Model 2450
Interactive SourceMeter® Instrument

Reference Manual

2450-901-01 Rev. B / September 2013

2450-901-01B

A Greater Measure of Confidence

KEITHLEY
A Tektronix Company

Model 2450

Interactive SourceMeter® Instrument

Reference Manual

Document number: 2450-901-01 Rev. B / September 2013

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

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

The types of product users are:

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

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

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

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

Keithley Instruments products are designed for use with electrical signals that are 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, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.

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

Chassis connections must only be used as shield connections for measuring circuits, NOT as 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 danger. 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 caution, risk of electric shock. Use standard safety precautions to avoid personal contact with these voltages.

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

The symbol indicates a connection terminal to the equipment frame.

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

Safety precaution revision of January 2013.

Table of Contents

Section 1

Introduction

In this section:

Welcome

Thank you for choosing a Keithley Instruments product. The Model 2450 Interactive SourceMeter® instrument is a precise, low-noise instrument that combines a stable DC power supply, true current source, electronic load, and a high-impedance multimeter. The design of this instrument features intuitive setup and control, enhanced signal quality and range, and better resistivity and resistance capabilities than similar products on the market.

With 0.012 percent basic accuracy at 6½-digit resolution, the Model 2450 delivers 59 readings per second over the IEEE-488 bus. At 4½-digit resolution, it can read up to 1359 readings per second into its internal buffer.

Extended warranty

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

Contact information

CD-ROM contents

The following CD-ROMs are shipped with each Model 2450 instrument:

- The Interactive SourceMeter® SMU Instruments Product Information CD-ROM (Keithley Instruments part number 24GDI-950-01)
- Test Script Builder Integrated Development Environment CD-ROM (Keithley Instruments part number KTS-850)
- Keithley KickStart Startup Software CD (Keithley Instruments part number KKS-850-01)

The Interactive SourceMeter SMU Instruments Product Information CD-ROM contains:

- **Quick Start Guide:** Provides unpacking instructions, describes basic connections, reviews basic operation information, and provides a quick test procedure to ensure the instrument is operational.
- **User's Manual:** Provides application examples that you can use as a starting point to create your own applications.
- **Reference Manual:** Includes advanced operation topics, maintenance information, troubleshooting procedures, optimization strategies, and in-depth descriptions of programming commands.
- **KickStart Software Quick Start Guide:** Provides instructions to quickly make measurements and get results without having to program test scripts.
- **Accessories information:** Documentation for accessories that are available for the Model 2450.

The Test Script Builder Integrated Development Environment CD-ROM contains:

The installation files for the Test Script Builder Integrated Development Environment: This software provides an environment in which you can develop and execute a test program, and it gives you the ability to load a test program onto the instrument. Running a program that is loaded on the instrument eliminates the need to send individual commands from the host computer to the instrument when running a test.

The Keithley KickStart Startup Software CD-ROM contains:

- **The KickStart software.**
- **KickStart Software Quick Start Guide:** Provides instructions to quickly make measurements and get results without having to program test scripts.

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, configuration lists, triggering, the digital I/O port, and TSP-Link synchronization lines.
- **Source-measure considerations:** Describes best practices and recommended procedures that can increase measurement speed, accuracy, and sensitivity.
- **Introduction to SCPI commands:** Describes how to control the instrument using SCPI commands.
- **SCPI command reference:** Contains programming notes and an alphabetical listing of all SCPI commands available for the Model 2450.
- **Introduction to TSP operation:** Describes the basics of using Test Script Processor (TSP®) commands to control the instrument and describes how to control the instrument using TSP commands and Test Script Builder (TSB®) software, TSP-Link system expansion, and TSP-Net.
- **TSP command reference:** Contains programming notes and an alphabetical listing of all TSP commands available for the Model 2450.
- **Frequently asked questions:** Contains information that answers commonly asked questions.
- **Next steps:** Contains sources of additional information.
- **Maintenance:** Contains information about instrument maintenance, including line fuse replacement and firmware upgrades.
- **Common commands:** Contains descriptions of IEEE Std. 488.2 common commands.
- **Status model:** Describes the Model 2450 status model.
- **Using the Model 2450 in a Model 2400 application:** Provides information on using the Model 2450 as a drop-in replacement for a Model 2400 and information on how to convert Model 2400 SCPI code to Model 2450 SCPI code.

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

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

Capabilities and features

The Model 2450 has the following features:

- High-resolution, five-inch touch-screen display with enhanced graphical data visualization and on-screen debug and error histories
- Reduced source noise in band (<10 Hz) and wideband (<1 MHz)
- Ability to perform sensitive measurements on low-level signals
- Higher V-sense levels, easier 4-wire configuration, and better four-point probing performance than earlier models
- Expanded voltage, current, and resistance measurement ranges
- Simplified trigger model with source and memory configuration lists
- Front-panel USB-A connector for flash-drive support; rear-panel USB-B connector for communication, control, and data transfer
- Five front-panel banana jacks, front-panel safety earth ground, and four rear-panel triaxial connectors
- Backward compatibility with Model 2400 SCPI programs using Model 2400 emulation mode

Some additional capabilities of the Model 2450:

- Source-measure sweep capabilities (linear and logarithmic staircase sweeps)
- Four-quadrant source and sink operation
- Limit testing with a built-in comparator for pass/fail testing
- Digital I/O for stand-alone binning operations or interface to a component handler
- SCPI and Test Script Processor (TSP™) programming languages with remote interface ports (IEEE-488/GPIB, USB, and LAN)
- Built-in math expressions and user-defined expressions (using a remote interface)
- Up to one-million-point reading buffer with seven setups (five user defaults, factory default, *RST default) that can be stored and recalled
- Resistance and power measurement functions
- High-capacitance mode for load impedance up to 50 μF (microfarads)
- Filtering to reduce reading noise
- Overcurrent and overvoltage protection
- Safety interlock through rear-panel connector
- Trigger model supports extensive triggering and synchronization schemes at hardware speeds
- LXI® Core Specification 1.4 compliance
- TSP-Link® system expansion interface that test system builders can use to connect multiple instruments in a master and subordinate configuration. TSP-Link is a high-speed trigger synchronization and communication bus.; advanced Test Script Processor (TSP®) scripting engine features enable parallel script execution across a network
- Supports IEEE-488 (GPIB), USB, and ethernet local area network (LAN) connections

General ratings

The Model 2450 instrument's general ratings and connections are listed in the following table.

Category	Specification
Supply voltage range	100 V to 240 V_{RMS}, 50 Hz or 60 Hz (autosensing at power up)
Input and output connections	See Rear panel overview (on page 2-7)
Environmental conditions	For indoor use only **Altitude:** Maximum 2000 meters (6562 feet) above sea level **Operating:** 0 °C to 50 °C (32 °F to 122 °F), 70% relative humidity up to 35 °C (95 °F); derate 3% relative humidity per °C, 35 °C to 50 °C (95 °F to 122 °F) **Storage:** –25 °C to 65 °C (–13 °F to 149 °F) **Pollution degree:** 1 or 2

Section 2

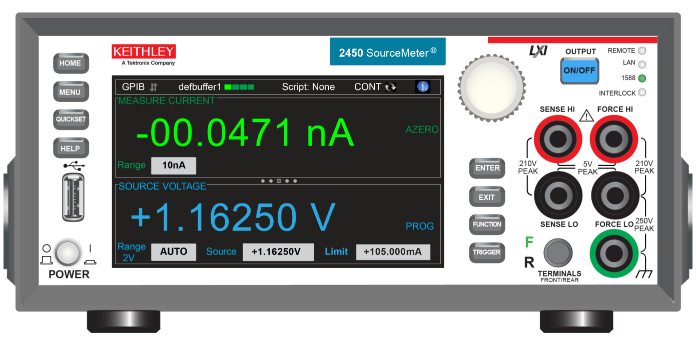
General operation

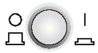
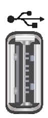
In this section:

Front-panel overview

The front panel of the Model 2450 is shown below. Descriptions of the controls on the front panel follow the figure.

