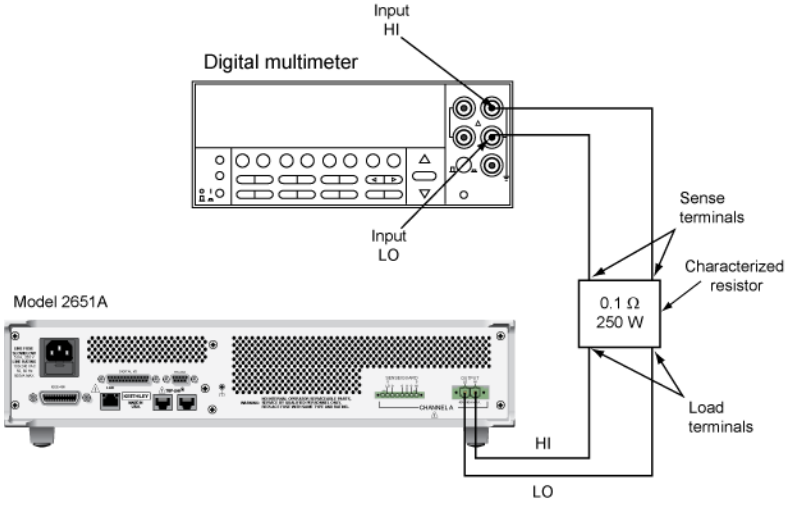


Figure 130: Connections for current verification (1 A range and above)



Current measurement accuracy

Follow the steps below to verify that Model 2651A current measurement accuracy is within specified limits. The procedure involves applying accurate currents from the Model 2651A current source and then verifying that Model 2651A current measurements are within required limits.

1. With the power off, connect the digital multimeter to the Model 2651A terminals as shown in Connections for current verification (100 mA range and below).
2. Select the multimeter dc current measuring function.
3. Set the Model 2651A to both source and measure current by pressing the **SRC** and then the **MEAS** keys. Make sure the source output is turned on.
4. Verify measure current accuracy for each of the currents listed in the table below for ranges 100 nA through 100 mA. For each measurement:
   - Select the correct source range.
   - Set the Model 2651A output current such that the digital multimeter reading is the value indicated in the source current column of the table below. It may not be possible to set the current source to get exactly the required reading on the digital multimeter. Use the closest possible setting and modify the reading limits accordingly.
   - If necessary, press the **TRIG** key to display readings.
   - Verify that the Model 2651A current reading is within the limits given in the table below.
5. Repeat the procedure for negative calibrator currents with the same magnitudes as those listed.

Current measurement accuracy limits

Source and measure range[1]	Source current[2]	Current reading limits (1 year, 18° C to 28° C)
100 nA	50.000 nA 90.000 nA	49.460 nA to 50.540 nA 89.428 nA to 90.572 nA
1 μA	0.5000 μA 0.9000 μA	497.60 nA to 502.40 nA 897.28 nA to 902.72 nA
10 μA	5.0000 μA 9.0000 μA	4.9880 μA to 5.0120 μA 89.848 μA to 9.0152 μA
100 μA	50.000 μA 90.000 μA	49.965 μA to 50.035 μA 89.957 μA to 90.043 μA
1 mA	0.50000 mA 0.90000 mA	499.70 μA to 500.30 μA 899.62 μA to 900.38 μA
10 mA	5.0000 mA 9.0000 mA	4.9965 mA to 5.0035 mA 8.9957 mA to 9.0043 mA
100 mA	50.000 mA 90.000 mA	49.970 mA to 50.030 mA 89.962 mA to 90.038 mA
1 A	0.50000 A 0.90000 A	496.75 mA to 503.25 mA 896.55 mA to 903.45 mA
5 A	2.50000 A 4.50000 A	2.4958 A to 2.5043 A 4.4948 A to 4.5053 A

Current measurement accuracy limits

Source and measure range[1]	Source current[2]	Current reading limits (1 year, 18° C to 28° C)
10 A	5.00000 A 9.00000 A	4.9880 A to 5.0120 A 8.9832 A to 9.0168 A
20 A	10.0000 A 18.0000 A	9.984 A to 10.016 A 17.978 A to 18.022 A
50 A (pulse)[3]	20.0000 A (dc level) 45.0000 A (pulse)	19.940 A to 20.060 A 44.888 A to 45.113 A
1. Measure range coupled to source range when simultaneously sourcing and measuring current. 2. As measured by precision digital multimeter. Use closest possible value, and modify reading limits accordingly if necessary. See Measurement limit calculations (on page 12-4). 3. The 20 A (dc level) source current uses the integrating ADC, while the 45 A (pulse) source current uses the FAST ADC called form the high speed ADC pulse verification script.		

6. Turn the output off, and change connections as shown in Connections for current verification (1 A range and above).
7. Select the DMM volts function.
8. Repeat steps 4 through 6 for the 1 A, 5 A, 10 A, and 20 A ranges; and for the 20 ADC level on the 50 A range. Calculate the current from the DMM voltage reading and characterized 0.1 Ω resistance value: I=V/R
9. Verification of a pulsed output (levels greater than 20 A on the 50 A range) is accomplished with the same techniques and characterized 0.1 Ω resistor (shunt) as used for other 1 A and above ranges. However, there is an additional requirement that the external DMM must be capable of synchronizing to and measuring the short duration shunt voltage pulse with the required precision.
10. The -45 A high speed ADC pulse verification script (on page 12-11) provides an example for setting up a single 1 ms pulse on the 50 A range and measuring the last 500 µs of the pulse using the Model 2651A instrument's fast ADC. The script also sets up a trigger signal that will appear on the rear panel digital I/O connector and be synchronized to the start of the fast ADC measurement (500 µs after the start of the pulse). The external DMM must be able to complete its measurement of the shunt voltage within 500 µs of receiving this trigger signal.

-45 A high speed ADC pulse verification script

As shown, the script generates a single -45 A pulse. To generate a different current level, edit the line:

```
PulseLevel = -45
```

Refer to Instrument programming (on page 6-1) for information on loading and running scripts. Refer to Functions and features (on page 3-1) for detailed information about generating pulses and triggers.

```
-- '-45 A high speed ADC pulse verification script'--
-- Set up SMU idle state ranges and limits.
smua.reset()
smua.source.highc = smua.DISABLE
smua.source.delay = 0
smua.source.func = smua.OUTPUT_DCAMPS
smua.source.rangei = 50
smua.measure.rangei = 50
smua.measure.rangev = 10
smua.source.limitv = 10

-- Create a variable for pulsed output value.
PulseLevel = -45
-- Note: Make sure that the source idle value has the same polarity
--       as the pulse value to avoid a ~100 uS polarity change
--       time delay during the pulse.
smua.source.leveli = .000001 * PulseLevel

-- Set up single 1 mS pulse using timer[1].
PulseTimer = trigger.timer[1]
PulseTimer.delay = .001
PulseTimer.stimulus = smua.trigger.ARMED_EVENT_ID
PulseTimer.count = 1

-- Set up source pulse parameters.
smua.trigger.endpulse.stimulus = PulseTimer.EVENT_ID
smua.trigger.endpulse.action = smua.SOURCE_IDLE
smua.trigger.source.limitv = 10
smua.trigger.source.stimulus = smua.trigger.ARMED_EVENT_ID
smua.trigger.source.action = smua.ENABLE
-- Use a single item list to program the pulse level
smua.trigger.source.listi({PulseLevel})

-- Set up measurement trigger 500 uS into the 1 mS pulse.
MeasStartTimer = trigger.timer[2]
MeasStartTimer.delay = .0005
MeasStartTimer.stimulus = smua.trigger.ARMED_EVENT_ID
MeasStartTimer.count = 1

-- Set up Fast ADC to take 500 measurements at 1 uS interval.
smua.measure.autozero = smua.AUTOZERO_OFF
smua.measure.filter.enable = smua.FILTER_OFF
smua.measure.adc = smua.ADC_FAST
smua.trigger.measure.action = smua.ASYNC
smua.trigger.measure.stimulus = MeasStartTimer.EVENT_ID
```

```
smua.measure.delay = 0
smua.measure.count = 500
smua.measure.interval = 0.000001

-- Use dedicated buffers to receive measurements.
smua.nvbuffer1.clear()
smua.nvbuffer2.clear()
-- Use nvbuffer1 for current and nvbuffer2 for volts
smua.trigger.measure.iv(smua.nvbuffer1, smua.nvbuffer2)

-- Generate an external trigger signal on digital I/O connector
--  pin 8 (10 uS falling edge pulse in this example), synchronized
--  to the start of the Fast ADC measurement for use by
--  an external measurement instrument.
digio.trigger[8].mode = digio.TRIG_FALLING
digio.trigger[8].pulsewidth = 10e-6
digio.trigger[8].stimulus = MeasStartTimer.EVENT_ID

-- Set up trigger model state machine for one pulse
--  when smua.trigger.arm.set() command is sent.
smua.trigger.arm.stimulus = trigger.EVENT_ID
smua.trigger.arm.count = 1
smua.trigger.count = 1

-- Enable output (initially at idle level).
smua.source.output = smua.OUTPUT_ON

-- Enter trigger model state machine.
smua.trigger.initiate()

-- Start pulse.
smua.trigger.arm.set()
waitcomplete()

-- Pulse is done - the output can be turned off.
smua.source.output = smua.OUTPUT_OFF

-- Retrieve all 500 individual measurements as comma-separated list
--  (parameters can be adjusted to retrieve a subset).
format.data = format.ASCII
printbuffer(1, 500, smua.nvbuffer1) -- current measurements.
printbuffer(1, 500, smua.nvbuffer2) -- voltage measurements.

-- Use getstats functions to retrieve aggregate information.
istats = smua.buffer.getstats(smua.nvbuffer1)
vstats = smua.buffer.getstats(smua.nvbuffer2)
print(istats.mean) -- average of current measure
print(vstats.mean) -- average of voltage measure
```

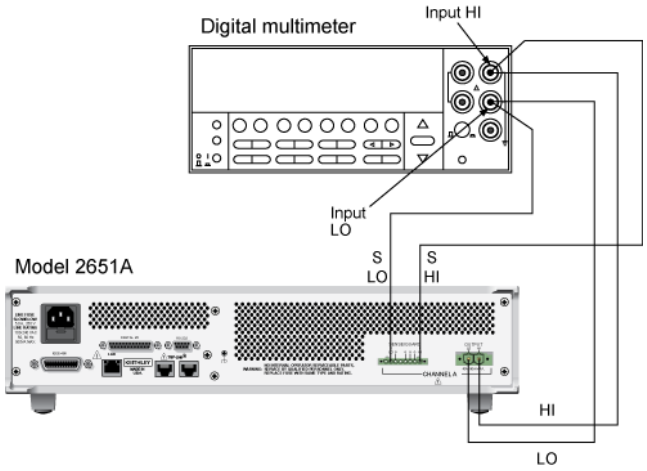
Voltage source accuracy

Follow the steps below to verify that the Model 2651A output voltage accuracy is within specified limits. To perform this test, you will set the output voltage to each full-range value and measure the voltages with a precision digital multimeter.

To perform the voltage source accuracy test:

1. With the power off, connect the digital multimeter (DMM) to the Model 2651A output terminals using 4-wire connections, as shown below.

Figure 131: Connections for voltage verification



2. Set the multimeter measuring function to dc volts.
3. Press the **SRC** key to source voltage and make sure the source output is turned on.
4. Enable the Model 2651A 4-wire (remote sense) mode:
   a. a. Press the **CONFIG** key and then the **SRC** key.
   b. b. Select **V-SOURCE > SENSE-MODE > 4-WIRE**.
5. Verify output voltage accuracy for each of the voltages listed in the following table. For each test point:
   - Select the correct source range.
   - Set the Model 2651A output voltage to the indicated value.
   - Verify that the multimeter reading is within the limits given in the table.

-

Output voltage accuracy limits

Source range	Output voltage setting*	Output voltage limits (1 year, 18° C to 28° C)
100 mV	50.000 mV 90.000 mV	49.490 mV to 50.510 mV 89.482 mV to 90.518 mV
1 V	0.50000 V 0.90000 V	0.49940 V to 0.50060 V 0.89932 V to 0.90068 V
10 V	5.0000 V 9.0000 V	4.9940 V to 5.0060 V 8.9932 V to 9.0068 V
20 V	10.000 V 18.000 V	9.993 V to 10.007 V 17.991 V to 18.009 V
40 V	20.000 V 36.000 V	19.984 V to 20.016 V 35.981 V to 36.019 V
* Modify voltage limits if necessary. See Source limit calculations (on page 12-4).		

6. Repeat the procedure for negative output voltages with the same magnitudes as those listed in the previous table, as applicable.

Voltage measurement accuracy

Follow the steps below to verify that the Model 2651A voltage measurement accuracy is within specified limits. To perform this test, you will set the source voltage, as measured by a precision digital multimeter, and then verify that the Model 2651A voltage readings are within required limits.

1. With the power off, connect the digital multimeter to the Model 2651A output terminals using 4-wire connections (the same Connections for voltage verification as shown in Voltage source accuracy (on page 12-13)).
2. Select the multimeter dc volts function.
3. Enable the Model 2651A 4-wire (remote sense) mode:

   a. Press the **CONFIG** key and then the **MEAS** key.

   b. Select **V-MEAS > SENSE-MODE > 4-WIRE**.
4. Set the Model 2651A SMU to both source and measure voltage by pressing the **SRC** and then the **MEAS** keys.
5. Make sure the source output is turned on (if off, press the **OUTPUT ON/OFF** control).
6. Verify voltage measurement accuracy for each of the voltages listed in the table (see below). For each test point:
   - Select the correct source range.
   - Set the Model 2651A output voltage such that the digital multimeter reading is the value indicated in the source voltage column of the table below. It may not be possible to set the voltage source to get exactly the required reading on the digital multimeter. Use the closest possible setting and modify the reading limits accordingly.

- Verify that the Model 2651A voltage reading is within the limits given in the table.

Voltage measurement accuracy limits

Source and measure range*	Source voltage**	Voltage reading limits (1 year, 18° C to 28° C)
100 mV	50.000 mV 90.000 mV	49.690 mV to 50.310 mV 89.682 mV to 90.318 mV
1 V	0.50000 V 0.90000 V	0.49960 V to 0.50040 V 0.89952 V to 0.90048 V
10 V	5.0000 V 9.0000 V	4.9960 V to 5.0040 V 8.9952 V to 9.0048 V
20 V	10.000 V 18.000 V	9.993 V to 10.007 V 17.991 V to 18.009 V
40 V	20.000 V 36.000 V	19.984 V to 20.016 V 35.981 V to 36.019 V
* Measure range coupled to source range when simultaneously sourcing and measuring voltage. ** As measured by precision digital multimeter. Use closest possible value, and modify reading limits accordingly if necessary. See Measurement limit calculations (on page 12-4).		

7. Repeat the procedure for negative source voltages with the same magnitudes as those listed.
8. Perform high speed verification procedure for the High Speed ADC Volts measurements:
   a. Change to using the high-speed ADC by pressing the **SPEED** key and then selecting FAST from the menu.
   b. Repeat the procedure (step 6) for each of the voltages listed in the following table.
   c. Repeat the procedure for negative source voltages with the same magnitudes as those listed in the following table.
   d. Change back to the integrating ADC by pressing the **SPEED** key and then selecting NORMAL from the menu.

High Speed ADC Volts measurement accuracy limits

Source and measure range[1]	Source voltage[2]	Voltage reading limits (1 year, 18° C to 28° C)
1 V	0.50000 V 0.90000 V	0.49915 V to 0.50085 V 0.89895 V to 0.90105 V
1. Measure range coupled to source range when simultaneously sourcing and measuring voltage. 2. As measured by precision digital multimeter. Use closest possible value, and modify reading limits accordingly if necessary. See Measurement limit calculations (on page 12-4).		

Adjustment

WARNING

The information in this section is intended for qualified service personnel only, as described by the types of product users in the Safety precautions pages, provided at the beginning of this document. Do not attempt these procedures unless you are qualified to do so.

Some of these procedures may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

Use the procedures in this section to calibrate the Model 2651A.

These procedures require accurate test equipment to measure precise dc voltages and currents.

NOTE

Product specifications are subject to change. Listed uncertainties and test limits are provided only as an example. Always verify values against actual product specifications.

Environmental conditions

Temperature and relative humidity

Conduct the adjustment procedures at an ambient temperature of 18 °C to 28 °C, with relative humidity of less than 70 percent (unless otherwise noted).

NOTE

Product specifications that are listed as 18 °C to 28 °C assume adjustment has been done at 23 °C. If the Model 2651A is adjusted at a different temperature, the specifications apply to ±5 °C of that temperature.

Line power

The Model 2651A requires a line voltage of 100 V to 240 V at a line frequency of 50 Hz or 60 Hz. The instrument must be adjusted within this range.

Warmup period

Allow the Model 2651A to warm up for at least two hours before adjusting the instrument.

If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the internal temperature of the instrument to stabilize. Typically, allow one extra hour to stabilize an instrument that is 10 °C outside the specified temperature range.

Allow the test equipment to warm up for the minimum time specified by the manufacturer.

Adjustment considerations

When performing the adjustment procedures:

- Make sure that the test equipment is properly warmed up and connected to the correct Model 2651A terminals.
- Always allow the source signal to settle before calibrating each point.
- Do not connect test equipment to the Model 2651A SMU through a scanner or other switching equipment.
- If an error occurs during calibration, the Model 2651A will generate an appropriate error message. See Error summary list (on page 8-3) for more information.

WARNING

The maximum common-mode voltage (voltage between LO and chassis ground) is 250 V dc. Exceeding this value may cause a breakdown in insulation, creating a shock hazard that could result in personal injury or death.

The input/output terminals of the Model 2651A High Power System SourceMeter® instrument SMU are rated for connection to circuits rated Measurement Category I only, with transients rated less than 1500 V peak above the maximum rated input. Do not connect the Model 2651A terminals to CAT II, CAT III, or CAT IV circuits. Connection of the Model 2651A terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2651A while the instrument is powered on. Turn off the equipment from the front panel or disconnect the main power cord from the rear of the Model 2651A before handling cables. Putting the equipment into standby does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Adjustment cycle

Perform an adjustment at least once a year to make sure the instrument meets or exceeds its specifications.

Recommended calibration adjustment equipment

The table below contains the recommended equipment for the calibration adjustment procedures. You can use alternate equipment as long as that equipment has specifications equal to or greater than those listed in the table. When possible, test equipment specifications should be at least four times better than corresponding Model 2651A specifications.