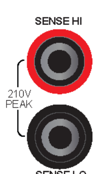
Figure 1: Model 2450 front panel



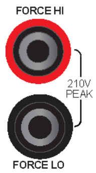
Power switch		Press the power switch so that it is in the on position (|) to turn the Model 2450 on. Press the power switch so that it is in the off position (O) to turn it off.
HOME key		Press the **HOME** key to return the display to the home (default) screen.
MENU key		Press the **MENU** key to open the main menu. Press the icons on the main menu to open source, measure, views, trigger, scripts, and system settings screens. For details, see Menu overview (on page 2-19).
QUICKSET key		Press the **QUICKSET** key to open a menu of preconfigured one-touch setups, including voltmeter, ammeter, ohmmeter, and power supply setups. Also allows you to choose test functions and quickly adjust performance for better resolution or speed. For details, see QuickSet controls menu (on page 2-20).
HELP key		Opens context-sensitive help for the area that has focus on the display. If there is no specific focus when you press the **HELP** key, overview information for the screen you are viewing displays.
USB port		Supports flash drives that comply with USB 2.0 standards and 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 from a remote interface. The flash drive must be formatted as a FAT drive.
Touch screen		The Model 2450 has a high-resolution, five-inch color touch-screen display with additional swipe screens. You can access additional interactive screens by pressing the front-panel **MENU**, **QUICKSET**, and **FUNCTION** keys. See Using the touch-screen interface (on page 2-8) for details.
Navigation control		**Turning the navigation control:** Moves the cursor to the left or the right to highlight a listed value or menu item so that you can select it.
ENTER key		Press the **ENTER** key to select the highlighted choice or to edit the field that presently has focus.

EXIT key		Press the **EXIT** key to return to the previous screen or close a dialog box. For example, press the **EXIT** key when the main menu page is displayed to return to the home screen. When you are viewing a subscreen (for example, the Event Log screen), press the **EXIT** key to return to the main menu screen.
FUNCTION key		Press the **FUNCTION** key to display instrument functions. To select one of the functions, press the function on the touch screen. Alternatively, turn the navigation control to highlight a function, and then press the control to select the function.
TRIGGER key		The TRIGGER key provides access to trigger-related settings and operations. Pressing the TRIGGER key results in different actions, depending on the instrument state. For details, see Switching between measurement methods (on page 3-92).
OUTPUT ON/OFF switch		Press the **OUTPUT ON/OFF** switch to turn on the source output; the key illuminates when the source output is on. Press the **OUTPUT ON/OFF** switch again to turn off the source output.
REMOTE LED indicator		Illuminates when the instrument is controlled through a remote interface.
LAN LED indicator		Illuminates when the instrument is connected to a local area network (LAN).
1588 LED indicator		Illuminates when the instrument is connected to an IEEE-1588 compliant device.

NOTE

INTERLOCK LED indicator		Illuminates when the interlock is enabled.
Sense terminals		Use SENSE HI and SENSE LO terminal connections to measure voltage at the device under test (DUT). Using sense leads gives you more accurate voltage sourcing and measurement at the DUT, because measurement of the voltage drop across the force leads is eliminated.

Force terminals

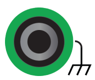


Use FORCE HI and FORCE LO terminal connections to source or sink voltage or current to or from a device under test (DUT).

FRONT/REAR TERMINALS switch

Press the **FRONT/REAR TERMINALS** switch to activate terminals on the front or rear panel. When the front-panel terminals are active, a green "F" is visible to the left of the FRONT/REAR switch. When the rear-panel terminals are active, a yellow "R" is visible to the left of the switch.

Chassis connection



Banana jack connector that provides a chassis connection.

Power the instrument on or off

Follow the procedure below to connect the Model 2450 to line power and turn on the instrument. The Model 2450 operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz or 60 Hz. Line voltage is automatically sensed. Make sure the operating voltage in your area is compatible.

The Model 2450 must be turned on and allowed to warm up for at least one hour to achieve rated accuracies.

CAUTION

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

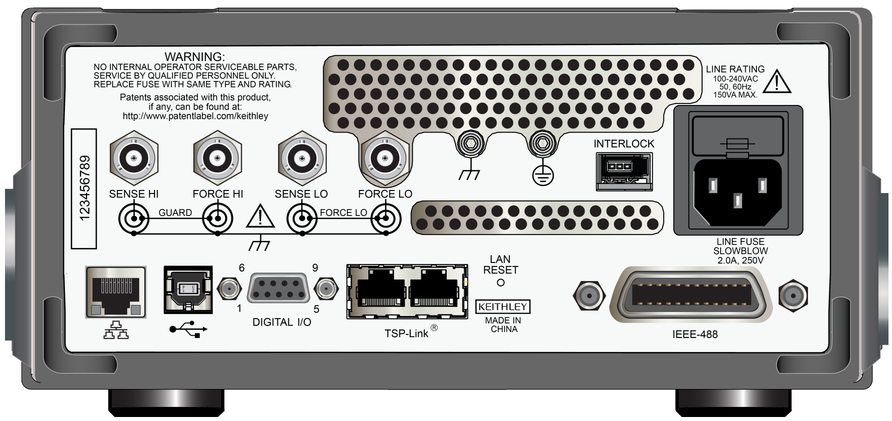
WARNING

The power cord supplied with the Model 2450 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.

To connect the power cord:

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

Figure 2: Model 2450 rear panel



To turn a Model 2450 on and off:

1. Disconnect any devices under test (DUTs) from the Model 2450 before turning the instrument on.
2. To turn your instrument on, press the front-panel **POWER** switch to place it in the on (**I**) position. A status bar is displayed as the instrument powers up. The Home screen is displayed when power up is complete.
3. To turn your instrument off, press the front-panel **POWER** switch to place it in the off (O) position.

Turning the Model 2450 output on

You can turn the Model 2450 output on from the front panel and by sending remote commands.

To turn the output on using the front panel:

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

To turn the output on using SCPI commands:

```
:OUTPut:STATe ON
```

To turn the output on using TSP commands:

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

Turning the Model 2450 output off

⚠ WARNING

Turning the Model 2450 output off does not place the instrument in a safe state (an interlock is provided for this function).

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2450 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 2450 before handling cables. Putting the equipment into an output-off state does not guarantee that the outputs are powered off if a hardware or software fault occurs.

When the source of the instrument is turned off, it may not completely isolate the instrument from the external circuit. You can use the output-off setting to place the Model 2450 in a known, noninteractive state during idle periods, such as when changing the device under test. The output-off states that can be selected for a Model 2450 are normal, high-impedance, zero, or guard. See Output-off state (on page 2-82) for additional detail.

Using the front panel:

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

Using SCPI commands:

To turn the output on, send the command:

```
:OUTPut:STATe ON
```

To turn the output off, send the command:

```
:OUTPut:STATe OFF
```

Using TSP commands:

To turn the output on, send the command:

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

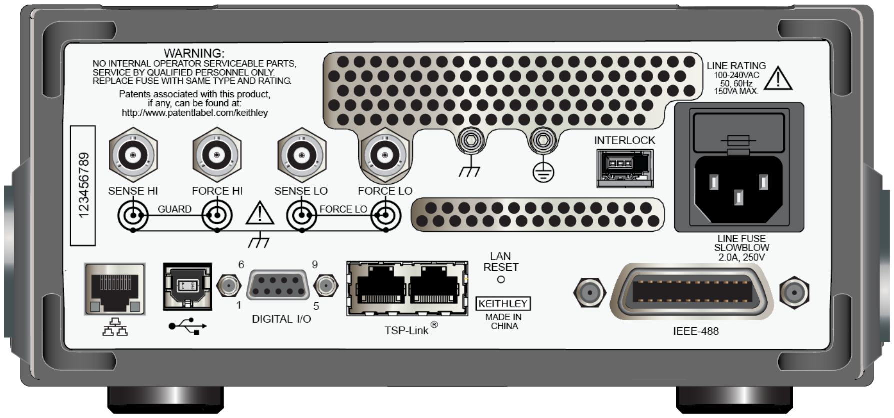
To turn the output off, send the command:

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

Rear panel overview

The rear panel of the Model 2450 is shown below; descriptions follow the figure.

Figure 3: Model 2450 rear panel



SENSE and FORCE connectors

These triaxial terminals provide connections for SENSE HI and SENSE LO, FORCE HI and FORCE LO, GUARD, and chassis ground.

Protective earth (safety ground)

Ground screw for connection to protective earth (safety ground). Connect to protective earth using recommended wire size (#16 AWG or larger).

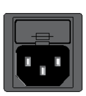
Chassis ground

Ground screw for connections to chassis ground. This provides a connection terminal to the equipment frame.

Interlock connector

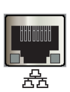
Interlock connection for use with an interlock switch, such as a test fixture. When properly connected, the safety interlock of the Model 2450 places the outputs of the instrument in a safe state. For details, see Using the interlock (on page 2-70).

Line fuse and power receptacle

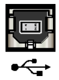
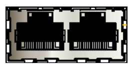
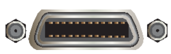


The line fuse, located just above the power receptacle, protects the power line input of the instrument. For safety precautions and other details, see Line fuse replacement (on page A-1) and Power the instrument on and off.

LAN port



Supports full connectivity on a 10 Mbps or 100 Mbps network. The Model 2450 is a LXI version 1.4 Core 2011 compliant instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet LAN).

USB port		USB-B connector for communication, control, and data transfer. For details, see USB communications (on page 2-60).
Digital I/O port		A digital input/output port that can be used to control external digital circuitry. The port provides 6 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.
TSP-Link ports		TSP-Link® system expansion interface that test system builders can use to connect multiple instruments in a master and subordinate configuration. TSP-Link is a high-speed trigger synchronization and communication bus.
LAN reset		Reverts the LAN settings and the password for the instrument to default values.
IEEE-488 port		GPIB connection; the default setting for the Model 2450 is 18.

Using the touch-screen interface

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

The following topics describe the features of the touch-screen in more detail.

⚠ CAUTION

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

Navigating the touch screen

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

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

NOTE

Some of the interactive screens have additional information on them that is not visible until you scroll down on the screen. These scrolling screens are identified by a scroll indicator on the right side of the screen. To scroll down, lightly swipe the screen (not the scroll indicator) in an upward motion. To scroll back to the top, lightly swipe in a downward motion. You can control how far the screen scrolls by using a longer swiping motion to scroll in larger increments, or by using a shorter swiping motion to scroll in smaller increments.

To access the additional swipe screens:

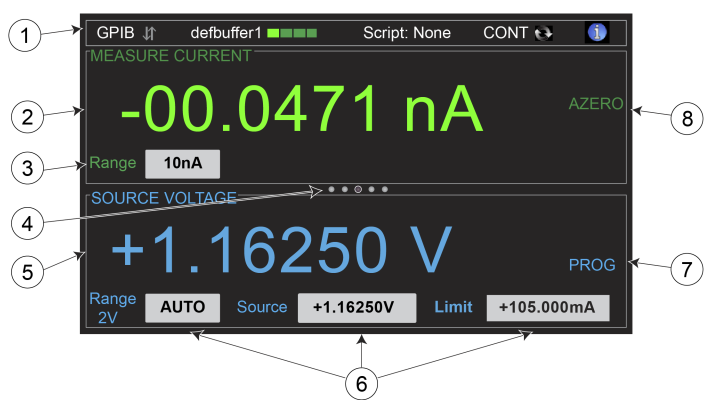
Touch the lower half of the screen and gently swipe to the left or right with your finger.

The following topics describe each of the Model 2450 screens in more detail.

Home screen

This is the default screen that you see whenever you turn the Model 2450 on or when you press the HOME key. The following figure shows the home screen with the different areas of the screen numbered. Descriptions of the screen areas are in the table following the figure.

Figure 4: Model 2450 Home screen



#	Screen element	Description
1	**System status and event indicators**	Located at the top of the Home screen, these indicators provide information about the present state of the instrument. Some of the indicators open up a dialog box with more information or a settings menu when selected. For details, see Status and error indicators (on page 2-13).
2	**MEASURE view area**	The green part of the home screen; displays the value of the present measurement.
3	**Measure range button**	Located in the lower left corner of the MEASURE view area. Shows the presently set measure range; select the button next to the indicator to change the range.
4	**Swipe screen context indicator**	Located between the top and bottom halves of the screen. Each circle represents one swipe screen. As you swipe right or left, a different circle changes color, indicating where you are in the screen sequence. The default position of the indicator when the instrument is turned on is the center circle (Home screen).
5	**SOURCE view area**	Blue part of the home screen. When the output is on and source readback is off, it displays the programmed value of the source. When the output is on and source readback is on, it displays the measured value of the source. When the output is off, displays Output Off.
6	**Source settings buttons**	Bottom of the SOURCE view part of the home screen. Shows the presently set source, source range, and source limit values. You can change these values from the front panel by selecting the buttons on this screen.
7	**Source function indicators**	Located on the right edge of the SOURCE view area, these indicators show the present source function of the instrument: • MEAS: The value shown is the actual source value (source readback is on) • PROG: The value shown is the programmed source value (source readback is off) • SRQ: A service request is asserted.
8	**Measure function indicators**	Located on the right edge of the MEASURE view area, these indicators show the present measure function of the instrument: • FILT: The filter indicator; a digital filter is enabled. • MATH: Math operations are enabled. • AZERO: The instrument can be set to automatically get new measurements of its internal ground and voltage reference between each reading • REL: Relative measurements are enabled.

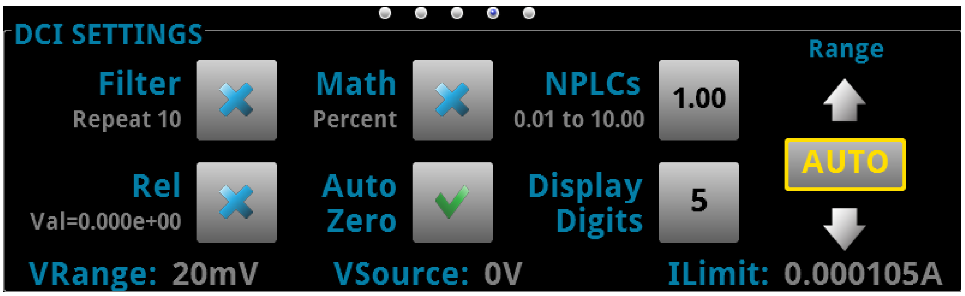
Interactive swipe screens

The Model 2450 touch-screen display has multiple screens that you can access by gently swiping left or right on the lower half of the display. The following topics describe each of these screens.

SETTINGS swipe screen

The SETTINGS screen gives you front-panel access to some instrument settings so that you can change, enable, or disable them quickly. It also shows what the present settings are.

Figure 5: SETTINGS swipe screen



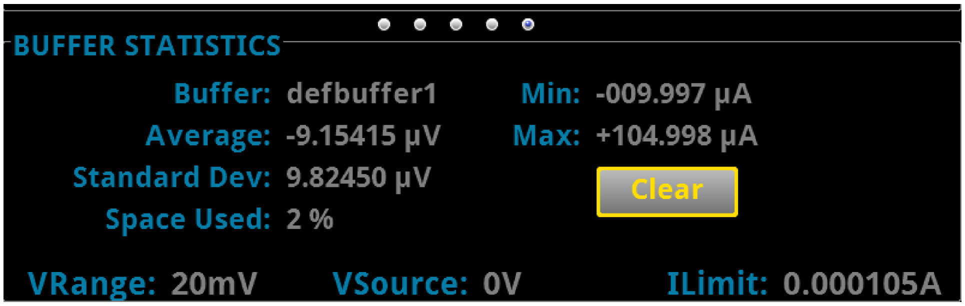
To disable or enable a setting, select the box next to the setting so that it shows an X (disable) or a check mark (enable).

Setting	Description
Filter	Shows the present type of the filter (repeating average or moving average).
Math	Shows the present math function (percent, reciprocal, or y = mx+b).
Rel	Shows the present value of the relative offset function.
Auto Zero	Sets the instrument so that it automatically returns the source to a zero value before taking another reading.
NPLC	Sets the number of power line cycles (0.01 to 10.00).
Display Digits	Sets the number of digits visible on the display (3 to 6).
Auto	Turns autoranging on or off.
Range arrows	Moves the autorange setting up or down a level. Press the up arrow to move up a range; press the down arrow to move down a range.

STATISTICS swipe screen

The STATISTICS swipe screen contains information about the state of the active buffer and the readings in it. You can also use the **Clear Stats button** on this screen to clear the data used in the statistics calculation.

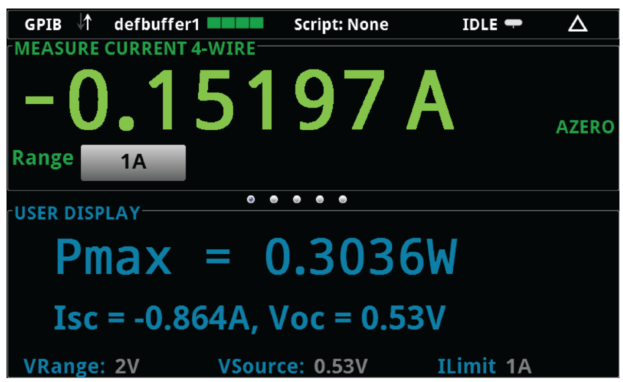
Figure 6: STATISTICS swipe screen



USER DISPLAY swipe screen

You can program custom text that will appear on the USER DISPLAY swipe screen. For example, you can program the Model 2450 to show statistics from a source-measure test. For details about using remote commands to program the display, see Customizing a message for the User Display screen (on page 2-38).