Description	Manufacturer/model	Accuracy	
Digital multimeter		DC voltage	50 mV: ±1500 ppm 90 mV: ±880 ppm 0.5 V: ±200 ppm 0.9 V: ±130 ppm 5 V: ±200 ppm 9 V: ±130 ppm 10 V: ±170 ppm 18 V: ±110 ppm 20 V: ±200 ppm 36 V: ±130 ppm
		DC current	50 nA: ±2700 ppm 90 nA: ±1500 ppm 500 nA: ±1200 ppm 900 nA: ±750 ppm5 μA: ±600 ppm 9 μA: ±420 ppm 50 μA: ±170 ppm 90 μA: ±110 ppm 0.5 mA: ±150 ppm 0.9 mA: ±100 ppm 5 mA: ±170 ppm 9 mA: ±110 ppm 50 mA: ±150 ppm 90 mA: ±100 ppm 500 mA: ±1600 ppm 900 mA: ±950 ppm 2.5 A: ±420 ppm 4.5 A: ±290 ppm 5 A: ±600 ppm 9 A: ±460 ppm 10 A: ±400 ppm 18 A: ±310 ppm
		DC current Pulse only	20 A: ±750 ppm 45 A: ±620 ppm
Precision resistor: 0.1 Ω, 250 W, 0.1%	Isotek RUG-Z-R100-0.1-TK1	Resistance*	0.1 Ω: ±75 ppm
50 Ω resistors (2)	Any suitable.**		
* Resistor used to test 1 A, 5 A, 10 A, and 20 A ranges. Before use, characterize the resistor to the uncertainty shown. ** Used for contact check calibration. Before use, characterize with the resistance function of the digital multimeter.			

Calibration adjustment overview

The following topics contain an overview of the entire calibration adjustment procedure.

Parameter values

The full-scale parameters are 90 percent of full-scale as indicated (see the table in Step sequence (on page 12-19)). Note that you cannot send a value of 0 for the two zero parameters. Instead, you must send a very small value, such as 1e-30 or -1e-30.

Sense modes

The table titled "Model 2651A calibration steps" in Step sequence (on page 12-19) lists the sense modes for the calibration steps. Note that each source and measure range is calibrated using the LOCAL sense mode. In addition, the 1 V through 40 V ranges are calibrated using the REMOTE sense mode, and the 1 V, 1 mA, and 5 A source ranges are calibrated using the CALA sense mode.

Step sequence

Adjustment steps must be performed in a specific sequence. See the table titled "Model 2651A adjustment steps." Note that all steps are performed using 2-wire (local sensing) except as noted. Adjustment of each range is performed as a four-point adjustment:

- + ZERO
- + FULL SCALE
- – ZERO
- – FULL SCALE

NOTE

Before performing the adjustment steps, refer to Parameter values (on page 12-19) and Sense modes (on page 12-19).

Model 2651A adjustment steps

Function[1]	Adjustment steps[2]	Adjustment points[3]	Sense mode[4]
Voltage source and measure See Step 2. Voltage adjustment (on page 12-22)	100 mV	±1e-30, ±90 mV	smua.SENSE_LOCAL
	100 mV	±1e-30, ±90 mV	smua.SENSE_REMOTE
	1 V	±1e-30, ±0.9 V	smua.SENSE_LOCAL
	1 V	±1e-30, ±0.9 V	smua.SENSE_CALA
	1 V	±1e-30, ±0.9 V	smua.SENSE_REMOTE
	10 V	±1e-30, ±9 V	smua.SENSE_LOCAL
	10 V	±1e-30, ±9 V	smua.SENSE_REMOTE
	20 V	±1e-30, ±18 V	smua.SENSE_LOCAL
	20 V	±1e-30, ±18 V	smua.SENSE_REMOTE
	40 V	±1e-30, ±36 V	smua.SENSE_LOCAL
	40 V	±1e-30, ±36 V	smua.SENSE_REMOTE
Current source and measure See Step 3. Current adjustment (on page 12-27)	100 nA	±1e-30, ±90 nA	smua.SENSE_LOCAL
	1 µA	±1e-30, ±0.9 µA	smua.SENSE_LOCAL
	10 µA	±1e-30, ±9 µA	smua.SENSE_LOCAL
	100 µA	±1e-30, ±90 µA	smua.SENSE_LOCAL
	1 mA	±1e-30, ±0.9 mA	smua.SENSE_LOCAL
	1 mA	±1e-30, ±0.9 mA	smua.SENSE_CALA
	10 mA	±1e-30, ±9 mA	smua.SENSE_LOCAL
	100 mA	±1e-30, ±90 mA	smua.SENSE_LOCAL
	1 A	±1e-30, ±0.9 A	smua.SENSE_LOCAL
	5 A	±1e-30, ±4.5 A	smua.SENSE_LOCAL
	5 A	±1e-30, ±4.5 A	smua.SENSE_CALA
	10 A	±1e-30, ±9 A	smua.SENSE_LOCAL
	20 A	±1e-30, ±18 A	smua.SENSE_LOCAL
	50 A	±1e-30, ±20 A[5]	smua.SENSE_LOCAL
	100 A[6]	±1e-30, ±20 A[5]	smua.SENSE_LOCAL
Fast ADC	See Step 4. Fast ADC adjustment (on page 12-31)		
Contact check	See Step 5. Contact check adjustment (on page 12-31)		

1. Adjust only the source for the SENSE_CALA sense steps and for the 100 A SENSE_LOCAL steps.
2. Steps must be performed in the order shown.
3. Do not use actual 0 values for zero adjustment points. Use very small values such as ±1e-30. Adjustment polarities must also be set as shown in the procedures.
4. Output must be off before changing to the CALA sense mode.
5. The maximum dc level that can be used to adjust 50 A and 100 A ranges is 20 A.
6. The 100 A current source range adjust improves dynamic (pulse) operation of the 50 A range. "`smua.source.rangei = 100`" is valid only when calibration is unlocked.

Calibration commands quick reference

The following table summarizes remote calibration commands. For a more complete description of these commands, refer to the TSP command reference (on page 7-1).

Calibration commands	
Command	**Description**
`smua.cal.adjustdate = adjustDate`	Set date when the adjustment was done.
`smua.cal.date = calDate`	Set calibration date (`calDate` of 0 indicates date not set).
`smua.cal.due = calDue`	Set date when calibration should be performed (`calDue` of 0 indicates date not set).
`smua.cal.fastadc()`	Performs calibration adjustment of the fast analog-to-digital converter (ADC).
`smua.cal.lock()`	Lock out calibration.
`smua.cal.password = "newPassword"`	Change password to "`newPassword`".
`smua.cal.polarity = calPolarity`	Set polarity: smua.`CAL_AUTO` (automatic polarity). smua.`CAL_NEGATIVE` (negative polarity). smua.`CAL_POSITIVE` (positive polarity).
`smua.cal.restore(calset)`	Load set of calibration constants: smua.`CALSET_NOMINAL` (nominal constants). smua.`CALSET_FACTORY` (factory constants). smua.`CALSET_DEFAULT` (normal constants). smua.`CALSET_PREVIOUS` (previous constants).
`smua.cal.save()`	Store constants in nonvolatile memory as DEFAULT calibration set.
`calstate = smua.cal.state`	Request calibration state: smua.`CALSTATE_CALIBRATING` smua.`CALSTATE_LOCKED` smua.`CALSTATE_UNLOCKED`
`smua.cal.unlock("password")`	Unlock calibration (default password: `KI0026XX`)
`smua.measure.calibratei(range, cp1Measured, cp1Reference, cp2Measured, cp2Reference)`	Adjust current measurement range calibration*: ±`range` (measurement range to adjust). `cp1Measured` (Model 2651A measured value for cal. point 1). `cp1Reference` (reference measurement for cal. point 1). `cp2Measured` (Model 2651A measured value for cal. point 2). `cp2Reference` (reference measurement for cal. point 2).
`smua.measure.calibratev(range, cp1Measured, cp1Reference, cp2Measured, cp2Reference)`	Adjust voltage measure range calibration*: ±`range` (measurement range to adjust). `cp1Measured` (Model 2651A measured value for cal. point 1). `cp1Reference` (reference measurement for cal. point 1). `cp2Measured` (Model 2651A measured value for cal. point 2). `cp2Reference` (reference measurement for cal. point 2).
`smua.source.calibratei(range, cp1Expected, cp1Reference, cp2Expected, cp2Reference)`	Adjust current source range calibration*: ±`range` (source range to adjust). `cp1Expected` (source value programmed for cal. point 1). `cp1Reference` (reference measurement for cal. point 1). `cp2Expected` (source value programmed for cal. point 2). `cp2Reference` (reference measurement for cal. point 2).

Calibration commands	
Command	**Description**

`smua.source.calibratev(`*`range`*`,`
`  `*`cp1Expected`*`,` *`cp1Reference`*`,`
`  `*`cp2Expected`*`,` *`cp2Reference`*`)`
	Adjust voltage source range calibration*:
	±*`range`* (source range to adjust).
	`cp1Expected` (source value programmed for cal. point 1).
	`cp1Reference` (reference measurement for cal. point 1).
	`cp2Expected` (source value programmed for cal. point 2).
	`cp2Reference` (reference measurement for cal. point 2)

`smua.contact.calibratelo(`
`  `*`cp1Measured`*`,` *`cp1Reference`*`,`
`  `*`cp2Measured`*`,` *`cp2Reference`*`)`
	Adjust the low/sense low contact check measurement calibration.
	`cp1Measured` (value measured by SMU for cal. point 1).
	`cp1Reference` (reference measurement for cal. point 1).
	`cp2Measured` (value measured by SMU for cal. point 2).
	`cp2Reference` (reference measurement for cal. point 2).

`smua.contact.calibratehi(`
`  `*`cp1Measured`*`,` *`cp1Reference`*`,`
`  `*`cp2Measured`*`,` *`cp2Reference`*`)`
	Adjust the high/sense high contact check measurement calibration.
	`cp1Measured` (value measured by SMU for cal. point 1).
	`cp1Reference` (reference measurement for cal. point 1).
	`cp2Measured` (value measured by SMU for cal. point 2).
	`cp2Reference` (reference measurement for cal. point 2)

* Calibration point 1 should be performed at approximately 0% of range; calibration point 2 should be performed at approximately 90% of range. See Step sequence (on page 12-19) for calibration points.	

Adjustment procedure

Use the following procedure to perform remote calibration adjustment by sending commands over a communications interface. The remote commands and appropriate parameters are separately summarized for each step.

Step 1. Prepare the Model 2651A for adjustment

A. Connect the Model 2651A to the controller IEEE-488 interface, RS-232 port, or LAN using an appropriate interface cable.

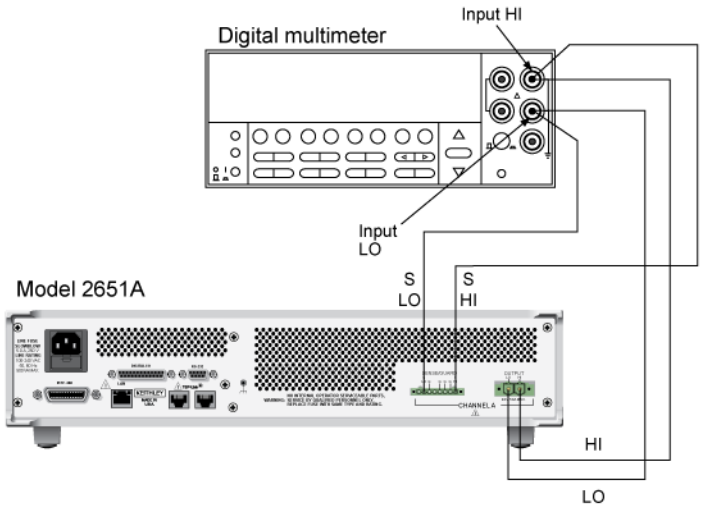
B. Turn on the Model 2651A and the test equipment. Allow them to warm up for at least two hours before performing adjustment.

C. Make sure the IEEE-488, RS-232, or LAN interface parameters are set up properly. To configure the interface, press the **MENU** key, and then select **RS232**, **LAN**, or **GPIB,** as applicable; configuration of the USB interface is not necessary so it is not available.

Step 2. Voltage calibration adjustment

A. Connect the Model 2651A SMU to the digital multimeter using the 4-wire connections shown in the figure below, and select the multimeter dc volts function.

Figure 132: Connections for voltage calibration



B. Send the following commands in order to initialize voltage calibration:

```
smua.cal.unlock("KI0026XX")
smua.reset()
smua.source.func = smua.OUTPUT_DCVOLTS
```

NOTE

It is not necessary to set the measure range when following this procedure for calibration because the measure range is locked to the source range when measuring the source function.

C. Perform each calibration adjustment for the voltage source and measure function step listed in Step sequence (on page 12-19) as follows:

1. Select the range being calibrated with this command:

   ```
   smua.source.rangev = range
   ```

2. Select the correct sense mode based on the calibration step for the voltage source and measure function from the Step sequence (on page 12-19), for example:

   ```
   smua.sense = smua.SENSE_LOCAL
   ```

3. Select positive polarity, and then set the source output to the positive zero value. For example:

   ```
   smua.cal.polarity = smua.CAL_POSITIVE
   smua.source.levelv = 1e-30
   ```

4. Turn on the output:

   ```
   smua.source.output = smua.OUTPUT_ON
   ```

5. Allow the readings to settle, then get both the multimeter and Model 2651A voltage readings at the positive zero value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

   ```
   Z_rdg = smua.measure.v()
   ```

6. Turn off the output:

   ```
   smua.source.output = smua.OUTPUT_OFF
   ```

7. Set the source output to the positive full-scale value for the present range, for example:

   ```
   smua.source.levelv = 0.9
   ```

8. Turn on the output:

   ```
   smua.source.output = smua.OUTPUT_ON
   ```

9. Allow the readings to settle, then get both the multimeter and Model 2651A voltage readings at the positive full-scale output value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

   ```
   FS_rdg = smua.measure.v()
   ```

10. Turn off the output:

    ```
    smua.source.output = smua.OUTPUT_OFF
    ```

11. Send the source calibration command using the range, +zero and +FS multimeter readings, and +zero and +FS source values for the parameters:

    ```
    smua.source.calibratev(range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
    ```

    Where:

range	= The present calibration range
src_Z	= The +zero Model 2651A programmed source output value
DMM_Z_rdg	= The +zero DMM measurement
src_FS	= The +FS Model 2651A programmed source output value
DMM_FS_rdg	= The +FS DMM measurement

    Typical values for the Model 2651A 1 V range:

    ```
    smua.source.calibratev(1, 1e-30, 1e-5, 0.9, 0.903)
    ```

12. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Model 2651A readings, and the range setting for the parameters. For example:

    ```
    smua.measure.calibratev(range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
    ```

    Where:

    range = The present calibration range
    Z_rdg = The +zero Model 2651A measurement
    DMM_Z_rdg = The +zero DMM measurement
    FS_rdg = The +FS Model 2651A measurement
    DMM_FS_rdg = The +FS DMM measurement

    Typical Model 2651A 1 V range values:

    ```
    smua.measure.calibratev(1, 1e-4, 1e-5, 0.92, 0.903)
    ```

13. Select negative polarity, then set the source output to the negative zero value, for example:

    ```
    smua.cal.polarity = smua.CAL_NEGATIVE
    smua.source.levelv = -1e-30
    ```

14. Turn on the output:

    ```
    smua.source.output = smua.OUTPUT_ON
    ```

15. Allow the readings to settle, then get both the multimeter and Model 2651A voltage readings at the negative zero value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

    ```
    Z_rdg = smua.measure.v()
    ```

16. Turn off the output:

    ```
    smua.source.output = smua.OUTPUT_OFF
    ```

17. Set the source output to the negative full-scale value, for example:

    ```
    smua.source.levelv = -0.9
    ```

18. Turn on the output:

    ```
    smua.source.output = smua.OUTPUT_ON
    ```

19. Allow the readings to settle, then get both the multimeter and Model 2651A voltage readings at the negative full-scale output value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

    ```
    FS_rdg = smua.measure.v()
    ```

20. Turn off the output:

    ```
    smua.source.output = smua.OUTPUT_OFF
    ```

21. Send the source calibration command using the range, -zero and -FS multimeter readings, and -zero and -FS source values for the parameters:

    ```
    smua.source.calibratev(-range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
    ```

Where:

-range = The negative of the present calibration range
src_Z = The −zero Model 2651A programmed source output value
DMM_Z_rdg = The −zero DMM measurement
src_FS = The −FS Model 2651A programmed source output value
DMM_FS_rdg = The −FS DMM measurement

Typical values for the Model 2651A 1 V range:

```
smua.source.calibratev(-1, -1e-30, -1e-4, -0.9, -0.896
```

22. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Model 2651A readings and range setting for the parameters:

```
smua.measure.calibratev(-range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
```

Where:

-range = The negative of the present calibration range
Z_rdg = The -zero Model 2651A measurement
DMM_Z_rdg = The -zero DMM measurement
FS_rdg = The -FS Model 2651A measurement
DMM_FS_rdg = The -FS DMM measurement

Typical Model 2651A 1 V range values:

```
smua.measure.calibratev(-1, -1e-4, -1e-6, -0.89, -0.896
```

D. Be sure to complete each of the 22 steps of C for all 11 voltage steps in Step sequence (on page 12-19) before performing current calibration.

E. Select automatic polarity mode:

```
smua.cal.polarity = smua.CAL_AUTO
```

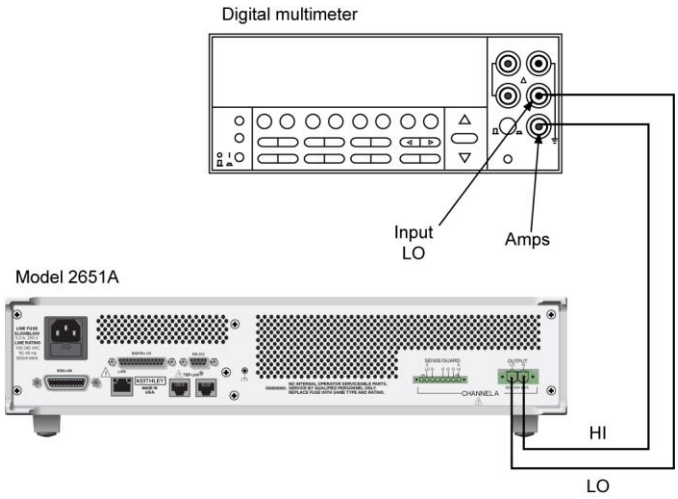
Step 3. Current calibration adjustment

A. Connect the Model 2651A SMU to the digital multimeter (see the following figure), and then select the multimeter dc current function.