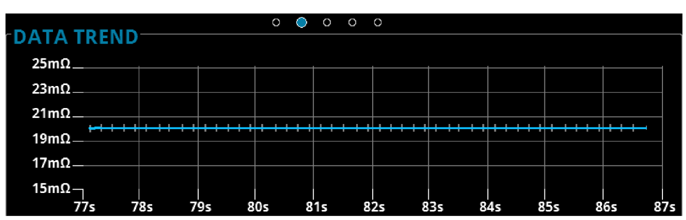
Figure 7: Example of user-defined text on the USER DISPLAY screen



DATA TREND swipe screen

The DATA TREND swipe screen shows a graphical representation of the readings in the presently selected buffer.

Figure 8: DATA TREND swipe screen



For a larger view of the graph and to access graph settings, swipe the DATA TREND screen in an upward motion. This opens up the Graph tab (also accessible by pressing the **MENU** key and selecting **Graph** under Views).

Status and error indicators

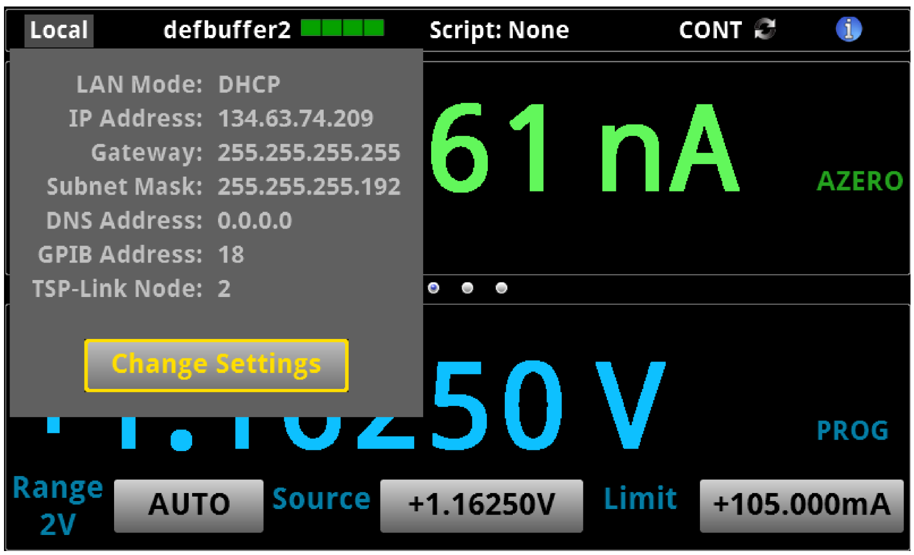
The indicators across the top of the home screen contain information about the following instrument settings and states. Some of the indicators also provide access to a subset of instrument settings that you can change.

Press an indicator (or highlight it by turning the navigation control and then pressing **ENTER**) to get more information about the present state of the instrument.

Communication settings status indicator

Press this indicator to see a list of present communications settings. Press the Change Settings button at the bottom of the information screen to change the communications settings.

Figure 9: Model 2450 communications status indicator



Indicator	Meaning
Local	Instrument is controlled from the front panel.
GPIB	Instrument is communicating through a GPIB interface.
TCPIP	Instrument is communicating through a LAN interface.
VXI-11	Instrument is communicating using VXI-11.
USBTMC	Instrument is communicating through a USB interface.
DHCP	Instrument is communicating using DHCP.
Telnet	Instrument is communicating through Telnet.
TSP-Link	Instrument is communicating through TSP-Link.
Slave	Instrument is a subordinate in a TSP-Link system.

Instrument communication activity indicator

Located directly to the right of the communications settings status indicator, these up and down arrows flash when the instrument is actively communicating with a remote interface.

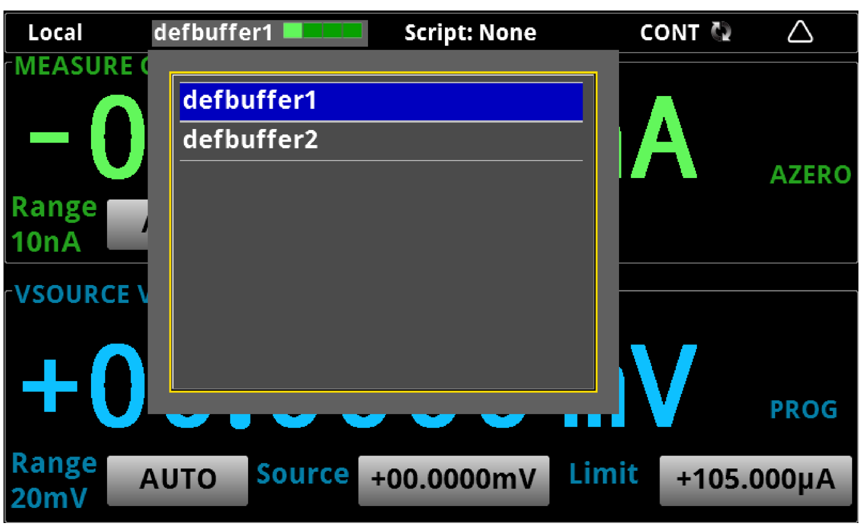
Figure 10: Instrument communication activity indicator

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

Active buffer indicator

Located to the right of the instrument active state indicator arrows, this indicator shows the name of the active reading buffer. Select the indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar. The green bar next to the buffer name indicates how full the buffer is.

Figure 11: Model 2450 active buffer indicator expanded



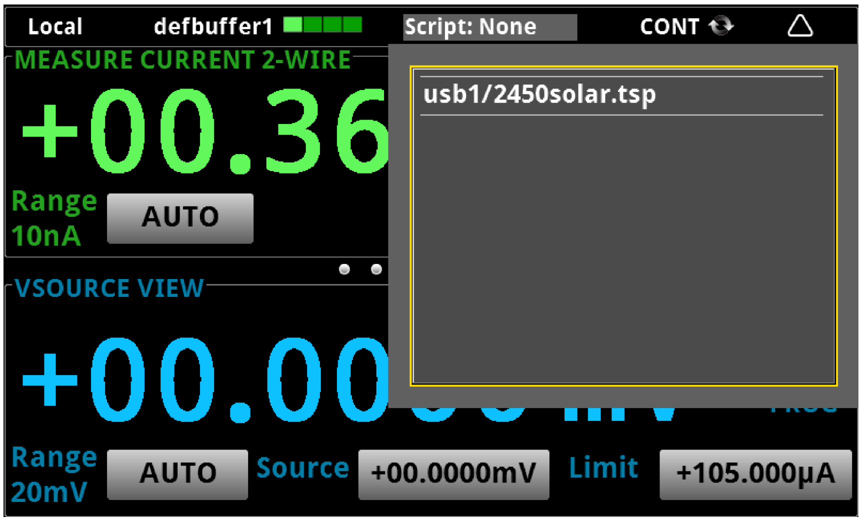
Active script indicator

Select this indicator to display a menu of available scripts. Select a script name to run the script. If a script is running, this indicator shows the name of the active script.

NOTE

If you select a script that is not compatible with the source-measure function that is active when you make select the script, you will get an event message. Because many parameters you set are related to a specific function, make sure you select the source and measure functions before you make changes to other instrument settings. The settings options you have depend upon the active source and measure functions that are selected when you make the changes. If you make a change that is not compatible with the active source and measure functions, you may get unexpected results or you may receive an event message.

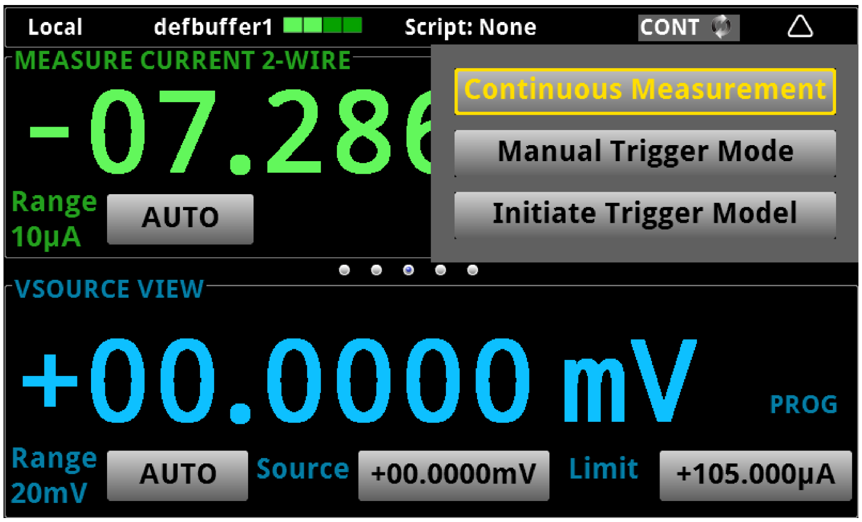
Figure 12: Model 2450 active script indicator



Trigger mode indicator

Trigger mode indicator: Located to the right of the active script indicator, this indicator shows the active trigger measurement method. Press the indicator to open a menu of available trigger measurement methods. Press one of the buttons on the menu to change the trigger measurement method. In the figure below, Continuous Measurement is the present trigger operating mode.