B. Send this command to initialize current calibration:

```
smua.source.func = smua.OUTPUT_DCAMPS
```

Figure 133: Connections for current calibration (100 mA range and below)



C. Perform each calibration step for the current source and measure function listed in Model 2651A step sequence (on page 12-19) for the 100 mA range and below as follows:

1. Select the range being calibrated:

```
smua.source.rangei = range
```

NOTE

It is not necessary to set the measure range when following this procedure for calibration because the measure range is locked to the source range when measuring the source function.

2. Select the correct sense mode based on the calibration step for the current source and measure function listed in Model 2651A step sequence (on page 12-19), for example:

   ```
   smua.sense = smua.SENSE_LOCAL
   ```

3. Select positive polarity, then set the source output to the positive zero value:

   ```
   smua.cal.polarity = smua.CAL_POSITIVE
   smua.source.leveli = 1e-30
   ```

4. Turn on the output:

   ```
   smua.source.output = smua.OUTPUT_ON
   ```

5. Allow the readings to settle, then get both the multimeter and Model 2651A current readings at the positive zero value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

   ```
   Z_rdg = smua.measure.i()
   ```

6. Turn off the output:

   ```
   smua.source.output = smua.OUTPUT_OFF
   ```

7. Set the source output to the positive full-scale value for the present range, for example:

   ```
   smua.source.leveli = 90e-3
   ```

8. Turn on the output:

   ```
   smua.source.output = smua.OUTPUT_ON
   ```

9. Allow the readings to settle, then get both the multimeter and Model 2651A current readings at the positive full-scale output value (the Model 2651A measurement is not necessary if calibration is being done on the CALA sense mode or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

   ```
   FS_rdg = smua.measure.i()
   ```

10. Turn off the output:

    ```
    smua.source.output = smua.OUTPUT_OFF
    ```

11. Send the source calibration command using the range, zero and +FS multimeter readings, and zero and +FS source values for the parameters:

    smua.source.calibratei(*range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg*)

    Where:

    range = The present calibration range
    src_Z = The +zero Model 2651A source output value
    DMM_Z_rdg = The +zero DMM measurement
    src_FS = The +FS Model 2651A source output value
    DMM_FS_rdg = The +FS DMM measurement

    Typical values for the 100 mA range:

    ```
    smua.source.calibratei(100e-3, 1e-30, 1e-5, 90e-3, 88e-3)
    ```

12. If this step is not on the CALA sense mode or 100 A range, send the measure calibration command using the multimeter and Model 2651A readings, and range setting for the parameters:

    ```
    smua.measure.calibratei(range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
    ```

    Where:

    range = The present calibration range
    Z_rdg = +zero Model 2651A measurement
    DMM_Z_rdg = The +zero DMM measurement
    FS_rdg = +FS Model 2651A measurement
    DMM_FS_rdg = The +FS DMM measurement

    Typical 100 mA range values:

    ```
    smua.measure.calibratei(100e-3, 1e-6, 1e-5, 0.089, 0.088)
    ```

13. Select negative polarity, then set the source output to the negative zero value, for example:

    ```
    smua.cal.polarity = smua.CAL_NEGATIVE
    smua.source.leveli = -1e-30
    ```

14. Turn on the output:

    ```
    smua.source.output = smua.OUTPUT_ON
    ```

15. Allow the readings to settle, then get both the multimeter and Model 2651A current readings at the negative zero value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense or 100 A range). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

    ```
    Z_rdg = smua.measure.i()
    ```

16. Turn off the output:

    ```
    smua.source.output = smua.OUTPUT_OFF
    ```

17. Set the source output to the negative full-scale value, for example:

    ```
    smua.source.leveli = -90e-3
    ```

18. Turn on the output:

    ```
    smua.source.output = smua.OUTPUT_ON
    ```

19. Allow the readings to settle, then get both the multimeter and Model 2651A current readings at the negative full-scale output value (the Model 2651A measurement is not necessary if this calibration step is being done on the CALA sense mode). The two measurements should be made as close as possible in time. Use this command for the Model 2651A:

    ```
    FS_rdg = smua.measure.i()
    ```

20. Turn off the output:

    ```
    smua.source.output = smua.OUTPUT_OFF
    ```

21. Send the source calibration command using the -range, -zero and -FS multimeter readings, and -zero and -FS source values for the parameters:

    ```
    smua.source.calibratei(-range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)
    ```

    Where:

-range	= The negative of the present calibration range
src_Z	= The zero Model 2651A source output value
DMM_Z_rdg	= The zero DMM measurement
src_FS	= The FS Model 2651A source output value
DMM_FS_rdg	= The FS DMM measurement

    Typical values for the 100 mA range:

    ```
    smua.source.calibratei(-100e-3, -1e-30, -1e-6, -90e-3, -89.2e-3)
    ```

22. If this step is not on the CALA sense mode or 100 A range, send the measure calibration command using the multimeter and Model 2651A readings, and range setting for the parameters:

    ```
    smua.measure.calibratei(-range, Z_rdg, DMM_Z_rdg, FS_rdg, DMM_FS_rdg)
    ```

    Where:

-range	= The negative of the present calibration range
Z_rdg	= The zero Model 2651A measurement
DMM_Z_rdg	= The zero DMM measurement
FS_rdg	= The FS Model 2651A measurement
DMM_FS_rdg	= The FS DMM measurement

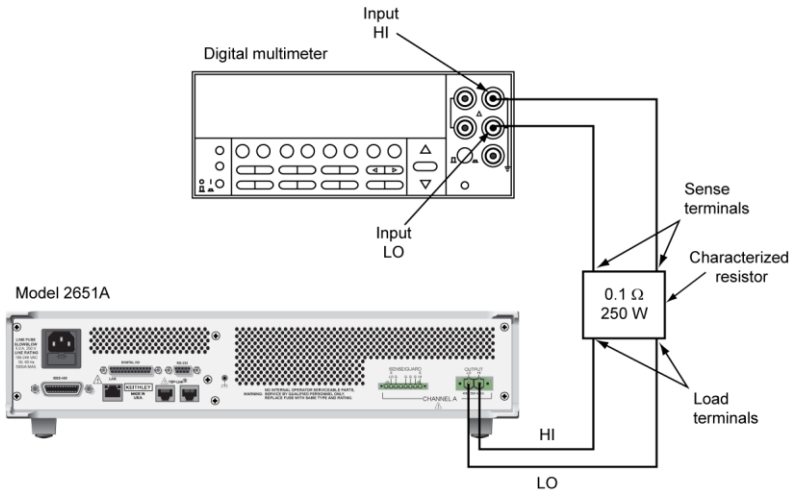
    Typical 100 mA range values:

    ```
    smua.measure.calibratei(-100e-3, -1e-5, -1e-6, -91e-3, -89.2e-3)
    ```

D. Before continuing, be sure to complete each of the 22 steps of C for all the source and measure ranges 100 mA and below for the current source and measure function listed in the Model 2651A step sequence (on page 12-19).

E. Change connections as shown in the following figure.

Figure 134: Connections for current calibration (1 A range and above)



F. Select the DMM dc volts function.

G. Repeat the 22 steps of C for the 1 A, 5 A, 10 A, 20 A, 50 A, and 100 A ranges. Compute the current reading from the DMM voltage reading and characterized 0.1 Ω resistance value: I = V/R.

H. Select automatic polarity mode:

```
smua.cal.polarity = smua.CAL_AUTO
```

Step 4. Fast ADC calibration adjustment

Perform fast analog-to-digital converter (ADC) calibration adjustment:

```
smua.cal.fastadc()
```

NOTE

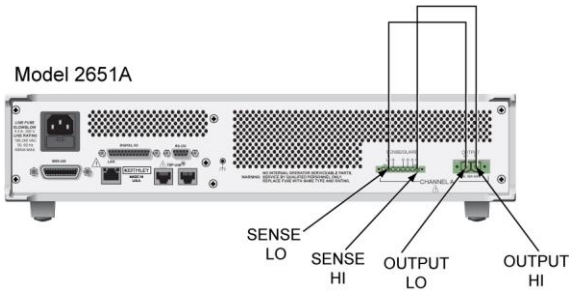
Make sure both the voltage and current calibration steps have been completed before calling the `smua.cal.fastadc()` function.

Step 5. Contact check calibration adjustment

A. As illustrated in the following figure:

- Short the Model 2651A SENSE LO and LO terminals together.
- Short the SENSE HI and HI terminals together.

Figure 135: Connections for contact check 0 ohm calibration



B. Allow the readings to settle, then get the Model 2651A readings:

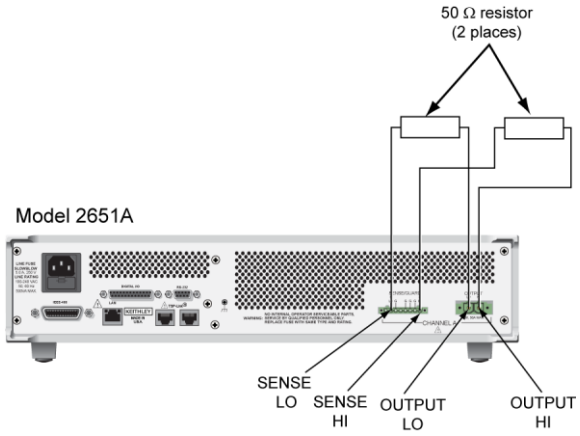
```
r0_hi, r0_lo = smua.contact.r()
```

C. Characterize both 50 Ω resistors using the resistance function of the digital multimeter.

D. As illustrated in the following figure:

- Connect a 50 Ω resistor between the SENSE LO and LO terminals.
- Connect the second 50 Ω resistor between the SENSE HI and HI terminals.

Figure 136: Connections for contact check 50 ohm calibration



E. Allow the readings to settle, then get the Model 2651A readings:

```
r50_hi, r50_lo = smua.contact.r()
```

F. Send the contact check low calibration adjustment command:

```
smua.contact.calibratelo(r0_lo, Z_actual, r50_lo, 50_ohm_actual)
```

Where:

r0_lo	=	Model 2651A 0 Ω low measurement
Z_actual	=	Actual zero value; the resistance of the short between the SENSE LO and LO terminals
r50_lo	=	Model 2651A 50 Ω low measurement
50_ohm_actual	=	Actual 50 Ω resistor value; the actual value of the resistor between the SENSE LO and LO terminals

Typical values:

```
smua.contact.calibratelo(r0_lo, 0, r50_lo, 50.15)
```

Where `r0_lo` is the same value as measured in step B, and `r50_lo` is the same value as measured in step D.

G. Send the contact check high calibration command:

```
smua.contact.calibratehi(r0_hi, Z_actual, r50_hi, 50_ohm_actual)
```

Where:

r0_hi	=	Model 2651A 0 Ω high measurement
Z_actual	=	Actual zero value; the resistance of the short between the SENSE HI and HI terminals
r50_hi	=	Model 2651A 50 Ω high measurement
50_ohm_actual	=	Actual 50 Ω resistor value; the value of the resistor between the SENSE HI and HI terminals

Typical values:

```
smua.contact.calibratehi(r0_hi, 0, r50_hi, 50.15)
```

Where r0_hi is the same value as measured in step B, and r50_hi is the same value as measured in step D.

Step 6. Program calibration dates

Use the following command to set the calibration adjustment date:

```
smua.cal.adjustdate = os.time{year=2019, month=12, day=1}
```

Optionally, it is possible to set the calibration date and calibration due date with the following commands:

```
smua.cal.date = os.time{year=2019, month=12, day=1}
```

```
smua.cal.due = os.time{year=2020, month=12, day=1}
```

If you do not wish to set a calibration date or calibration due date and want to clear the previous values, use the following commands:

```
smua.cal.date = 0
```

```
smua.cal.due = 0
```

The actual year, month, day, and (optional) hour and minute should be used (seconds can be given but are essentially ignored due to the precision of the internal date storage format). The allowable range for the year is from 1970 to 2037, the month is from 1 to 12, and the day is from 1 to 31.

Step 7. Save calibration constants

Calibration adjustment is now complete, so you can store the calibration constants in nonvolatile memory by sending the following command:

```
smua.cal.save()
```

NOTE

Unless you send the save command, the calibration adjustment you just performed is temporary.

Step 8. Lock out calibration

To lock out further calibration adjustment, send the following command after completing the adjustment procedure:

```
smua.cal.lock()
```

Section 13

LAN concepts and settings

In this section:

Overview

This section describes how to connect to the LAN.

The Model 2651A is version 1.5 LXI Device Specification 2016 compliant. The Model 2651A is a scalable test system that can connect directly to a host computer or interact with a DHCP or DNS server and other LXI-compliant instruments on a local area network (LAN). The Model 2651A also supports Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are useful on a LAN with no central administration.

The Model 2651A is compliant with the IEEE Std 802.3 and supports full connectivity on a 10 or 100 megabits-per-second network. The LAN interface is an alternative to GPIB that can be used to build flexible test systems that include web access.

NOTE

Please read this entire section before you connect the Model 2651A to the LAN.

Establishing a point-to-point connection

To enable access to the instrument web interface and other web applications from a computer, use a one-to-one LAN connection and set up a static IP address between the host computer and the instrument.

The following instructions describe how to configure the IP address of the instrument. The IP address of the instrument is based on the present IP address of the host computer. Each device on the LAN (corporate or private) requires a unique IP address.

CAUTION

Contact your corporate information technology (IT) department for permission before you connect the Model 2651A to a corporate network.

If you have problems, see LAN troubleshooting suggestions (on page 8-8).

NOTE

Record all network configurations before modifying any existing network configuration information on the network interface card. Once the network configuration settings are updated, the previous information is lost. This may cause a problem reconnecting the host computer to a corporate network, particularly if DHCP Enabled = NO (disabled).

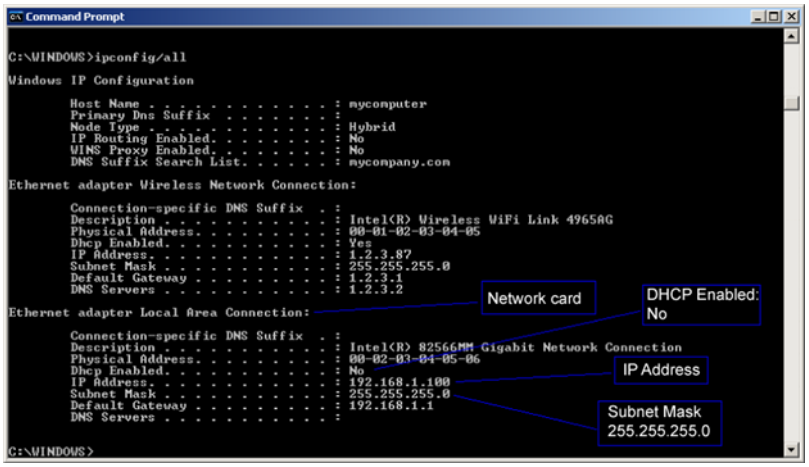
Be sure to return all settings to their original configuration before reconnecting the host computer to a corporate network. Failure to do this could result in loss of data. Contact your system administrator for more information.

Step 1: Identify and record the existing IP configuration

To identify the existing IP configuration:

1. Open the command prompt window.
2. At the command prompt, type `ipconfig/all` and press the **Enter** key. A list of existing IP configuration information for your computer is displayed.

Figure 137: Computer IP configuration using the command prompt



NOTE

If the information for the ethernet adapter displays `Media Disconnected`, close the command prompt and go to Step 2: Disable DHCP to use the existing computer IP address (on page 13-3).

3. When the information is displayed, record the following information for the network card:
   - DHCP mode: _____
   - IP address: _____
   - Subnet mask: _____
   - Default gateway: _____
   - DNS servers: _____

CAUTION

The `ipconfig/all` command displays the configuration of every network card. Make sure that you record the information for the proper network card.

4. If:
   - **DHCP Enabled = Yes:** Go to Step 2: Disable DHCP to use the existing computer IP address (on page 13-3)
   - **DHCP Enabled = No:** Go to Step 3: Configure the LAN settings of the instrument (on page 13-4).
5. To exit the IP configuration screen, type **exit** at the command prompt and press **Enter**.

Step 2: Disable DHCP to use the existing computer IP address

NOTE

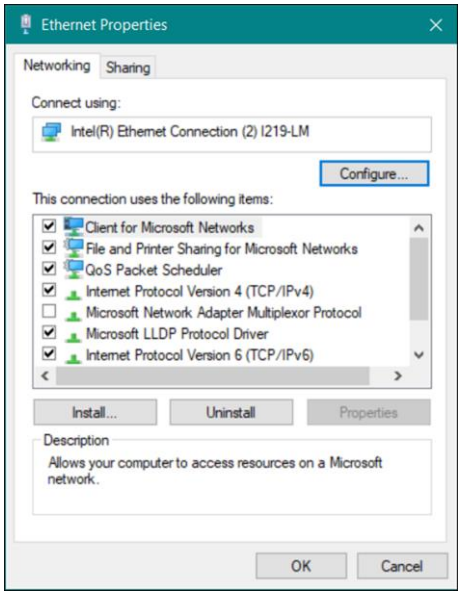
Do not change the IP address at any time without talking to your system administrator. Entering an incorrect IP address can prevent your workstation from connecting to your corporate network.

See the appropriate instructions below for your operating system. These instructions show the default options. Be aware that there may be differences in these steps if your Microsoft Windows options are customized or if you do not have administrator status.

To disable DHCP:

1. From the Start menu, select **View Network Connections**.
2. Right-click **Ethernet** and select **Properties**.
3. Select **Internet Protocol Version 6**.

Figure 138: Ethernet networking properties



4. Select **Properties**.
5. Select **Use the following IPv6 address**. The option for "Use the following DNS server addresses" is automatically selected.
6. Set the IP address. If the IP address and subnet mask fields:
   - **Contain values:** Record the IP address, subnet mask, default gateway, and DNS servers to use in Step 3: Configure the LAN settings of the instrument (on page 13-4).
   - **Are blank:** In the IP address field, enter `192.168.1.100`. In the subnet mask field, enter `255.255.255.0`. These are used to configure the LAN settings of the instrument.
7. Click **OK** to close the Internet Protocol (TCP/IP) Properties dialog box.
8. Click **Close** to close the Ethernet Properties dialog box.
9. Close the Network Connections window.

Step 3: Configure the LAN settings of the instrument

NOTE

These steps assume that you are making all the settings in the order shown here. If you only change one or a few settings, be aware that you need to apply the settings before they are in effect. To apply the settings, from the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To configure the Model 2651A using the front panel:

1. Press the **MENU** key to display the MAIN MENU.
2. Use the navigation wheel to select **LAN**. The LAN CONFIG menu is displayed.

3. Change the IP address assignment method:
   a. Select **CONFIG > METHOD > MANUAL**, and then press the **ENTER** key.
   b. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
4. Enter the IP address using the LAN CONFIG menu:
   a. Select **CONFIG > IP-ADDRESS**.
   b. Refer to the recorded computer's IP address (Step 1: Identify and record the existing IP configuration (on page 13-2)). A portion of the computer's IP address is used as a base for the instrument's unique ID. Only the last three numbers (after the last decimal point) of the IP address differ between the computer and the instrument. If the subnet mask is 255.255.255.0, the last three digits can be any value from 1 to 255.

      For example, the Internet Protocol (TCP/IP) Properties dialog box shows that the computer's IP address is 192.168.1.100 (see the figure titled "Internet protocol (TCP/IP) Properties dialog box" in Step 2: Disable DHCP to use the existing computer IP address (on page 13-3)). A unique IP address for the instrument might be 192.168.001.101.

NOTE

The IP address of the instrument can have leading zeros, but the IP address of the computer cannot.

   c. Use the navigation wheel to select and enter an appropriate IP address for the instrument. Be sure to record the IP address to use in Step 4: Access the web interface of the instrument (on page 13-6).
   d. Press **ENTER** key or navigation wheel to confirm the changes.
   e. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
5. Change the subnet mask from the LAN CONFIG menu:
   a. Select **CONFIG > SUBNETMASK**, and then press the **ENTER** key. The SUBNETMASK menu item is to the right of GATEWAY. Use the navigation wheel to scroll through the options.
   b. Modify the SUBNETMASK value to match the computer settings recorded earlier (or `255.255.255.000` if DHCP Enabled = YES).
   c. Press the **ENTER** key or the navigation wheel when you are finished changing all the characters.
   d. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
6. From the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

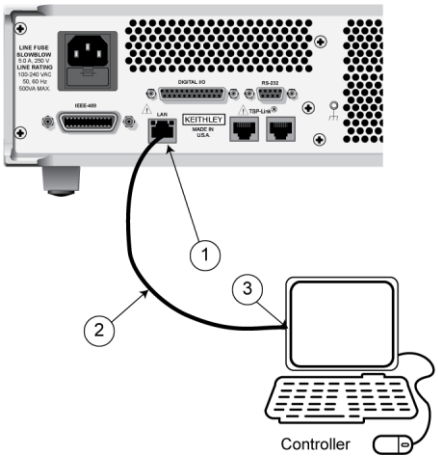
Step 4: Install the crossover cable

Connect the supplied crossover cable between the computer's network interface card and the LAN connector on the instrument’s rear panel. There are multiple connectors on the Model 2651A rear panel. Be sure to connect to the LAN connection port (see the following figure).

NOTE

Connect the crossover cable into the same computer LAN port used during instrument configuration to ensure that the system is using the correct network card.

Figure 139: LAN connection



1. Model 2651A LAN connection port
2. Crossover cable
3. Ethernet port (located on the host computer)

Step 5: Access the web interface of the instrument

1. Open a web browser on the host computer.
2. Enter the IP address of the instrument in the web browser address box. For example, if the instrument IP address is 192.168.1.101, enter `192.168.1.101` in the browser address box.
3. Press **Enter** on the computer keyboard to open the web interface of the instrument.

NOTE

If the web interface does not open in the browser, see LAN troubleshooting suggestions (on page 8-8).

Connecting to the LAN

Each device on the LAN (corporate or private) requires a unique IP address. Contact your corporate information technology (IT) department for details about obtaining an IP address before you deploy the Model 2651A on a corporate or private network.

NOTE

Contact your corporate IT department for permission before you connect the Model 2651A to a corporate network.

Setting the LAN configuration method

There are two methods used to configure the LAN.

AUTO: Use the AUTO setting to allow the DHCP server to automatically set the LAN settings.

You do not need to set the LAN options manually. The DHCP server automatically configures the IP address, subnet mask, and the default gateway. To use this option, a DHCP server must be available on the LAN.

MANUAL: Use the MANUAL setting to manually configure the communication parameters.

The MANUAL setting requires you to configure the following:

- IP address
- Gateway
- Subnet mask

To select a LAN configuration method:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > METHOD**.
2. Select either **AUTO** or **MANUAL**.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key until you return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Setting the IP address

NOTE

Contact your corporate information technology (IT) department to secure a valid IP address for the instrument when placing the instrument on a corporate network.

To set the IP address when LAN configuration method is set to MANUAL:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > IP-ADDRESS**.
2. Turn the navigation wheel to select and enter a valid IP address for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Setting the gateway

NOTE

Contact your corporate information technology (IT) department to secure a valid gateway for the instrument when placing the instrument on a corporate network.

To set the gateway when LAN configuration method is set to MANUAL:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > GATEWAY**.
2. Turn the navigation wheel to select and enter a valid gateway address for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Setting the subnet mask

NOTE

Contact your corporate information technology (IT) department to secure a valid subnet mask for the instrument when placing the instrument on a corporate network.

To set the subnet mask when LAN configuration method is set to MANUAL:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > SUBNETMASK**.
2. Turn the navigation wheel to select and enter a valid subnet mask for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Configuring the domain name system (DNS)

The Domain Name System (DNS) lets you type a domain name in the address bar to connect to the instrument. If you use DNS, you can use a name instead of an IP address.

Example:

```
Model2651A.XYZcompany.com
```

NOTE

Contact your corporate information technology (IT) department for information about DNS. If a DNS server is not part of the LAN infrastructure, do not use this setting.

To enable or disable DNS host name verification:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > DNS > VERIFY**.
2. Turn the navigation wheel to select either **ENABLE** or **DISABLE**. When enabled, the instrument performs a DNS lookup to verify the DNS host name matches the value specified in the lan.config.dns.hostname (on page 7-143) attribute.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To enable or disable DNS registration:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > DNS > DYNAMIC**.
2. Turn the navigation wheel to select either **ENABLE** or **DISABLE**. DNS registration works with the DHCP to register the host name specified in the `lan.config.dns.hostname` attribute with the DNS server.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To set the DNS server IP addresses:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > DNS**.
2. Turn the navigation wheel to select either **DNS-ADDRESS1** or **DNS-ADDRESS2**.
3. Press the **ENTER** key.
4. Turn the navigation wheel to select and enter a valid IP address for the DNS server.
5. Press the **ENTER** key.
6. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
7. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

LAN speeds

Another characteristic of the LAN is speed. The Model 2651A negotiates with the host computer and other LXI-compliant devices on the LAN to transmit data at the highest speed possible. LAN speeds must be configured to match the speed of the other instruments on the network.

To set the LAN speed:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > SPEED**.
2. Turn the navigation wheel to select either **10 Mbps** or **100 Mbps.**
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the previous menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Duplex mode

The duplex mode is based on the LAN configuration. There are two settings:

- **Half-duplex:** Allows communications in both directions, but only one direction is active at a time (not simultaneously).
- **Full:** Permits communications in both directions simultaneously.

To set the duplex mode:

1. From the front panel, press **MENU** key and select **LAN > CONFIG > DUPLEX**.
2. Turn the navigation wheel to select either **HALF** or **FULL.**
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Viewing LAN status messages

To view the LAN status messages:

1. From the front panel, press the **MENU** key and select **LAN > STATUS > CONFIG/FAULT**.
2. Press the **ENTER** key.

Figure 140: LAN CONFIG/FAULT

There are two types of LAN status messages:

- **LAN fault messages:** Communicate issues related to physical connectivity.
- **LAN configuration messages:** Communicate issues or events related to configuration.

The following table displays possible fault and configuration messages.

LAN CONFIG/FAULT messages

LAN message type	Possible messages
LAN fault	Could not acquire IP address
	Duplicate IP address detected
	DHCP lease lost
	Lan Cable Disconnected
LAN configuration	Starting DHCP Configuration
	DHCP Server Not Found
	DHCP configuration started on xxx.xxx.xxx.xxx
	Searching for DNS server(s)
	Starting DLLA Configuration
	DLLA Failed
	DLLA configuration started on xxx.xxx.xxx.xxx
	Starting Manual Configuration
	Manual configuration started on xxx.xxx.xxx.xxx
	Closed

Viewing the network settings

To view the active network settings:

1. From the front panel, press the **MENU** key, and then select **LAN > STATUS**.
2. Use the navigation wheel to select one of the following network settings:
   - **IP-ADDRESS**
   - **GATEWAY**
   - **SUBNET-MASK**
   - **METHOD**
   - **DNS**
   - **MAC-ADDRESS**
3. Press the **ENTER** key to view the active setting.
4. Press the **EXIT (LOCAL)** key once to return to the STATUS menu.

Confirming the active speed and duplex negotiation

The Model 2651A automatically detects the speed and duplex negotiation active on the LAN. Once the speed and duplex negotiation is detected, the instrument automatically adjusts its own settings to match the LAN settings.

To confirm the active LAN speed and duplex mode:

1. From the front panel, press the **MENU** key.
2. Select **LAN > STATUS**.
3. Use the navigation wheel to select one of the following:
   - **SPEED**
   - **DUPLEX**
4. Press the **ENTER** key to view the active setting.
5. Press the **EXIT (LOCAL)** key once to return to the STATUS menu.

To set the duplex mode:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > DUPLEX**.
2. Turn the navigation wheel to select either **HALF** or **FULL.**
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

Confirming port numbers

To view the port number assigned to each remote interface protocol:

1. From the front panel, press the **MENU** key, and then select **LAN > STATUS > PORT**.
2. Use the navigation wheel to select one of the following:
   - **RAW-SOCKET**
   - **TELNET**
   - **VXI-11**
   - **DST**
3. Press the **ENTER** key to view the port number.
4. Press the **EXIT (LOCAL)** key once to return to the PORT menu.

The following table displays the remote interface protocols supported by the Model 2651A and their assigned port numbers.

Port number

Command interface	Port number
Raw socket	5025
Telnet	23
VXI-11	1024
DST (dead socket termination)	5030

Selecting a LAN interface protocol

You can use a remote interface protocol to connect to the Model 2651A. The Model 2651A provides Telnet, VXI-11, and raw socket LAN interfaces, with associated LAN protocols (each interface uses a different protocol). Select the interface based on the protocol needed.

You can also use a dead socket termination interface (DST) to troubleshoot connection problems.

NOTE

You can only use one remote interface at a time. Although multiple ethernet connections to the instrument can be opened, only one can be used to control the instrument at a time.

VXI-11 connection

This remote interface is similar to GPIB and supports message boundaries, serial poll, and service requests (SRQs). A VXI-11 driver or NI-VISA software is required. Test Script Builder (TSB) uses NI-VISA and can be used with the VXI-11 interface. You can expect a slower connection with this protocol.

Raw socket connection

All Keithley instruments that have LAN connections support raw socket communication. This means that you can connect to the TCP/IP port on the instrument and send and receive commands. A programmer can easily communicate with the instrument using the Winsock API on computers with the Microsoft® Windows® operating system or using the Berkeley Sockets API on Linux® or Apple® computers.

Dead socket connection

The dead socket termination (DST) port is used to terminate all existing ethernet connections. A dead socket is a socket that is held open by the instrument because it has not been properly closed. This most often happens when the host computer is turned off or restarted without first closing the socket. This port cannot be used for command and control functions.

Use the dead socket termination port to manually disconnect a dead session on any open socket. All existing ethernet connections are terminated and closed when the connection to the dead socket termination port is closed.

Telnet connection

The Telnet protocol is similar to raw socket and can be used when you need to interact directly with the instrument. Telnet is often used for debugging and troubleshooting. You need a separate Telnet program to use this protocol.

The Model 2651A supports the Telnet protocol, which you can use over a TCP/IP connection to send commands to the instrument. You can use a Telnet connection to interact with scripts or send real-time commands.

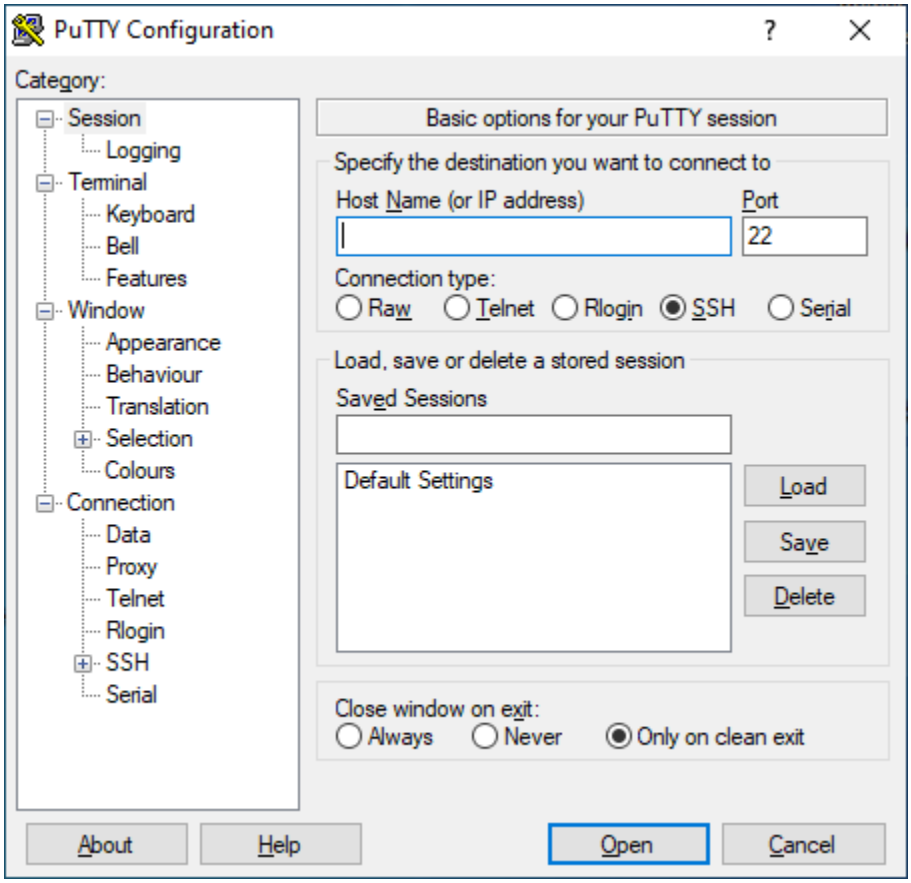
Configuring a Telnet connection

This procedure uses PuTTY, which is open source, cross-platform, and usable under the MIT license. Consult the PuTTY help or user manual for other usage concerns not covered in this document.

To connect with the Model 2651A using PuTTY on a Windows system:

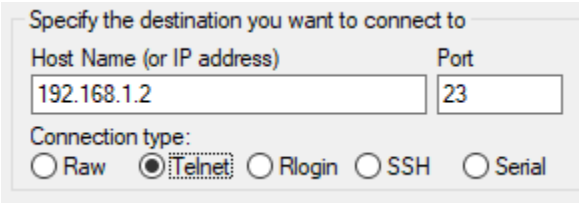
1. On the host computer, open PuTTY. The PuTTY Configuration dialog box opens.

Figure 141: PuTTY configuration description dialog box



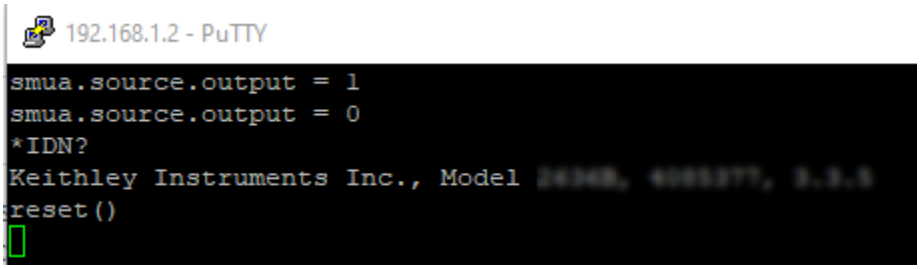
2. In **Host Name (or IP address)**, enter the instrument IP address, such as `192.168.1.101`.
3. In Port, enter **23**.
4. For Connection Type, select **Telnet**.

Figure 142: Telnet connection settings example



5. Select **Open** to start the Telnet session.
6. Use PuTTY to interact directly with the instrument.

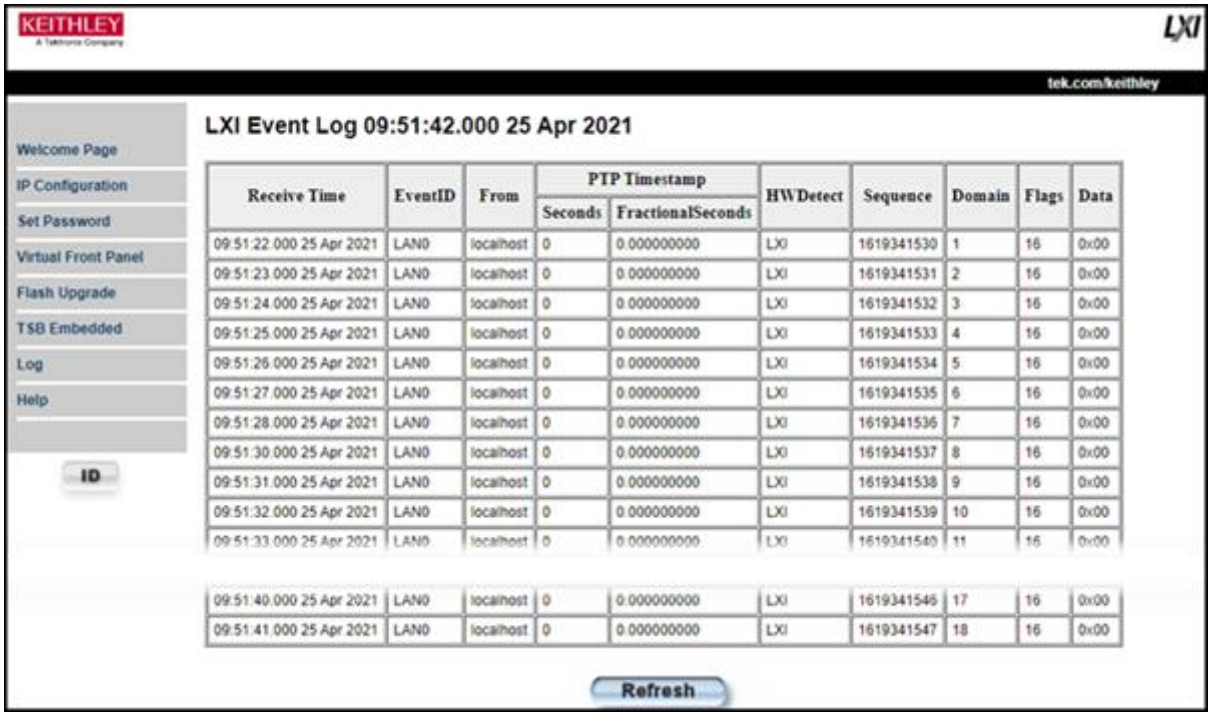
Figure 143: Communicating with the instrument using PuTTY



Logging LAN trigger events in the event log

You can use the event log to record all LXI triggers generated and received by the Model 2651A. You can view the event log using any command interface or the embedded web interface. The following figure shows the view of the LXI event log from the Log option in the embedded web interface.