Figure 13: Model 2450 trigger operating mode indicator



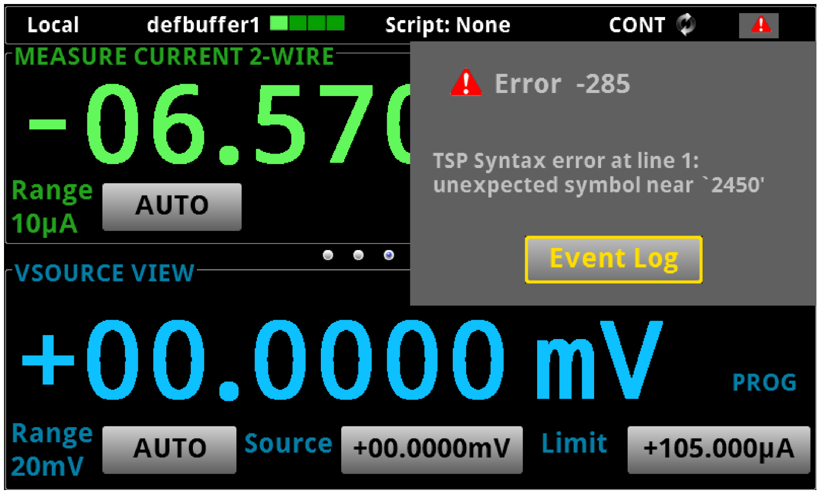
Indicator	Meaning
CONT	Continuous measurement. The instrument is taking measurements continuously.
MAN	Manual trigger mode. Press the front-panel TRIGGER key to initiate a single measurement.
RUN	Trigger model measurement method. The instrument is running the presently selected trigger model.
IDLE	Trigger model measurement method. The trigger model is not running.
WAIT	Trigger model measurement method. The trigger model is waiting on an event.

System event indicators

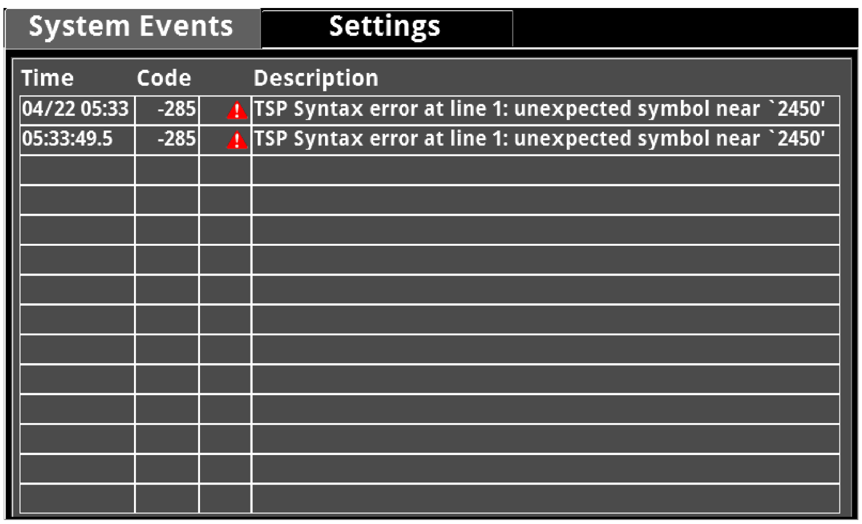
Located on the right end of the instrument status indicator bar, this indicator has a different appearance based on the type of event that has been logged. A red or yellow triangle means instrument has encountered an error or logged a warning message, and a blue circle means the instrument has logged an informational message. A white-outlined triangle means no new error or event messages have been logged since you last viewed the error log.

Press the indicator to open a message screen with a brief description of the error, warning, or event. Press the Event Log button to see the System Events entries, which contain more detailed descriptions of the errors. For more information about the Event Log, see Using the event log (on page 2-117).

Figure 14: Model 2450 error and message indicator



The figure below shows the event log entries for the error message displayed in the figure above.



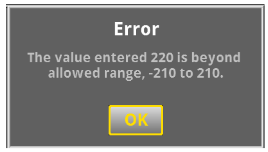
The event log indicator looks different depending on the type of event that has occurred. The following table describes the different icons and what they mean.

Icon	Description
	A white-outlined triangle means that the system event log has not logged any new events since the last time the event log was viewed.
	A blue circle means that an informational event message has been logged. The message is for information only. This is used to indicate status changes or information that may be helpful to the user. It also includes commands if the Log Command option is on.
	A yellow triangle means that a warning event message has been logged. This message indicates that a change occurred that could affect operation.
	A red triangle means that an error event message has been logged. This may indicate that a command was sent incorrectly.

Displayed error and status messages

During operation and programming, front-panel messages may be briefly displayed. Messages are either information, warning, or error notifications.

Figure 15: Example front-panel error message



Adjusting the backlight brightness and timer

You can adjust the brightness of the Model 2450 touch screen display and buttons from the front panel or over a remote interface. You can also set the backlight to dim after a specified period of time has passed with no front-panel activity (available from front panel only).

NOTE

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

To adjust the backlight brightness from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Backlight Brightness. The Display Brightness dialog box opens.
4. Touch the sliding adjustment scale to the left of the present setting to make the backlight dimmer. Touch the scale to the right of the present setting to make the backlight brighter.
5. Select **OK** to save your settings.

To set the backlight timer from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Backlight Dimmer. The Backlight Dimmer dialog box opens.
4. Select a timeout setting.
5. Select **OK**.

To adjust the brightness using the SCPI remote interface:

Send the following command:

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

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

- 100%: `ON100`
- 75%: `ON75`
- 50%: `ON50`
- 25%: `ON25`
- Display off: `OFF`
- Display and all indicators off: `BLACkout`

To adjust the backlight using TSP commands:

Send the following command:

```
display.lightstate = brightness
```

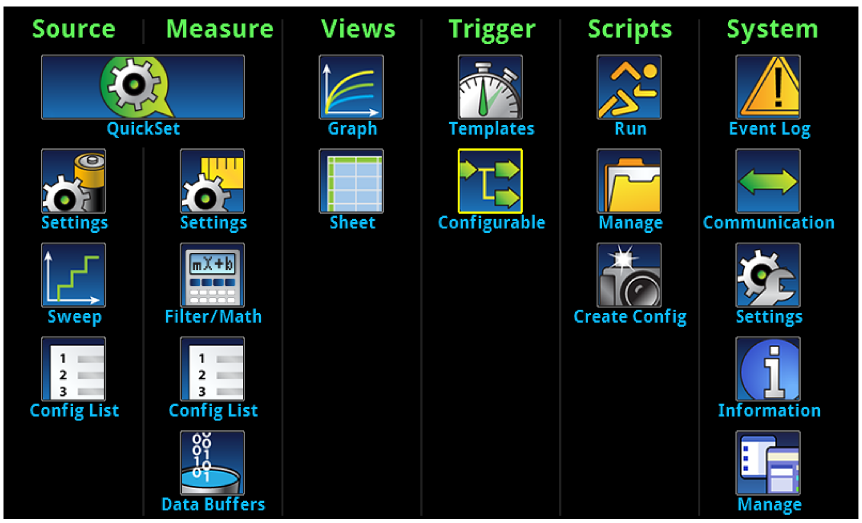
Where *`brightness`* is one of the following options:

- 100%: `display.STATE_LCD_100`
- 75%: `display.STATE_LCD_75`
- 50%: `display.STATE_LCD_50`
- 25%: `display.STATE_LCD_25`
- Display off: `display.STATE_LCD_OFF`
- Display and all indicators off: `display.STATE_BLACKOUT`

Menu overview

To access the main menu, press the **MENU** key on the Model 2450 front panel. The organization of the main menu is shown in the figure below.

Figure 16: Model 2450 main menu



The main menu is organized into six submenus, which are labeled in green across the top of the display. The icons in each column open interactive screens.

QuickSet controls menu

The QuickSet controls menu (centered under Source and Measure on the main menu) allows you to select predefined setups for Function and "One-Touch" instrument-type emulation. This menu also contains a sliding scale adjustment for Performance (resolution versus speed).

CAUTION

When you select a Quick Setup, the instrument turns the output on. Carefully consider and configure the appropriate output-off state, source, and limits before connecting the Model 2450 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

The table below shows the default settings for each of the quick setups.