Figure 144: LXI Event Log



Receive Time	EventID	From	PTP Timestamp		HWDetect	Sequence	Domain	Flags	Data
			Seconds	FractionalSeconds					
09:51:22.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341530	1	16	0x00
09:51:23.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341531	2	16	0x00
09:51:24.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341532	3	16	0x00
09:51:25.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341533	4	16	0x00
09:51:26.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341534	5	16	0x00
09:51:27.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341535	6	16	0x00
09:51:28.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341536	7	16	0x00
09:51:30.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341537	8	16	0x00
09:51:31.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341538	9	16	0x00
09:51:32.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341539	10	16	0x00
09:51:33.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341540	11	16	0x00
09:51:40.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341546	17	16	0x00
09:51:41.000 25 Apr 2021	LAN0	localhost	0	0.000000000	LXI	1619341547	18	16	0x00

The timestamp, event identifier, IP address, and the domain name identify the incoming and outgoing LXI trigger packets. The following table provides detailed descriptions for the columns in the event log.

Event log descriptions

Column title	Description	Example
Receive Time	Displays the date and time that the LAN trigger occurred in UTC, 24-hour time	`11:46:44.000 11 Mar 2020`

Event ID	Identifies the `lan.trigger[N]` that generates an event	`LAN0 = lan.trigger[1]`
		`LAN1 = lan.trigger[2]`
		`LAN2 = lan.trigger[3]`
		`LAN3 = lan.trigger[4]`
		`LAN4 = lan.trigger[5]`
		`LAN5 = lan.trigger[6]`
		`LAN6 = lan.trigger[7]`
		`LAN7 = lan.trigger[8]`

From	Displays the IP address for the device that generates the LAN trigger	`localhost` `192.168.5.20`
Timestamp	A timestamp that identifies the time the event occurred; the timestamp uses the following: ■ PTP timestamp ■ Seconds ■ Fractional seconds; the Model 2651A does not support the IEEE Std 1588 standard; the values in this field are always 0 (zero)	
HWDetect	Identifies a valid LXI trigger packet	`LXI`
Sequence	Each instrument maintains independent sequence counters: ■ One for each combination of UDP multicast network interface and UDP multicast destination port ■ One for each TCP connection	
Domain	Displays the LXI domain number; the default value is 0 (zero)	`0`
Flags	Contain data about the LXI trigger packet; values are: ■ 1 - Error ■ 2 - Retransmission ■ 4 - Hardware ■ 8 - Acknowledgments ■ 16 - Stateless bit	`16`
Data	The values for this are always 0 (zero)	

Accessing the event log from the command interface

You can access the event log from any remote command interface. The event log must be enabled before LXI trigger events can be viewed. To enable the event log, send:

```
eventlog.enable = 1
```

To view the event log from a remote interface, send:

```
print(eventlog.all())
```

This command outputs one or more strings similar to the following:

```
14:14:02.000 17 Jun 2019, LAN0, 10.80.64.191, LXI, 0, 1560780842, not available, 0,
   0x10,0x00
```

The string displays the same information as the web interface. Commas separate the fields. The fields output in the following order:

- Received time (UTC time)
- Event ID
- From (Sender)
- HWDetect / version
- Domain
- Sequence number
- Timestamp (PTP time)
- Epoch (from 1588)
- Flags
- Data

See the table in Logging LAN trigger events in the event log (on page 13-17) for detailed descriptions.

To generate log traffic, send the code:

```
local id = 1
lan.trigger[id].ipaddress = lan.status.ipaddress
lan.trigger[id].connect()
for domain = 1, 255 do
   print(domain)
   lan.lxidomain = domain
   lan.trigger[id].assert()
   delay(1)
end
```

Section 14

Common commands

In this section:

Common command summary

The IEEE Std 488.2 common commands that are supported by the Model 2651A are summarized in the following table. Although commands are shown in uppercase, common commands are not case sensitive, so you can use either uppercase or lowercase. Although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the Model 2651A does not strictly conform to that standard.

NOTE

Unlike other commands, like those listed in TSP commands (on page 7-7), each common command must be sent in a separate message.

The common commands cannot be used in scripts.

Command	Name	Description
`*CLS`	Clear status	Clears all event registers and Error Queue. For detailed information including status commands, see the Status model (on page 15-1).
`*ESE` *mask*	Event enable command	Program the Standard Event Status Enable Register. For detailed information including status commands, see the Status model (on page 15-1).
`*ESE?`	Event enable query	Read the Standard Event Status Enable Register. For detailed information including status commands, see the Status model (on page 15-1).
`*ESR?`	Event status register query	Read/clear the Standard Event Enable Register. For detailed information including status commands, see the Status model (on page 15-1).
`*IDN?`	Identification query	Returns the manufacturer, model number, serial number, and firmware revision levels of the unit. For detailed information, see Identification query: *IDN? (on page 14-3).
`*OPC`	Operation complete command	Set the Operation Complete bit in the Standard Event Register after all pending commands, including overlapped commands, have completed. For detailed information, see Operation complete and query: *OPC and *OPC? (on page 14-4).
`*OPC?`	Operation complete query	Places an ASCII "1" into the output queue when all selected device operations have completed. For detailed information, see Operation complete and query: *OPC and *OPC? (on page 14-4).
`*RST`	Reset command	Returns the Model 2651A to default conditions. For detailed information, see Reset: *RST (on page 14-4).
`*SRE` *mask*	Service request enable command	Programs the Service Request Enable Register. For detailed information including status commands, see the Status model (on page 15-1).
`*SRE?`	Service request enable query	Reads the Service Request Enable Register. For detailed information including status commands, see the Status model (on page 15-1).
`*STB?`	Status byte query	Reads the status byte register. For detailed information including status commands, see the Status model (on page 15-1).
`*TRG`	Trigger command	Generates the `trigger.EVENT_ID` trigger event for use with the trigger model. For detailed information, see Trigger: *TRG (on page 14-4).
`*TST?`	Self-test query	Returns a 0. For detailed information, see Self-test query: *TST? (on page 14-4).
`*WAI`	Wait-to-continue command	Waits until all previous commands have completed. For detailed information, see Wait-to-continue: *WAI (on page 14-5).

Script command equivalents

The TSP commands that can be included in scripts that are equivalent to the common commands are defined in the table below.

Common command	Script command equivalent
`*CLS`	`status.reset()`
`*ESE?`	`print(tostring(status.standard.enable))`
`*ESE` *mask*	`status.standard.enable =` *mask*
`*ESR?`	`print(tostring(status.standard.event))`
`*IDN?`	`print([[Keithley Instruments, Model]]..localnode.model..[[, ]]..localnode.serialno.. [[, ]]..localnode.revision)`
`*OPC?`	`waitcomplete() print([[1]])`
`*OPC`	`opc()`
`*RST`	`reset()`
`*SRE?`	`print(tostring(status.request_enable))`
`*SRE` *mask*	`status.request_enable =` *mask*
`*STB?`	`print(tostring(status.condition))`
`*TRG`	Not available
`*TST?`	`print([[0]])`
`*WAI`	`waitcomplete()`

Command reference

Details of all common commands (except those associated with the status model) are described below.

NOTE

Status command usage is in the Status model (on page 15-1).

Identification query: *IDN?

Retrieves the identification string.

`*IDN?` Command that reads ID information

The identification string includes the manufacturer, model number, serial number, and firmware revision levels. This string is sent in the following format:

```
Keithley Instruments, Model 2651A, xxxxxxx, yyyyy
```

Where:

xxxxxxx is the serial number

yyyyy is the firmware revision level

Operation complete and query: *OPC and *OPC?

Wait for pending overlapped commands to complete.

`*OPC` Operation complete command that sets the OPC bit

`*OPC?` Operation complete query that places a "`1`" in the output queue

When `*OPC` is sent, the OPC bit in the Standard Event Register (see Status model (on page 15-1)) is set when all overlapped commands complete. The `*OPC?` command places an ASCII "`1`" in the output queue when all previous overlapped commands complete.

Reset: *RST

Returns the instrument to default conditions.

`*RST` Command that returns the instrument to default conditions

When the `*RST` command is sent, the instrument returns to the default conditions. This performs the same actions as reset() (on page 7-197).

Self-test query: *TST?

Requests self-test results.

`*TST?` Places a zero (`0`) in the output queue

This command always places a zero (`0`) in the output queue. This command is included for common command compatibility only; the Model 2651A does not actually perform a self-test.

Trigger: *TRG

Generates a command interface trigger event for the trigger model.

`*TRG` This command generates the `trigger.EVENT_ID` trigger event for the trigger model

The `trigger.EVENT_ID` is a constant that contains the command interface trigger event number. You can set the stimulus of any trigger object to the value of this constant to have the trigger object respond to the trigger events generated by this command. See trigger.EVENT_ID (on page 7-410) and Using the remote trigger model (on page 3-38).

Wait-to-continue: *WAI

Suspends the execution of subsequent commands until all previous overlapped commands are finished.

`*WAI` This pauses until overlapped commands are complete

Two types of device commands exist:

- **Overlapped commands.** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands.** Commands whose operations finish before the next command is executed.

The `*WAI` command suspends the execution of subsequent commands until the instrument operations of all previous overlapped commands are finished. The `*WAI` command is not needed for sequential commands.

General bus commands

General commands are commands that have the same general meaning, regardless of the instrument (for example, DCL). The following table lists the general bus commands.

General bus commands

Command	Effect on Model 2651A
`DCL`	Returns the Model 2651A and all devices on the GPIB to known conditions. See DCL (on page 2-87) for details.
`GET`	Initiates a trigger. See GET (on page 2-87) for details.
`GTL`	Cancel remote; restore Model 2651A front-panel operation. See GTL (on page 2-87) for details.
`IFC`	Goes into talker and listener idle states. See IFC (on page 2-86) for details.
`LLO`	LOCAL key locked out. See LLO (on page 2-86) for details.
`REN`	Goes into remote operation when next addressed to listen. See REN (on page 2-86) for details.
`SDC`	Returns the Model 2651A to known conditions. See SDC (on page 2-87) for details.
`SPE`, `SPD`	Serial polls the Model 2651A. See SPE, SPD (on page 2-88) for details.

REN

The remote enable (REN) command is sent to the Model 2651A by the controller to set up the instrument for remote operation. Generally, place the instrument in the remote mode before you attempt to program it over the bus. Setting REN to true does not place the instrument in the remote state. You must address the instrument to listen after setting REN to true before it goes into remote operation.

IFC

The interface clear (IFC) command is sent by the controller to place the Model 2651A in the talker idle state and the listener idle state. The instrument responds to the IFC command by canceling illumination of the front-panel TALK or LSTN lights if the instrument was previously placed in one of these states.

Transfer of command messages to the instrument and transfer of response messages from the instrument are not interrupted by the IFC command. If transfer of a response message from the instrument was suspended by IFC, transfer of the message resumes when the instrument is addressed to talk. If transfer of a command message to the instrument was suspended by the IFC command, the rest of the message can be sent when the instrument is addressed to listen.

LLO

When the instrument is in remote operation, all front-panel controls are disabled, except the LOCAL and OUTPUT OFF keys (and the POWER switch). The local lockout (LLO) command disables the LOCAL key, but does not affect the OUTPUT OFF switch, which cannot be disabled.

GTL

Use the go to local (GTL) command to put a remote-mode instrument into local mode. Leaving the remote state also restores operation of all front-panel controls.

DCL

Use the device clear (DCL) command to clear the GPIB interface and return it to a known state. The DCL command is not an addressed command, so all instruments equipped to implement DCL are returned to a known state simultaneously.

When the Model 2651A receives a DCL command, it:

- Clears the input buffer, output queue, and command queue
- Cancels deferred commands
- Clears any command that prevents the processing of any other device command

The DCL command does not affect instrument settings and stored data.

SDC

The selective device clear (SDC) command is an addressed command that performs essentially the same function as the device clear (DCL) command. However, because each device must be individually addressed, the SDC command provides a method to clear only selected instruments, instead of clearing all instruments simultaneously with the DCL command.

When the Model 2651A receives an SDC command, it:

- Clears the input buffer, output queue, and command queue
- Cancels deferred commands
- Clears any command that prevents the processing of any other device command

An SDC call does not affect instrument settings and stored data.

GET

The group execute trigger (GET) command is a GPIB trigger that triggers the instrument to make readings from a remote interface.

SPE, SPD

Use the serial polling sequence to obtain the Model 2651A serial poll byte. The serial poll byte contains important information about internal functions (see Status model (on page 15-1)). Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line. The serial polling sequence may be performed at any time to obtain the status byte from the Model 2651A.

Section 15

Status model

In this section:

Overview

Each Keithley Instruments Model 2651A provides status registers and queues that are collectively referred to as the status model. Through manipulation and monitoring of these registers and queues, you can view and control various instrument events. You can include commands in your test program that can determine if a service request (SRQ) event has occurred and the cause of the event.

The heart of the status model is the Status Byte Register. All status model registers and queues flow into the Status Byte Register.

The entire status model is illustrated in the Status model diagrams (on page 15-5).

Status register set contents

Typically, a status register set contains the following registers:

- **Condition** (`.condition`): A read-only register that is constantly updated to reflect the present operating conditions of the instrument.
- **Enable Register** (`.enable`): A read-write register that allows a summary bit to be set when an enabled event occurs.
- **Event Register** (`.event`): A read-only register that sets a bit to 1 when the applicable event occurs. If the enable register bit for that event is also set, the summary bit of the register is set to 1.
- **Negative Transition Register (NTR)** (`.ntr`): When a bit is set in this read-write register, it enables a 1 to 0 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.

- **Positive Transition Register (PTR)** (`.ptr`): When a bit is set in this read-write register, it enables a 0 to 1 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.

An event is represented by a condition register bit changing from a 1 to 0 or 0 to 1. When an event occurs and the appropriate NTR or PTR bit is set, the corresponding event register bit is set to 1. The event bit remains latched to 1 until the event register is read or the status model is reset. When an event register bit is set and its corresponding enable bit is set, the summary bit of the register is set to 1. This, in turn, sets a bit in a higher-level condition register, potentially cascading to the associated summary bit of the Status Byte Register.

Queues

The Model 2651A uses queues to store messages. The queues include:

- Command queue: Holds commands that are available for execution.
- Output queue: Holds response messages.
- Error queue: Holds error and status messages.

When a queue contains data, it sets the condition bit for that queue in one of the registers. The condition bits are:

- Command queue: CAV in the Operation Status Remote Summary Register
- Output queue: MAV in the Status Byte Register
- Error queue: EAV in the Status Byte Register

The CAV, MAV, and EAV bits in the registers are cleared when the queue is empty. Queues empty when:

- Commands are executed
- Errors are read from the error queue
- Response messages are read from the instrument

All Model 2651A queues are first-in, first-out (FIFO).

The Status byte and service request enable registers (on page 15-5) shows how the queues are structured with the other registers.

Command queue

The command queue holds commands that have been received from a remote interface that are available for execution. This allows the Model 2651A to accept multiple commands and queue them for execution.

When a command is received from a remote interface, the command available (CAV) bit in the Operation Status Remote Summary Register is set. For additional detail, see status.operation.remote.* (on page **Error! Bookmark not defined.**).

Output queue

Response messages, such as those generated from print commands, are placed in the output queue. All remote command interfaces share the same output queue.

The output queue sets the message available (MAV) bit in the status model.

The data in the output queue is cleared by the `*CLS` command.

Error queue

The error queue holds error and status messages. As programming errors and status messages occur, a message that defines the error or status is placed in the error queue.

An error or status message is cleared from the error queue when it is read. You can also clear the error queue by sending the command `errorqueue.clear()`. An empty error queue clears the error available (EAV) bit in the Status Byte Register.

Messages in the error queue include a code number, message text, severity, and TSP-Link® node number. See Error summary list (on page 8-3) for a list of the messages.

When you read a single message from the error queue, the oldest message is read. When empty, the error number 0 and “No Error” is placed in the queue.

The commands used to control the error queue are listed in the following table.

Error queue commands

Error queue command	Description
`errorqueue.clear()`	Clear error queue of all errors.
`errorqueue.count`	Number of messages in the error/event queue.
`errorCode, message, severity, errorNode = errorqueue.next()`	Request error code, text message, severity, and TSP-Link node number.

Status function summary

The following functions and attributes control and read the various registers. Additional information for the various register sets is included later in this section. Also, refer to the specific command as listed in TSP commands (on page 7-7).

Status function summary

Type	Function or attribute
Status summary	status.condition status.node_enable status.node_event status.request_enable status.request_event status.reset
Measurement event	status.measurement.* status.measurement.buffer_available.* status.measurement.current_limit.* status.measurement.instrument.* status.measurement.instrument.smua.* status.measurement.reading_overflow.* status.measurement.voltage_limit.*
Operation status	status.operation.* status.operation.calibrating.* status.operation.instrument.* status.operation.instrument.digio.* status.operation.instrument.digio.trigger_overrun.* status.operation.instrument.lan.* status.operation.instrument.lan.trigger_overrun.* status.operation.instrument.smua.* status.operation.instrument.smua.trigger_overrun.* status.operation.instrument.trigger_blender.* status.operation.instrument.trigger_blender.trigger_overrun.* status.operation.instrument.trigger_timer.* status.operation.instrument.trigger_timer.trigger_overrun.* status.operation.instrument.tsplink.* status.operation.instrument.tsplink.trigger_overrun.* status.operation.measuring.* status.operation.remote.* status.operation.sweeping.* status.operation.trigger_overrun.* status.operation.user.*
Questionable status	status.questionable.* status.questionable.calibration.* status.questionable.instrument.* status.questionable.instrument.smua.* status.questionable.over_temperature.* status.questionable.unstable_output.*
Standard event	status.standard.*
System summary	status.system.* status.system2.* status.system3.* status.system4.* status.system5.*
* = .condition, .event, .ntr, .ptr and .enable;	

Status model diagrams

The following figures graphically describe the status model:

Figure 145: Status byte and service request enable registers

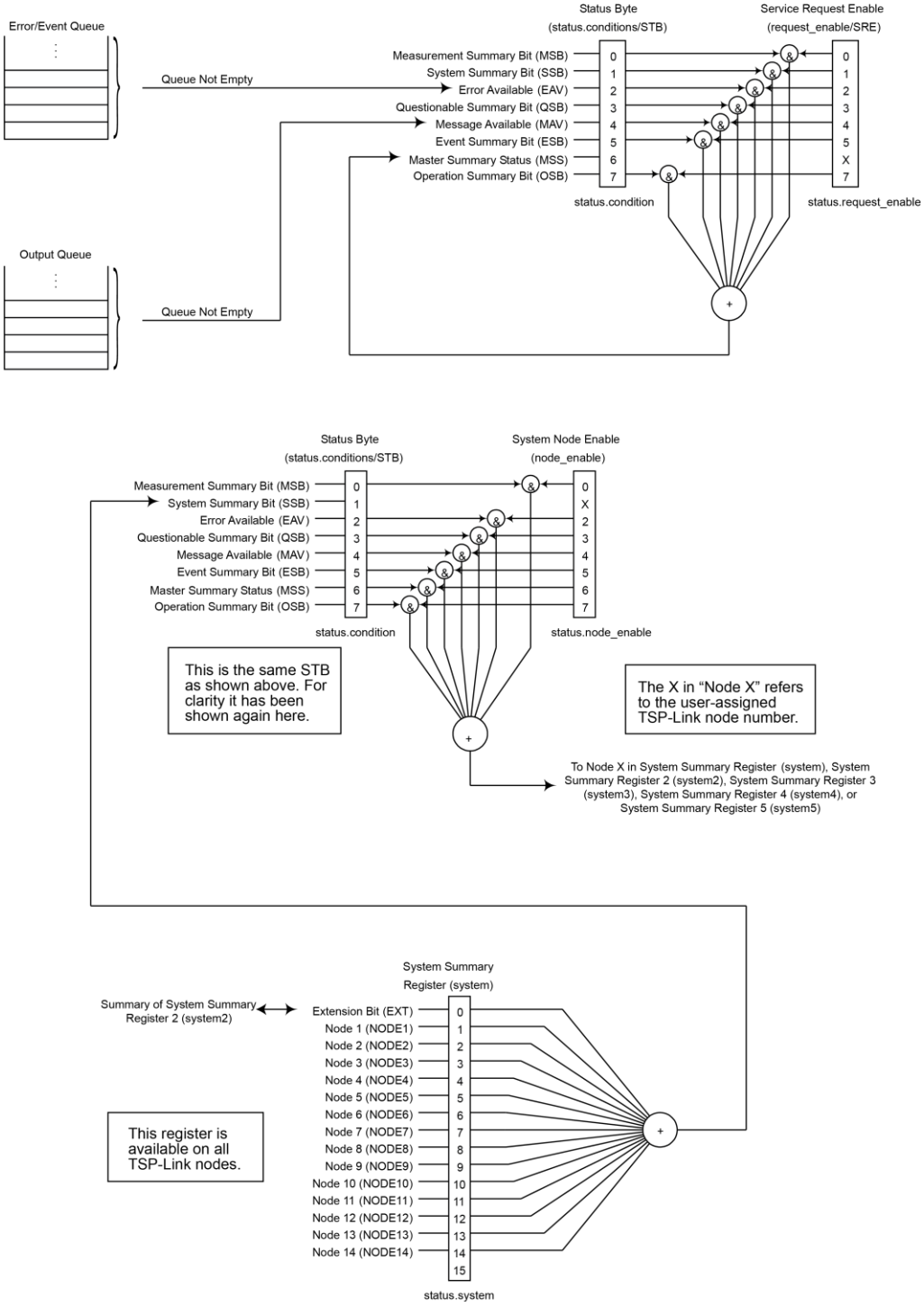


Figure 146: System summary and standard event registers

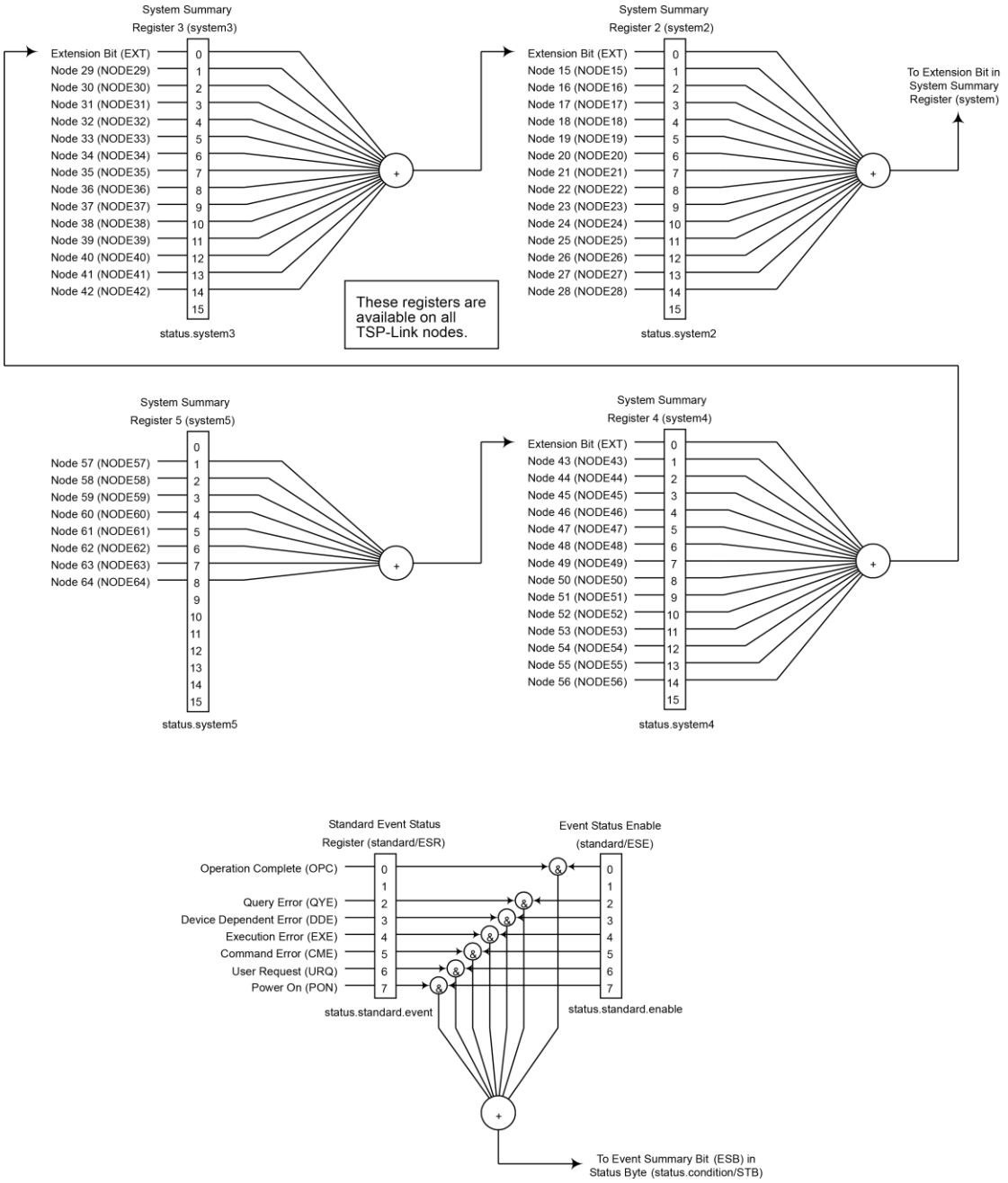


Figure 147: Measurement event registers

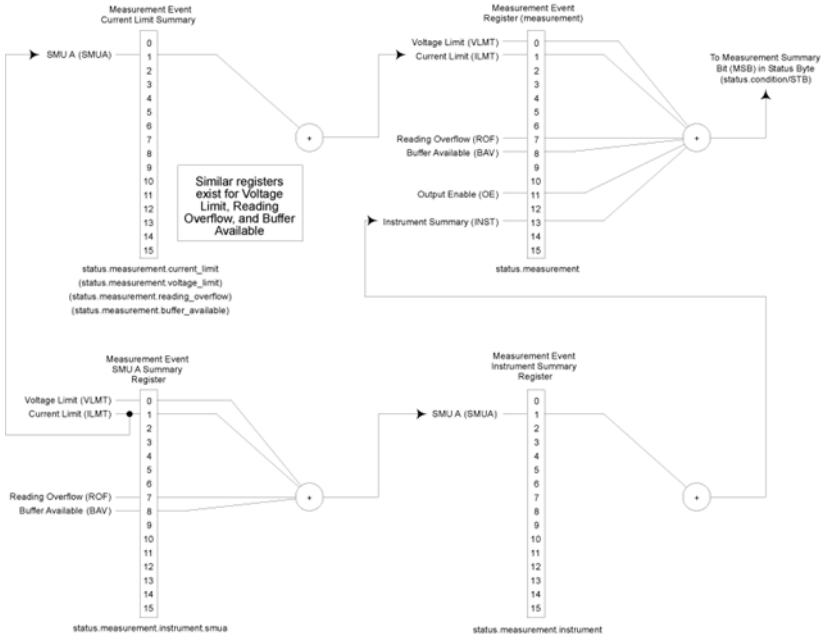


Figure 148: Operation status registers

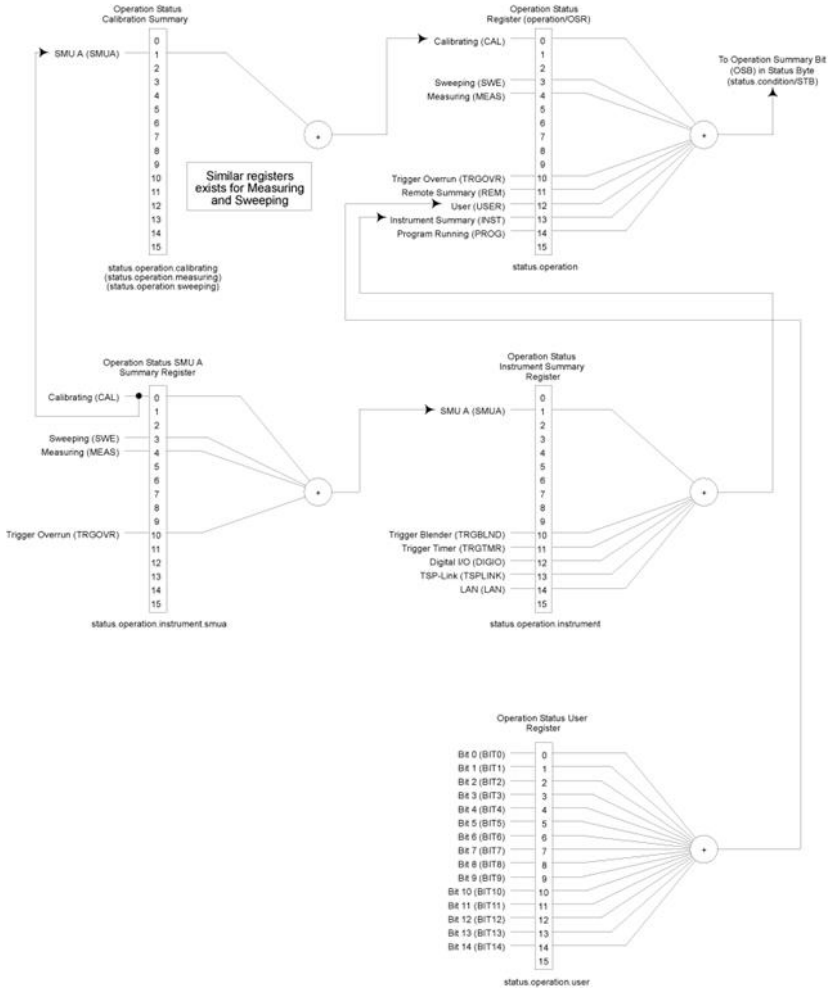


Figure 149: Operation status trigger overrun registers

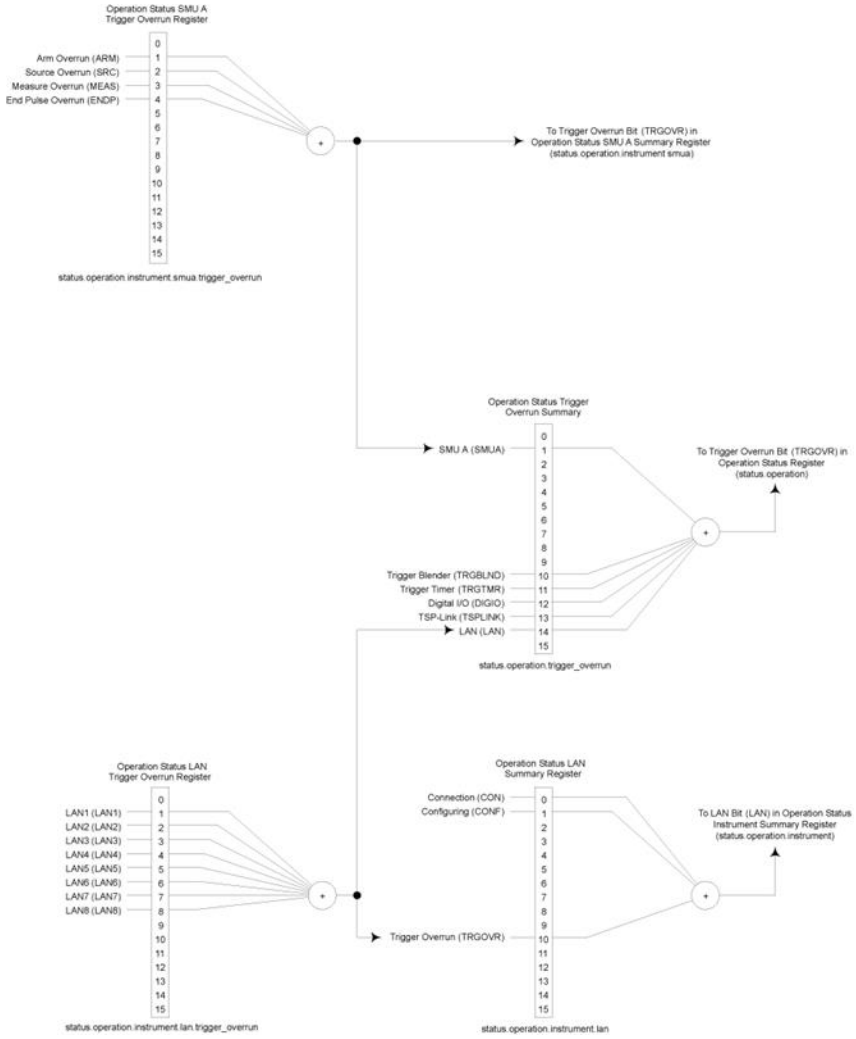


Figure 150: Operation status trigger timer, trigger blender, and remote registers