QuickSet controls default settings

QuickSet selection	Source settings	Measure settings
Source Voltage and Measure Current	Voltage 0 mV, 20 mV range, current limit 105 µA	Current, 100 nA range, 2-wire, autozero on
Source Voltage and Measure Voltage	Voltage 0 mV, 20 mV range, current limit 105 µA	Voltage, 20 mV range, 2-wire, autozero on
Source Voltage and Measure Resistance	Voltage 0 mV, 20 mV range, current limit 105 µA	Resistance, 100 nA range, 2-wire, autozero on
Source Voltage and Measure Power	Voltage 0 mV, 20 mV range, current limit 105 µA	Power, 100 nA range, 2-wire, autozero on
Source Current and Measure Voltage	Current 0 nA, 10 nA range, voltage limit 21 V	Voltage, 20 mV range, 2-wire, autozero on
Source Current and Measure Current	Current 0 nA, 10 nA range, voltage limit 21 V	Current, 10 nA range, 2-wire, autozero on
Source Current and Measure Resistance	Current 0 nA, 100 mA range, voltage limit 21 V	Resistance, 2 Ω range, 2 wire, autozero on
Source Current and Measure Power	Current 0 nA, 100 mA range, voltage limit 21 V	Power, 20 mV range, 2 wire, autozero on
Voltmeter	Current, 0 A, 10 µA range, voltage limit 200 V, output on, continuous triggering mode	Voltage, autorange on, NPLC = 1
Ammeter	Voltage, 0 V, 20 V range, 1 A current limit, output on, continuous triggering mode	Current, autorange on
Ohmmeter	Current, output on, 0.1 µA, autorange, continuous triggering mode	Resistance, auto-ohms, autorange on, 2-wire or 4-wire (user-specified)
Power Supply	Voltage, fixed source level (user-specified), autorange on, source readback on, current limit (user-specified), source delay 0.1 s, output on, continuous triggering mode	Current, autorange on, 2-wire or 4-wire (user-specified)

Source menu

The menus organized under Source in the main menu allow you to select, configure, and perform source and sweep operations from the Model 2450 front panel. The following topics describe the settings that are available on these interactive screens.

Source Settings menu

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

Setting	Description
Source Range	Set the source range for the selected source function. For more information, see Source range.
Output Off State	Select from Normal, Hi Impedance, Zero, and Guard output-off states. For more information, see Output-off state (on page 2-82).
High Capacitance	Turn on this setting to minimize overshoot, ringing, and instability when measuring low current while driving a capacitive load. For more information, see High capacitance operation (on page 4-21).
Source Readback	Turn on this setting to have the instrument record and display the actual value of the source, instead of the programmed value. Using source readback results in more accurate measurements, at the cost of a reduction in measurement speed. For more information, see Source readback (on page 2-110).
Overvoltage Protection	Set the overvoltage protection setting of the source output to restrict the maximum voltage level that the instrument can source. For more information, see Overvoltage protection (on page 2-98).
Source Delay	Set a delay for the selected source function. This delay is in addition to normal settling times. For more information, see Setting the source delay (on page 2-112).

Source Sweep menu

This menu allows you to set up a sweep and generate a source configuration list, simultaneously building the trigger model.

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

For more information about setting up sweeps, see Sweep operation (on page 3-51).

Source Config List menu

This menu allows you to select an existing source configuration list, create a new list, load configuration settings to and from the instrument (system), delete a configuration point, and view the settings of a point in a source configuration list. For more information about using configuration lists, see Configuration lists (on page 3-33).

Setting	Description
Select List	Select the button next to Config List to select the configuration list that you want to use.
New List	Create a new, empty configuration list. To populate the list with the present instrument settings, select **System to List**.
Delete Point	Delete a configuration list point from the selected configuration list.
System to List	Save the present instrument configuration to a point in the selected configuration list.
List to System	Restore the instrument to the settings stored in the selected configuration list point.
View Details	View details of a specific point in the selected configuration list. Details include settings such as function, value, delay, limit, range, autorange, and output state.

Measure menu

The menus organized under Measure in the main menu allow you to select, configure, and perform measure operations from the Model 2450 front panel. The following topics describe the settings that are available on these interactive screens.

Measure Settings menu

This menu contains settings for the presently selected measurement function, which is identified by an indicator in the upper right corner of the menu.

The options in this menu include the following settings.

Measure Range: Sets the measurement range to automatic or a specific range.

Sense Mode: Select 2-wire (local) or 4-wire (remote) sense mode.

Auto Range Low Limit: Set a lower limit to prevent the instrument from selecting a range that is too low for your application.

Limits: Set limits for pass/fail device testing.

Auto Zero: Determine if the instrument automatically gets new measurements of its internal ground and voltage reference.

NPLCs: Sets the amount of time that the input signal is measured. The figure below shows the indicator when the source current and measure voltage function is selected.

Figure 17: Model 2450 function indicator

Setting	Description
Measure range	Set the measurement range for the presently selected measurement function. You can only select a measurement range if you are sourcing one type of measurement and measuring another.
Sense Mode	Select 2-wire (local) or 4-wire (remote) sense mode. Use 2-wire sense if the error contributed by test lead IR drop is acceptable. For more accurate voltage source and measurement accuracy, use 4-wire sense mode.
Auto Range Low Limit	Set the Auto Range Low Limit to prevent the instrument from selecting a range that is lower than appropriate for your application.
Limits	Select the button next to Limits to open a menu of limit settings. You can set two separate sets of limit settings (Limit 1 and Limit 2). For each set of limits, you can turn the limit state off or on, turn the Auto Clear function off or on, and set the low and high limit values.
Auto Zero	Set Auto Zero to On to set the instrument so that it periodically gets new measurements of its internal ground and voltage reference. This setting increases measurement accuracy, but may slow measurement time.
NPLCs	Set the amount of time that the input signal is measured. Lower NPLC settings result in faster reading rates, but increased noise. Higher NPLC settings result in lower reading noise, but slower reading rates.
Offset Comp	The offset compensation setting is only available when the instrument is set to measure resistance (SVMR(Ω) or SIMV(Ω)). Turn this setting on to improve low-resistance measurement accuracy.

Measure Filter/Math menu

This menu contains settings that specify the way measurement information is returned.

Setting	Description
Filter State	Turn this setting on to enable filtering of measurements.
Filter Count	The setting sets the number of measurements that are averaged when filtering is enabled. Select a value from 1 to 100.
Filter Type	Select the type of averaging filter that is used for the selected measurement function when the measurement filter is enabled. Select moving average filtering to continuously add measurements to the stack on a first-in, first out basis, replacing the oldest measurement in the stack with a new measurement. Select repeating average filter to average a set of measurements and then flush the data out of the stack before averaging a new set of measurements.
Rel State	Use the relative offset feature to subtract a set value or a baseline reading from measurement readings. When you enable relative offset, all subsequent measurements are displayed as the difference between the actual measured value and the relative offset value.
Math State	When the Math State is set to on, any math operations specified for the present measurement function are performed before completing the measurement.
Math Function	When the Math State is set to on, you can specify which math operation is performed on measurements. You can choose one of the following math operations: • **mx+b:** Manipulate normal display readings by adjusting the m and b factors. • **Percent:** Specify a constant that is applied to the measurement and display measurements as percentages. • **Reciprocal:** The reciprocal math operation displays measurement values as reciprocals. The displayed value is 1/X, where X is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).
Zero Percent Reference	When the Math State is set to on and the Math Function is set to Percent, this setting specifies the constant when the math operation is set to percent; the range is −1e12 to +1e12.

Measure Config List menu

This menu allows you to select an existing measure configuration list, create a new list, load configuration settings to and from the instrument (system), and view the settings of a point in a configuration list. For more information about using configuration lists, see Configuration lists (on page 3-33).

Setting	Description
Select List	Select the button next to Config List to select the configuration list that you want to use.
New List	Create a new, empty configuration list. To populate the list with the present instrument settings, select **System to List**.
Delete Point	Delete a configuration list point from the selected configuration list.
System to List	Save the present instrument configuration to a point in the selected configuration list.
List to System	Restore the instrument to the settings stored in the selected configuration list point.
View Details	View details of a specific point in the selected configuration list. Details include settings such as function, value, delay, limit, range, autorange, and output state.

Measure Data Buffers menu

Selecting this main menu icon opens the MANAGE MEASURE BUFFERS screen. From this screen you can see the list of existing buffers and select one to be the active buffer. You can also create, save, delete, resize, and clear buffers from this screen.

Setting	Description
Refresh	Update the screen. This does not affect the reading buffers; it only affects the display of data on this screen.
New	Select **New** to create a new reading buffer. The new buffer is automatically set to be the active buffer.

To adjust settings for a specific buffer, select the buffer to display the Settings screen for that buffer.