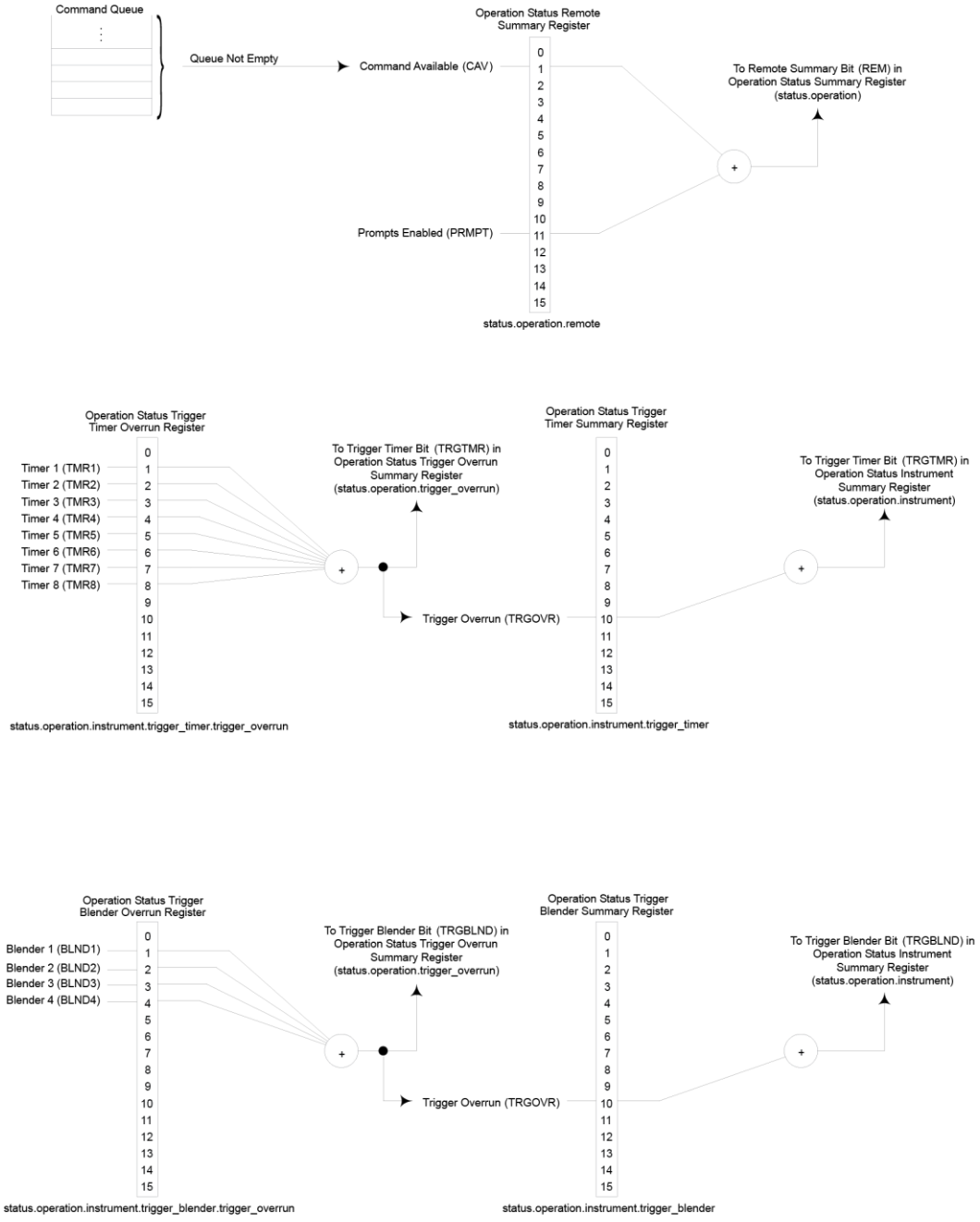


Figure 151: Operation status digital I/O and TSP-Link registers

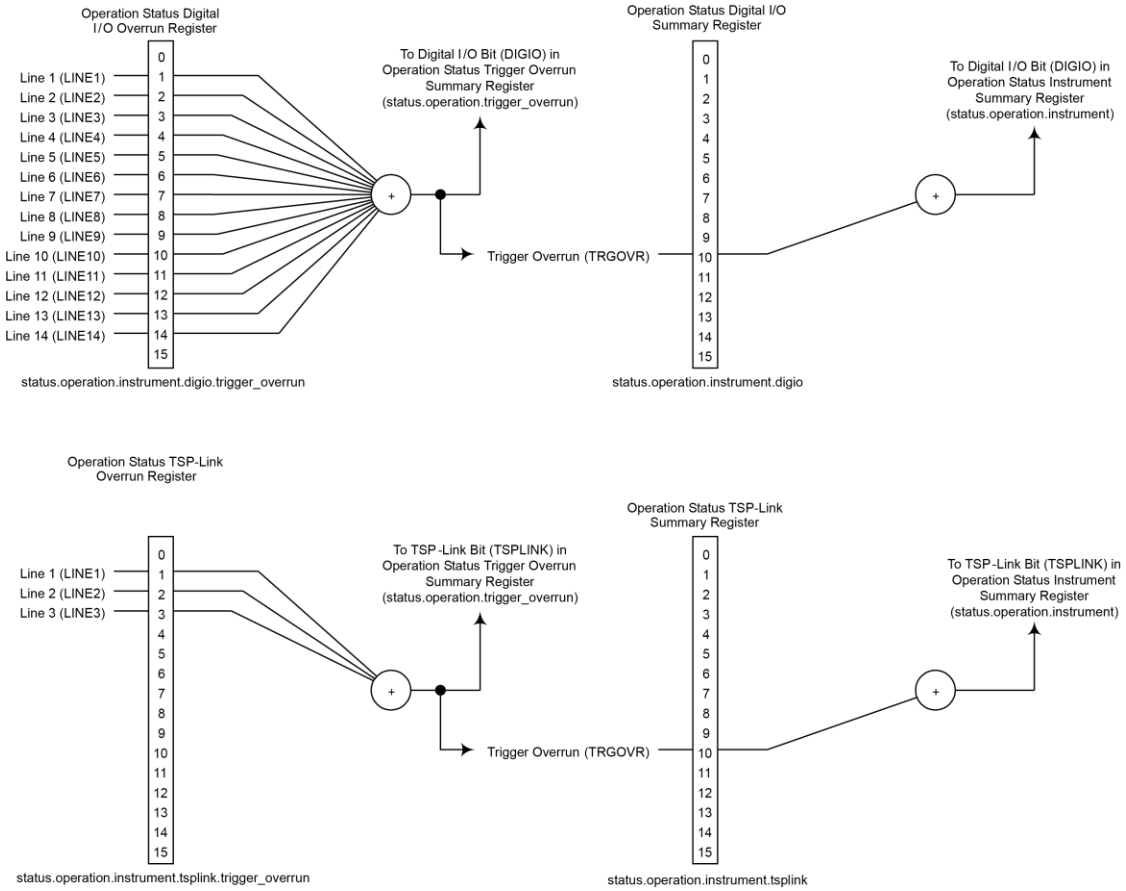
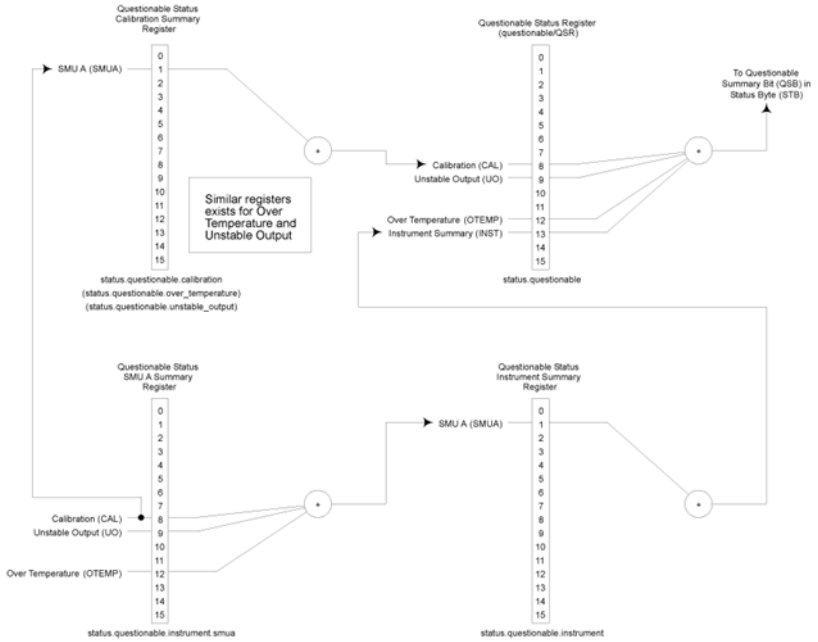


Figure 152: Questionable status registers



Clearing registers

You can use commands to reset the status registers.

`*CLS` resets the bits of the event and NTR registers to 0 and sets all PTR register bits on. This command also clears the output queue.

`status.reset()` resets bits of the event and NTR registers to 0 and sets all PTR register bits on. Refer to status.reset() (on page 7-380) for additional information.

In addition to these commands, you can reset the enable registers and the NTR to 0. To do this, send the individual command to program the register with a 0 as its parameter value. The PTR registers can be reset to their defaults by programming them with all bits on. The event registers are not programmable but you can clear them by reading them.

Programming and reading registers

The only registers that you can program are the enable and transition registers. All other registers in the status structure are read-only registers. The following explains how to determine the parameter values for the various commands used to program enable registers. The actual commands are summarized in Common commands (on page 14-1) and Status function summary (on page 15-4).

Programming enable and transition registers

A command to program an event enable or transition register is sent with a parameter value that determines the state (0 or 1) of each bit in the appropriate register. The bit positions of the register (see the following tables) indicate the binary parameter value and decimal equivalent. To program one of the registers, send the decimal value for the bits to be set. The registers are discussed further in Enable and transition registers (on page 15-19).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

When using a numeric parameter, registers are programmed by including the appropriate *mask* value. For example:

```
*ese 1169
status.standard.enable = 1169
```

To convert from decimal to binary, use the information shown in the above figure. For example, to set bits B0, B4, B7, and B10, use a decimal value of 1169 for the mask parameter (1169 = 1 + 16 + 128 + 1024).

Reading registers

Any register in the status structure can be read either by sending the common command query (where applicable), or by including the script command for that register in either the `print`() or `print(tostring())` command. The `print()` command outputs a numeric value; the `print(tostring())` command outputs the string equivalent. For example, any of the following commands requests the Service Request Enable Register value:

```
*SRE?
print(tostring(status.request_enable))
print(status.request_enable)
```

The response message is a decimal value that indicates which bits in the register are set. That value can be converted to its binary equivalent using the information in Programming enable and transition registers. For example, for a decimal value of 37 (binary value of 100101), bits B5, B2, and B0 are set.

Status byte and service request (SRQ)

Service requests (SRQs) allow an instrument to indicate that it needs attention or that some event has occurred. When the controller receives an SRQ, it allows the controller to interrupt tasks to perform other tasks in order to address the request for service.

For example, you might program your instrument to send an SRQ when:

- All instrument operations are complete
- An instrument error occurs
- A specific operation has occurred

Two 8-bit registers control service requests: The Status Byte Register and the Service Request Enable Register. See Status Byte Register (on page 15-15) for a description of the structure of these registers.

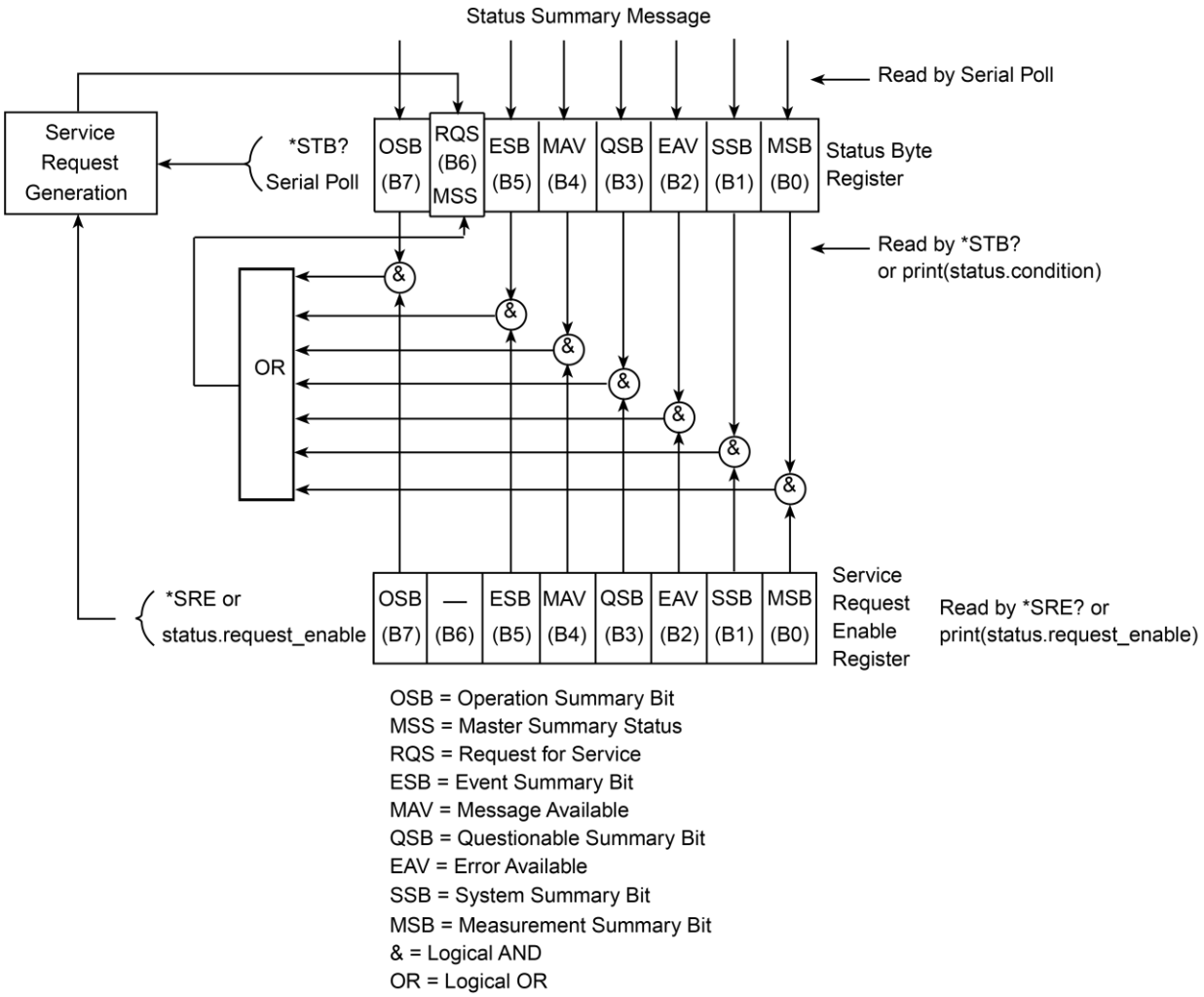
Service requests affect GPIB and VXI-11 connections. On a GPIB connection, the SRQ line is asserted. On a VXI-11 connection, an SRQ event is generated.

Status Byte Register

The summary messages from the status registers and queues are used to set or clear the appropriate bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are dependent upon the summary messages (0 or 1). For example, if the Standard Event Register is read, its register is cleared. As a result, its summary message resets to 0, which then 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, or MSS, bit.

Figure 153: Status byte and service request (SRQ)



The bits of the Status Byte Register are described as follows:

- **Bit B0, Measurement Summary Bit (MSB):** Set summary bit indicates that an enabled measurement event has occurred.
- **Bit B1, System Summary Bit (SSB):** Set summary bit indicates that an enabled system event has occurred.
- **Bit B2, Error Available (EAV):** Set bit indicates that an error or status message is present in the error queue.
- **Bit B3, Questionable Summary Bit (QSB):** Set summary bit indicates that an enabled questionable event has occurred.
- **Bit B4, Message Available (MAV):** Set bit indicates that a response message is present in the output queue.
- **Bit B5, Event Summary Bit (ESB):** Set summary bit indicates that an enabled standard event has occurred.

- **Bit B6, Request Service (RQS)/Master Summary Status (MSS):** Set bit indicates that an enabled summary bit of the Status Byte Register is set. Depending on how it is used, bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit:
  - When using the GPIB or VXI-11 serial poll sequence of the Model 2651A to obtain the status byte (serial poll byte), B6 is the RQS bit. See Serial polling and SRQ (on page 15-18) for details on using the serial poll sequence.
  - When using the `*STB?` common command or status.condition Status byte and service request commands (on page 15-19) to read the status byte, B6 is the MSS bit.
- **Bit B7, Operation Summary (OSB):** Set summary bit indicates that an enabled operation event has occurred.

Service Request Enable Register

The Service Request Enable Register controls the generation of a service request. This register is programmed by the user and is used to enable or disable the setting of bit B6 (RQS/MSS) by the Status Summary Message bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in Status Byte Register (on page 15-15), a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic "1" output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

You can set or clear the individual bits of the Service Request Enable Register by using the `*SRE` common command or `status.request_enable`. To read the Service Request Enable Register, use the `*SRE?` query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a `*SRE 0` or `status.request_enable = 0` is sent). The commands to program and read the SRQ Enable Register are listed in Status byte and service request commands (on page 15-19).

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ occurred and what caused it. If an SRQ occurred, the program can, for example, branch to an appropriate subroutine that services the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

The serial poll does not clear the low-level registers that caused the SRQ to occur. You must clear the low-level registers explicitly. Refer to Clearing registers (on page 15-13).

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit remains set until all enabled Status Byte Register summary bits are reset.

SPE, SPD (serial polling)

For the GPIB interface only, the SPE and SPD general bus commands are used to serial poll the High Power System SourceMeter® Instrument (see General bus commands (on page 2-86)). Serial polling obtains the serial poll byte (status byte). Typically, serial polling is used by the controller to determine which of several instruments has requested service with the SRQ line.

Status byte and service request commands

The commands to program and read the Status Byte Register and Service Request Enable Register are listed in the table below. Note that the table includes both common commands and their script command equivalents. For details on programming and reading registers, see Programming enable and transition registers and Reading registers (on page 15-14).

To reset the bits of the Service Request Enable Register to 0, use 0 as the parameter value for the command (for example, `*SRE 0` or `status.request_enable = 0`).

Status Byte and Service Request Enable Register commands

Command	Description

`*STB?`
or
`print(status.condition)`
	Read the Status Byte Register.

`*SRE mask`
or
`status.request_enable = mask`
	Program the Service Request Enable Register where *mask* = 0 to 255.

`*SRE?`
or
`print(status.request_enable)`
	Read the Service Request Enable Register.

Enable and transition registers

In general, there are three types of user-writable registers that are used to configure which bits feed the register summary bit and when it occurs. The registers are identified in each applicable command (as listed in TSP commands (on page 7-7)) as follows:

- **Enable register** (identified as `.enable` in the command listing of each attribute): Allows various associated events to be included in the summary bit for the register.
- **Negative-transition register** (identified as `.ntr` in the command listing of each attribute): A particular bit in the event register is set when the corresponding bit in the NTR is set, and the corresponding bit in the condition register transitions from 1 to 0.
- **Positive-transition register** (identified as `.ptr` in the command listing of each attribute): A particular bit in the event register is set when the corresponding bit in the PTR is set, and the corresponding bit in the condition register transitions from 0 to 1.

Controlling node and SRQ enable registers

Attributes to control system node and service request (SRQ) enable bits and read associated registers are summarized in the Status byte and service request enable registers (on page 15-5). For example, either of the following commands set the system node QSB enable bit:

```
status.node_enable = status.QSB
status.node_enable = 8
```

Status register sets

There are five status register sets in the status structure of a High Power System SourceMeter® Instrument:

- System Summary
- Standard Event Status
- Operation Status
- Questionable Status
- Measurement Event

System Summary Registers

As shown in Status model diagrams (on page 15-5), there are five register sets associated with system status events. These registers summarize the system status for various nodes connected to the TSP-Link® network (see TSP-Link system expansion interface). Note that all nodes on the TSP-Link network share a copy of the system summary registers once the TSP-Link system has been initialized. This feature allows all nodes to access the status models of other nodes, including service request (SRQ).

In a TSP-Link system, you can configure the status model so that a status event in any node in the system can set the RQS (request for service) bit of the Master Node Status Byte. See TSP-Link system status (on page 15-26) for details on using the status model in a TSP-Link system.

Commands for the system summary registers are summarized in the Status function summary (on page 15-4) table.