Setting	Description
Reading Size	Set the maximum number of readings the buffer can store (1 to 1,000,000). Note that when you resize a buffer, the readings contained in that buffer are cleared.
Fill Mode	Select **Continuous** to have the buffer fill continuously, overwriting old data when the buffer is filled. Select **Once** to have the buffer stop collecting data when it is filled (no data is overwritten).
Delete Buffer	Select this option to delete this buffer.
Clear Buffer	Select this option to clear data from this buffer.
Make Active	Set the buffer to be the active reading buffer.
Save To USB	Save the buffer to a `.csv` file, which can be opened by a spreadsheet program.

Views menu

The menus organized under View in the main menu allow you to select, configure, and view data from source-measure operations on the Model 2450. The following topics describe the settings that are available on these interactive screens.

Views Graph menu

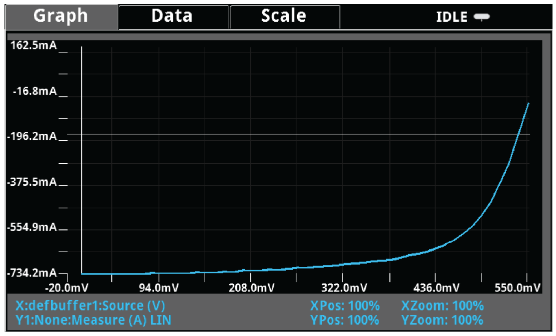
Selecting the Graph menu opens up a screen that contains a set of tabs that allow you set up and see real-time measurements in a graphical format. There are three tabs: Graph, Data, and Scale.

You can also set the trigger mode and initiate triggering on this menu by selecting the icon in the upper right corner of the tab and selecting the trigger mode.

Graph tab

The Graph tab shows readings in a graphical representation as they are being performed by the instrument. Settings you make on the Data and Scale tabs affect how readings appear on this screen.

Figure 18: Graph tab



You can zoom in or out in the graph view by placing two fingers on the screen and moving them together or apart in a pinching motion. You can also move the view of the graph to the left or right by placing a finger on the screen and moving it in either direction.

The X and Y axes can be set to show different values appropriate for your application. The bottom of the Graph tab contains a legend of the active axis and scale settings for the graph.

Data tab

The data tab contains settings that define what data will be shown in the graph.

Data tab setting	Description
Buffer	Select the button next to Buffer to select the buffer that contains the readings to be graphed.
Clear	Select Clear to clear the selected buffer.
X-Axis	Select the button next to X-Axis to select the data that is plotted on the X-Axis. You can select Source or Time.
Y-Axis	Select the button next to Y-Axis to select the data that will be plotted on the Y-Axis. You can select either Measure or Source. Note that if you have Source selected for the X-Axis and you try to set the Y-Axis to Source, you will receive an error message.
Marker	When Time is selected for the X-Axis, you can select the statistic to be graphed: Min/Max, Average, or Both.

Scale tab

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

Scale tab setting	Description
X-Axis and Y-Axis Scale	Set the reading value scale for each division for the function selected. The choices you see for this setting are directly related to what you have chosen to plot on the X or Y-axis (on the Data tab). For example, if you have chosen to plot Time on the X-axis, your selections on for the X-axis on the Scale tab will be seconds. For example, if you are sourcing current, you can specify that the value of the divisions on one of the axes on the graph are set to 1 A intervals.
X-Axis and Y-Axis Min Position	You can set the first visible value on the graph; the default setting is 0. The choices you see for this setting are directly related to what you have chosen to plot on the X or Y-axis (on the Data tab). For example, if you have chosen to plot Time on the X-axis, your selections on for the X-axis on the Scale tab will be seconds.
X-Axis Auto Scale	When the Auto Scale feature is on, the graph is scaled automatically to fit all the data that is in the selected reading buffer. If Auto Scale is off, the graph does not resize. You can scroll right or left to view the data. Note that if you pinch or swipe the display on the graph tab while readings are being made, Auto Scale is turned off.
Y-Axis Auto Scale	You can turn the Auto Scale function on or off for the Y-axis.
Y-Axis Scale Format	Select the Linear scale type to increase the step size on the graph in even increments. Select the Logarithmic scale type to increase the step size exponentially.

Views Sheet menu

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

Setting	Description
Buffer	Select the button next to Buffer to select the buffer you want to view.
Up and down arrows	Select the up or down arrow to jump up or down one page in the data displayed on the screen.
Jump	Select Jump to move to a specific row in the data sheet. This is useful to avoid scrolling through large lists of values.
Refresh	Select Refresh to update the list of values shown in the data sheet.
Data sheet window	Shows the list of readings in the selected buffer. Select a line in the sheet to see the details of that specific data point. You can scroll up or down in the data sheet window by swiping in an up or down motion on the sheet (or using the up and down arrows or Jump button).
Data Point Details	When you select a line in the sheet, a screen opens that shows a detailed list of settings that describe the instrument state when that specific data point was read.

Trigger menu

The menus organized under Trigger in the main menu allow you to configure triggering operations from the Model 2450 front panel. The following topics describe the settings that are available on these interactive screens.

Templates menu

The Model 2450 has preprogrammed trigger model templates that you can use from the instrument front panel. When you select a template, any user-specified settings for that template are visible in the lower part of the screen.

You can also customize the templates from the front panel using the Configurable menu under Trigger on the main menu screen. For details, see Configurable menu (on page 2-32).

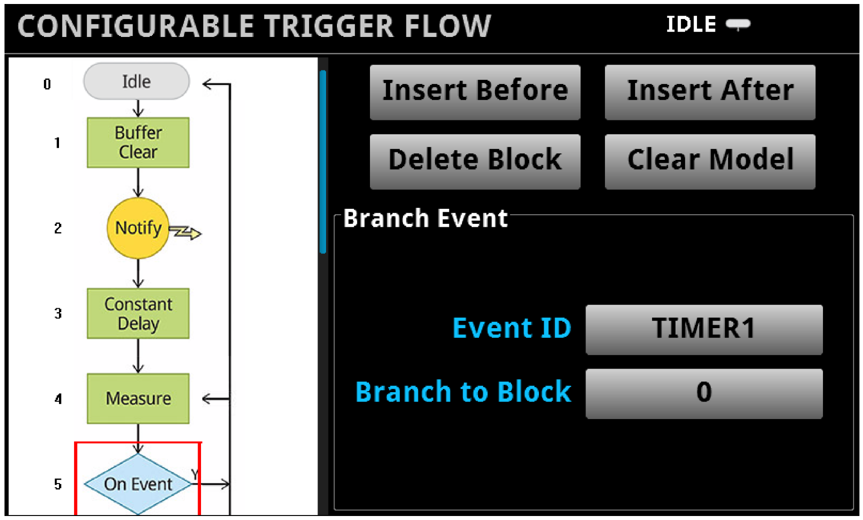
The table below describes the trigger model templates and available user-specified settings, and their default settings.

Template	Description
Empty	Selecting this template clears the present trigger model.
Config List	Creates a trigger model that can recall settings that are stored in a configuration list.
External Trigger	Creates a trigger model that allows you to use the digital I/O to send signals to trigger external instruments. Settings that you can change before generating the trigger model: • Input line (default `1`) • Count (default `10`) • Delay (default `10`) • Output Line (default `2`) • Buffer (default `defbuffer1`)
Simple Loop	Creates a trigger model that sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you defined in the count parameter. Settings that you can change before generating the trigger model: • Count (default `50`) • Delay (default `0.001s`) • Buffer (default `defbuffer1`)
Duration loop	Creates a trigger model that sets up a loop that sets the amount of time for which to take measurements (0 to 100,000 seconds), sets a constant delay, makes measurements for the specified amount of time (duration) and for the specified count, and saves the readings to the specified buffer. Settings that you can change before generating the trigger model: • Duration (default 1.00s) • Delay (default 0.001s) Buffer (default `defbuffer1`)Creates a trigger model that sets up a loop that sets the amount of time for which to take measurements (0 to 100,000 seconds), sets a constant delay, makes measurements for the specified amount of time (duration) and for the specified count, and saves the readings to the specified buffer. Settings that you can change before generating the trigger model: • Duration (default 1.00s) • Delay (default 0.001s) • Buffer (default `defbuffer1`)

Configurable menu

After you have selected one of the trigger model templates, you can see and modify the structure and parameters of the trigger model in the Configurable TriggerFlow menu.

Figure 19: Model 2450 configurable trigger menu



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

You can insert and delete trigger blocks, and you can clear the trigger model by selecting Clear Model.

Once you are finished modifying the trigger model, you can initiate it by pressing the front-panel TRIGGER button.

Scripts menu

The menus organized under Scripts in the main menu allow you to configure, run, and manage scripting operations from the Model 2450 front panel. The following topics describe the settings that are available on these interactive screens.