For example, either of the following commands sets the EXT enable bit:

```
status.system.enable = status.system.EXT
status.system.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Reading registers (on page 15-14). For example, the following command reads the System Enable Register:

```
print(status.system.enable)
```

The used bits of the system event registers are described as follows:

- **Bit B0, Extension Bit (EXT):** Set bit indicates that an extension bit from another system status register is set.
- **Bits B1 to B14 NODE*n*:** Indicates a bit on TSP-Link node *n* has been set (*n* = 1 to 64) (note that `status.system5` does not use bits B9 through B15).

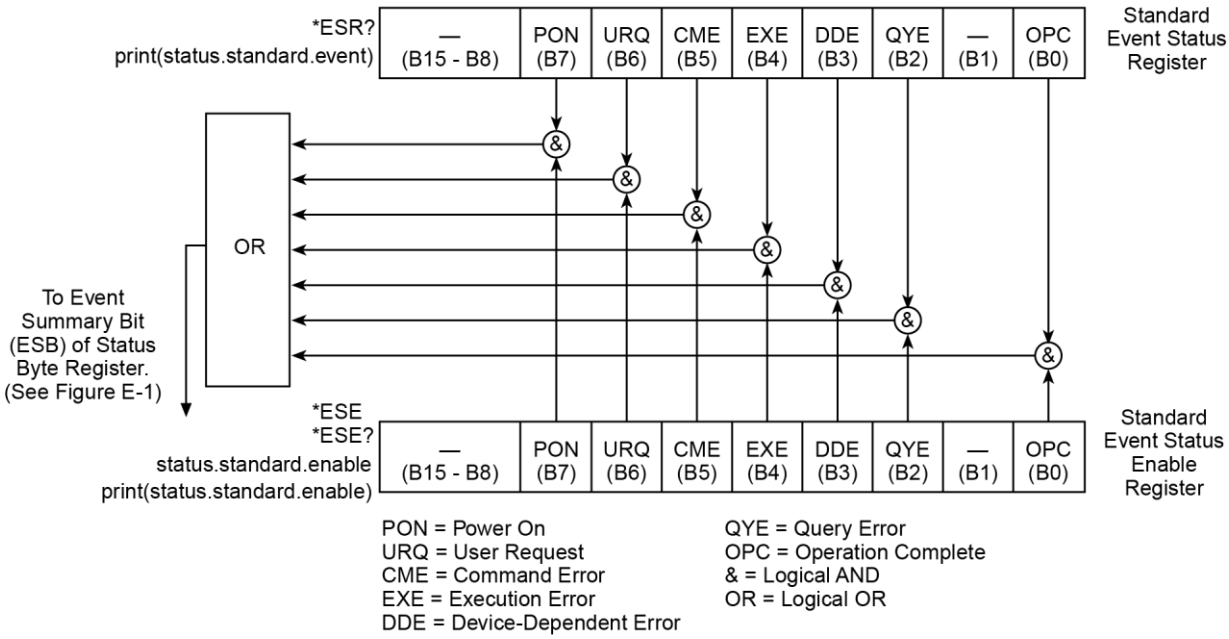
Standard Event Register

The bits used in the Standard Event Register are described as follows:

- **Bit B0, Operation Complete (OPC):** Set bit indicates that all pending selected device operations are completed and the Model 2651A instrument is ready to accept new commands. The bit is set in response to an *OPC command. The `opc()` function can be used in place of the *OPC command. See Common commands (on page 14-1) for details on the *OPC command.
- **Bit B1:** Not used.
- **Bit B2, Query Error (QYE):** Set bit indicates that you attempted to read data from an empty output queue.
- **Bit B3, Device-Dependent Error (DDE):** Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, Execution Error (EXE):** Set bit indicates that the Model 2651A instrument detected an error while trying to execute a command.
- **Bit B5, Command Error (CME):** Set bit indicates that a command error has occurred. Command errors include:
  - IEEE Std 488.2 syntax error: The Model 2651A instrument received a message that does not follow the defined syntax of IEEE Std 488.2.
  - Semantic error: Model 2651A instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented.
  - The instrument received a Group Execute Trigger (GET) inside a program message.
- **Bit B6, User Request (URQ):** Set bit indicates that the LOCAL key on the Model 2651A instrument front panel was pressed.
- **Bit B7, Power ON (PON):** Set bit indicates that the Model 2651A instrument has been turned off and turned back on since the last time this register was read.

Commands to program and read the register are summarized below and also in the Status function summary (on page 15-4) table.

Figure 154: Standard Event Register



Standard event commands

Command	Description

`*ESR?`
or
`print(status.standard.event)`
	Read Standard Event Status Register.

`*ESE <mask>`
or
`status.standard.enable = <mask>`
	Program the Event Status Enable Register:
	`<mask>` = 0 to 255
	See Status register set contents (on page 15-1).

`*ESE?`
or
`print(status.standard.enable)`
	Read Event Status Enable Register.

Operation Status Registers

As shown in the Operation status registers (on page 15-9) diagram of the status model, there are 22 register sets associated with operation status. Commands are summarized in Status register set contents (on page 15-1). You can also set bits using numeric parameter values. For details, see Programming enable and transition registers.

For example, either of the following commands sets the CAL enable bit (B0):

```
status.operation.enable = status.operation.CAL
status.operation.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Reading registers (on page 15-14). For example, the following command reads the Operation Status Enable Register:

```
print(status.operation.enable)
```

Commands to program and read the register are summarized in the table in Status function summary (on page 15-4).

This register set feeds to bit B7 (OSB) of the Status Byte. The bits used in the Operation Status Register set are described as follows:

- **Bit B0, Calibrating (CAL):** Set bit indicates that one or more channels are calibrating.
- **Bit B3, Sweeping (SWE):** Set bit indicates that one or more channels are sweeping.
- **Bit B4, Measuring (MEAS):** Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement.
- **Bit B10, Trigger Overrun (TRGOVR):** Set bit indicates that an enabled bit in the Operation Status Trigger Overrun Summary Register is set.
- **Bit B11, Remote Summary (REM):** Set bit indicates that an enabled bit in the Operation Status Remote Summary Register is set.
- **Bit B12, User (USER):** Set bit indicates that an enabled bit in the Operation Status User Register is set.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that an enabled bit in the Operation Status Instrument Summary Register is set.
- **Bit B14, Program Running (PROG):** Set bit indicates that a program is running.

For more information on the Operation Status Registers, refer to Status register set contents (on page 15-1) and the figures in this section.

Questionable Status Registers

This register set feeds to bit B3 (QSB) of the Status Byte. The bits used in the Questionable Status Register set are described as follows:

- **Bit B8, Calibration (CAL):** Set bit indicates that calibration is questionable.
- **Bit B9, Unstable Output (UO):** Set bit indicates that an unstable output condition was detected.
- **Bit B12, Over Temperature (OTEMP):** Set bit indicates that an over temperature condition was detected.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that a bit in the Questionable Status Instrument Summary Register is set.

Questionable Status Registers

As shown in the Operation event, I/O, and TSP-Link registers (on page 15-11) of the status model, there are seven register sets associated with Questionable Status. Commands are summarized in Status byte and service request (SRQ) (on page 15-15). You can also set bits by using numeric parameter values. For details, see Programming enable and transition registers.

For example, either of the following commands sets the CAL enable bit (B8):

```
status.questionable.enable = status.questionable.CAL
status.questionable.enable = 256
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Reading registers (on page 15-14). For example, the following command reads the Questionable Status Enable Register:

```
print(status.questionable.enable)
```

For more information about the Questionable Status Registers, refer to Status register set contents (on page 15-1) and the figures in this section.

Measurement Event Registers

As shown in the status model's Measurement event registers (on page 15-7), there are eight register sets associated with measurement event status. Commands are summarized in the Status register set contents (on page 15-1) topic. Note that bits can also be set by using numeric parameter values. For details, see Programming enable and transition registers .

For example, either of the following commands will set the VOLTAGE_LIMIT enable bit:

```
status.measurement.enable = status.measurement.VOLTAGE_LIMIT
status.measurement.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Reading registers (on page 15-14). For example, the following command will read the Measurement Event Enable Register:

```
print(status.measurement.enable)
```

This register set feeds to bit B0 (MSB) of the Status Byte. The bits used in the Measurement Event Registers are described as follows:

- **Bit B0, Voltage Limit (VLMT):** Set bit indicates that the voltage limit was exceeded. This bit will be updated only when either a measurement is taken or the `smuX.source.compliance` attribute is read.
- **Bit B1, Current Limit (ILMT):** Set bit indicates that the current limit was exceeded. This bit will be updated only when either a measurement is taken or the `smuX.source.compliance` attribute is read.
- **Bit B7, Reading Overflow (ROF):** Set bit indicates that an overflow reading has been detected.
- **Bit B8, Buffer Available (BAV):** Set bit indicates that there is at least one reading stored in either or both of the nonvolatile reading buffers.
- **Bit B11, Output Enable (OE):** Set bit indicates that output enable was asserted.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that a bit in the Measurement Instrument Summary Register is set.

Commands to program and read the register are summarized in the Status function summary (on page 15-4) table. For more information about the Measurement Event Registers, refer to Status register set contents (on page 15-1) and the figures in this appendix.

Register programming example

The command sequence below programs the instrument to generate a service request (SRQ) and set the system summary bit in all TSP-Link nodes when the current limit on channel A is exceeded.

```
-- Clear all registers.
status.reset()

-- Enable current limit bit in current limit register.
status.measurement.current_limit.enable = status.measurement.current_limit.SMUA

-- Enable status measure current limit bit.
status.measurement.enable = status.measurement.ILMT

-- Set system summary; enable MSB.
status.node_enable = status.MSB

-- Enable status SRQ MSB.
status.request_enable = status.MSB
```

TSP-Link system status

The TSP-Link® expansion interface allows instruments to communicate with each other. The test system can be expanded to include up to 32 TSP-enabled instruments. In a TSP-Link system, one node (instrument) is the master and the other nodes are the subordinates. The master can control the other nodes (subordinates) in the system. See TSP-Link system expansion interface for details about the TSP-Link system.

The system summary registers, shown in the Status byte and service request enable register and the System summary and standard event registers (on page 15-7), are shared by all nodes in the TSP-Link system. A status event that occurs at a subordinate node can generate an SRQ (service request) in the master node. After detecting the service request, your program can then branch to an appropriate subroutine that services the request. See Status byte and service request (SRQ) (on page 15-15) for details.

Status model configuration example

In this example, a current limit (compliance) event in SMU A of node 15 will set the RQS bit of the Status Byte of the master node. The commands to configure the status model for this example are provided in Status configuration (enable) commands (on page 15-27).

When a current limit (compliance) condition occurs in SMU A of Node 15, the following sequence of events will occur:

- Node 15: Bit B1 of the Measurement Event Current Limit Summary Register sets when the current limit (compliance) event occurs.
- Node 15: Bit B1 (ILMT) of the Measurement Event Register sets.

- Node 15: Bit B0 (MSB) of the Status Byte sets.
- System Summary Registers: Bit B1 (Node 15) of the System Summary Register 2 sets.

NOTE

The System Summary Registers are shared by all nodes in the TSP-Link system. When a bit in a system register of node 15 sets, the same bit in the master node system register also sets.

- System Summary Registers: Bit B0 (Extension) of the System Summary Register sets.
- Master Node: Bit B0 (MSB) of the Status Byte sets.
- Master node: With service request enabled, bit B6 (RQS) of the Status Byte sets. When your program performs the next serial poll of the master node, it will detect the current limit event and can branch to a routine to service the request.

The figure in Status configuration (enable) commands (on page 15-27) demonstrates the flow of information through the status model of node 15 and the master node.

Status configuration (enable) commands

The following commands (sent from the master node) enable the appropriate register bits for the above example:

Node 15 status registers: The following commands enable the current limit events for SMU A of node 15:

```
node[15].status.measurement.current_limit.enable = 2
node[15].status.measurement.enable = 2
node[15].status.node_enable = 1
```

The affected status registers for the above commands are indicated by labels A, B and C (see following figure).

Master node system summary registers: The following commands enable the required system summary bits for node 15:

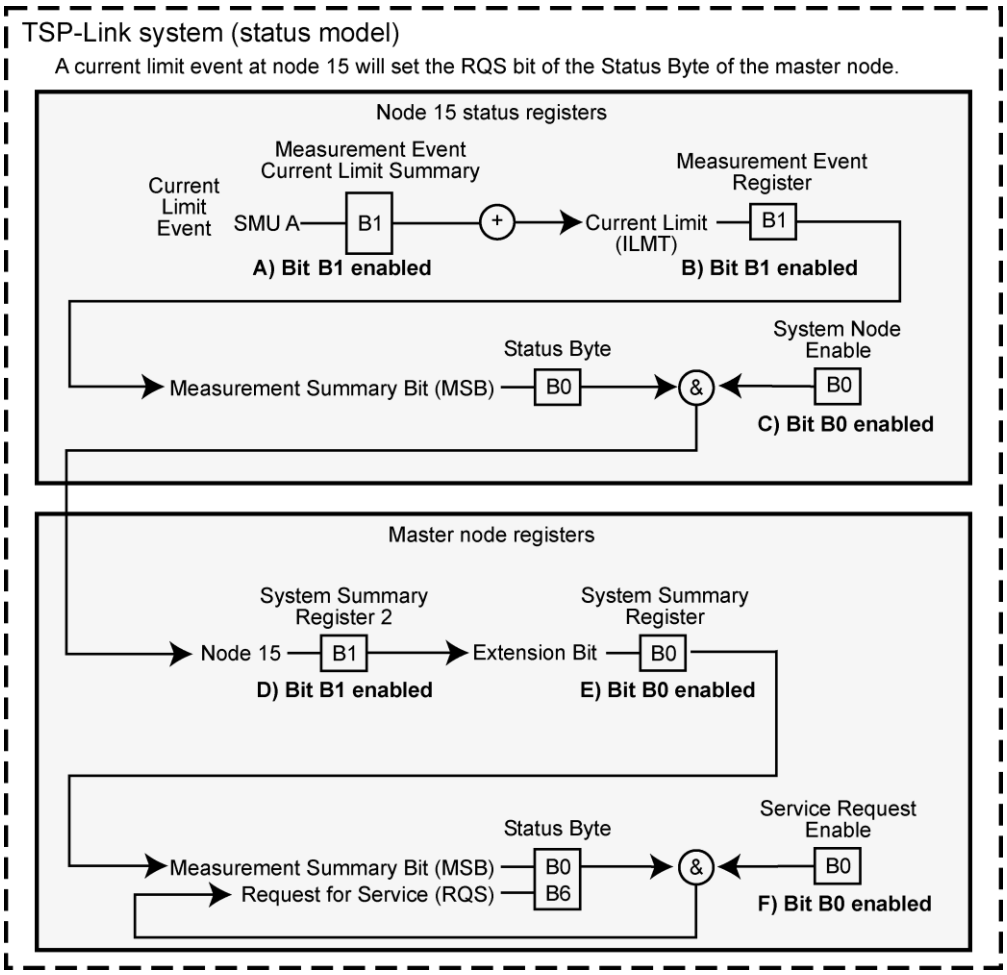
```
status.system2.enable = 2
status.system.enable = 1
```

The affected system summary registers for the above commands are indicated by labels D and E (see following figure).

Master node service request: The following command enables the service request for the measurement event:

```
status.request_enable = 1
```

The affected status register for the above command is indicated by label E, as shown in the following figure.



Section 16

Display character codes

In this section:

Model 2651A display character codes

The following tables contain the decimal values of the display character codes and the corresponding displays.

Display character codes (decimal 0 to 39)

Decimal	Display	Decimal	Display	Decimal	Display
000	reserved	012	reserved	026	▲
001	reserved	013	reserved	027	▼
002	reserved	014	reserved	028	◄
003	reserved	015	reserved	029	►
004	reserved	016	µ	030	
005	reserved	017	±	031	
006	reserved	018	Ω	032	(space)
007	reserved	019	°	033	!
008	reserved	020		034	"
009	reserved	021		035	#
010	reserved	022		036	$
011	reserved	023		037	%
012	reserved	024		038	&
013	reserved	025		039	' (apostrophe)

Display character codes (decimal 40 to 102)					
Decimal	**Display**	**Decimal**	**Display**	**Decimal**	**Display**
040	(	061	=	082	R
041	)	062	>	083	S
042	*	063	?	084	T
043	+	064	@	085	U
044	, (comma)	065	A	086	V
045	-	066	B	087	W
046	.	067	C	088	X
047	/	068	D	089	Y
048	0	069	E	090	Z
049	1	070	F	091	[
050	2	071	G	092	\
051	3	072	H	093	]
052	4	073	I	094	^
053	5	074	J	095	_
054	6	075	K	096	' (open single quote)
055	7	076	L	097	a
056	8	077	M	098	b
057	9	078	N	099	c
058	:	079	O	100	d
059	;	080	P	101	e
060	<	081	Q	102	f

Display character codes (decimal 103 to 165)					
Decimal	**Display**	**Decimal**	**Display**	**Decimal**	**Display**
103	g	124	|	145	
104	h	125	}	146	
105	i	126	~	147	
106	j	127		148	
107	k	128	(space)	149	
108	l	129		150	
109	m	130		151	
110	n	131		152	
111	o	132		153	
112	p	133		154	
113	q	134		155	
114	r	135		156	
115	s	136		157	
116	t	137		158	
117	u	138		159	¼
118	v	139		160	0
119	w	140		161	1
120	x	141		162	2
121	y	142		163	3
122	z	143		164	4
123	{	144		165	5

Display character codes (decimal 166 to 228)					
Decimal	**Display**	**Decimal**	**Display**	**Decimal**	**Display**
166	^{6}	187	Φ	208	æ
167	^{7}	188	∩	209	Æ
168	^{8}	189	∪	210	â
169	^{9}	190	÷	211	ä
170	α	191	≤	212	á
171	ß	192	≥	213	à
172	γ	193	≠	214	å
173	δ	194	≡	215	a̲
174	ε	195	≈	216	Ä
175	η	196	∞	217	Å
176	θ	197	>>	218	ê
177	λ	198	<<	219	ë
178	π	199	¿	220	é
179	ρ	200	¡	221	è
180	σ	201	¢	222	É
181	τ	202	£	223	î
182	φ	203	¥	224	ï
183	ω	204	$P_{\dagger}$	225	í
184	Γ	205	*ƒ*	226	ì
185	Δ	206	Ç	227	ô
186	Σ	207	ç	228	ö

Display character codes (decimal 229 to 255)

Decimal	Display	Decimal	Display	Decimal	Display
229	ó	238	ñ	247	
230	ò	239	Ñ	248	
231	O̲	240	ÿ	249	
232	Ö	241		250	
233	û	242		251	†
234	ü	243		252	↑
235	ú	244		253	↓
236	ù	245		254	←
237	Ü	246		255	→

Keithley Instruments
Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • 1-800-833-9200 • tek.com/keithley

KEITHLEY
A Tektronix Company

07/2020