Run scripts menu

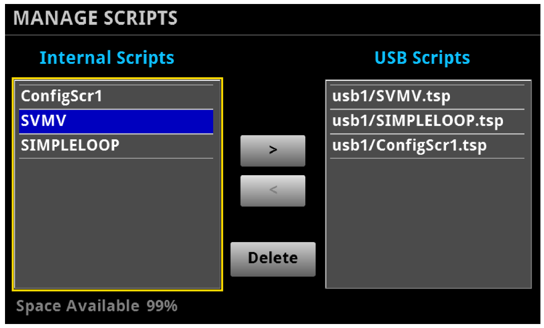
This menu contains a list of available scripts that you can select to run immediately or copy to a script that runs each time the instrument power is turned on.

Setting	Description
Available Scripts	Select a script from the list of available scripts. If there are any scripts saved on the Model 2450 or on a USB flash drive, the names of those scripts are listed.
Run Selected	Immediately runs the script that is selected in the Available Scripts list.
Copy to Power Up	Saves the selected script to the autoexec script that runs automatically when the instrument is turned on.

Manage scripts menu

You can use this menu to save internal scripts to a USB flash drive, or copy scripts on the USB flash drive to the instrument. You can also delete scripts from this menu. For more information about using scripts with the Model 2450, see Fundamentals of scripting for TSP (on page 7-4).

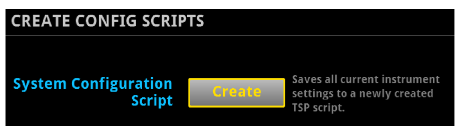
Figure 20: Model 2450 MANAGE SCRIPTS menu



Create Config menu

Selecting Create on this menu saves the present settings and any source or measure configuration lists that you have defined for the Model 2450 as a user configuration script that you can later recall or use on another instrument. For more information about user configuration scripts and setups, see Saving setups (on page 2-112).

Figure 21: Model 2450 CREATE CONFIG SCRIPTS menu



System menu

The menus organized under System in the main menu allow you to configure general instrument settings from the Model 2450 front panel. Among these settings are event log, communication, beeper and key clicks, backlight brightness and timer, time and date, system access level, password, and reading format settings.

The System Information menu is a read-only menu that shows the serial number, firmware version, line frequency, and calibration information.

The following topics describe the settings that are available on these interactive screens.

System Event Log menu

The System Event Log menu opens a screen that contains two tabs: The System Events tab and the Settings tab.

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

- Error: An error occurred. This may indicate that a command was sent incorrectly.
- Warning: This message indicates that a change occurred that could affect operation.
- Information: The message is for information only. This is used to indicate status changes or information that may be helpful to the user. It also includes commands if the Log Command option is on.

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

Settings tab settings	Description
Show Warning	Turns the display of warnings on or off. If you turn this off, the instrument continues to record warning and display popup messages, but does not display them on the System Events tab.
Show Information	Turns the display of information messages on or off. If you turn this off, the instrument continues to record information messages and display popup messages, but does not display them on the System Events tab.
Log Warning	Turns the logging of warnings on or off. If this is turned off, the instrument will not log or display popup messages for warnings.
Log Information	Turns the logging of information messages on or off. If this is turned off, the instrument will not log or display popup messages for information messages.
Log Command	Turns the logging of commands on or off. When this is turned on, the instrument records the commands that are sent to the instrument. It records commands sent from any interface (the front panel or a remote interface).
Popups	Allows you to select the type of popup message visible on the front panel, and allows you to turn off popup messages. Messages are still saved in the event log when popups are turned off.
Reset Popups	Sets event message popups to show for all message events (default settings).
Save to USB	Saves the event log to a .csv file on the USB flash drive. The filename is `eventlog.csv`.
Clear Log	Clears all entries from the event log.

System Communications menu

Selecting this menu opens a set of tabs with information about Model 2450 communications settings. Most of the tabs contain settings that you can change (there are no user settings on the USB tab).

GPIB tab settings	Description
Address	The GPIB address value is set to 18 at the factory. You can set the address to any address between 0 and 30 if it is unique in the system. This address cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

LAN tab settings*	Description
TCP/IP Mode	Select Manual to manually set the Local IP, gateway, and subnet mask values. Select Auto to set the instrument to automatically obtain an IP address.
Local IP	When TCP/IP Mode is set to Manual, you can see the present local IP address. To change the address, select the button next to Local IP and enter a new address.
Gateway	When TCP/IP Mode is set to Manual, you can see the present gateway address. To change the address, select the button next to Gateway and enter a new address.
Subnet	When TCP/IP Mode is set to Manual, you can see the present subnet mask address. To change the address, select the button next to Subnet and enter a new address.
Apply Settings	To save any changes you made on the LAN tab, select **Apply Settings**.
MAC Address	Read-only text that shows the present MAC address of the instrument.
* You must select **Apply Settings** after changing any of the settings on this tab to save your settings.	

TSP-Link tab settings	Description
Node	Select the button next to Node to set the TSP-Link node number for the instrument (1 to 64). Each instrument or enclosure attached to the TSP-Link expansion interface must be identified. This identification is called a TSP-Link node number, and the instruments or enclosures are called nodes. Each node must be assigned a unique node number.
Initialize	Select Initialize to have the Model 2450 find all TSP-Link connect instruments and form a network.

System Settings menu

This menu contains general instrument settings.

Setting	Description
Audible Errors	You can turn the beeps that occur when an error occurs on or off. The audible error setting is not affected by instrument reset or power cycle. For more information, see Instrument sounds (on page 2-67).
Key Click	You can turn the sound that occurs when you press a front-panel key on or off. The key-click setting is not affected by instrument reset or power cycle.
Backlight Dimmer	You can set the front-panel display to dim after a period of time, or you can set it so that it will never dim.
Backlight Brightness	You can adjust the brightness of the front-panel display. Selecting this setting opens a sliding adjustment scale that adjusts the brightness as a percent of total brightness.
Time and Date	Set the instrument month, day, year, and time from this menu.
Command Set	Select the type of commands to use when controlling the instrument from a remote interface (SCPI, TSP, and SCPI2400).
Password	The Model 2450 comes programmed with a default user name and password (case-sensitive), which you can change: • User name: `admin` • Password: `admin` See Instrument access (on page 3-4) for more information about controlling access to the instrument.
Reading Format	Set the format of the front-panel readings to Prefix (leading zeros) or Exponent.
Access Mode	You can specify that the control interfaces request access before taking control of the instrument. There are several levels of access: Full, Exclusive, Protected, and Lockout. For details, see Instrument access (on page 3-4).

System Information menu

The System Information menu is a read-only menu that shows the serial number, firmware version, line frequency, and calibration information.

System Manage menu

This menu gives you access to settings for instrument firmware and reset functions. For information about the Product Demo Button, contact your sales representative.

Settings	Description
Upgrade to New	Selecting this option initiates a firmware upgrade from a file on a USB flash drive. During the upgrade process, the instrument verifies that the version you are loading is newer than what is on the instrument. If the version is older or at the same revision level, no changes are made.
Downgrade to Older	Selecting this option returns the Model 2450 to a previous version of the firmware from a file on a USB flash drive. When you return to a previous version, the instrument verifies that the version you are loading is earlier than what is on the instrument.
Product Demo	Selecting this option starts a brief demonstration of the graphing capability of the Model 2450. To get correct results, you must have the appropriate demonstration fixture connected to the inputs. For more information, contact your sales representative.
System Reset	Selecting this option resets many of the instrument commands to their default values. For more information about what commands get reset, see Reset default values (on page 4-24).
LAN and Password Reset	Selecting this option reverts the LAN settings and the access password to their default values.

Display features

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

Setting the number of displayed digits

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

The number of displayed digits does not affect accuracy or speed. It also does not affect the format of readings that are returned from a remote command. To change the format of remote interface readings for Test Script Processor (TSP) commands, see format.asciiprecision (on page 8-62). The format of remote interface readings for SCPI commands cannot be changed.

Set the displayed digits using the front panel

From the front panel:

1. From the home page, swipe the bottom view until the SETTINGS screen is displayed.
2. Next to Display Digits, select the number.
3. Select the digits to display.

This setting takes effect the next time you make measurements.

Set the displayed digits using SCPI commands

To set number of displayed digits using SCPI commands:

Send the command:

```
:DISPlay:VOLTage:DIGits <n>
```

Where `<n>` is:

- `3`: 3.5 digit resolution
- `4`: 4.5 digit resolution
- `5`: 5.5 digit resolution
- `6`: 6.5 digit resolution

To set the displayed digits for current measurements, replace `VOLTage` with `CURRent`. To set it for resistance measurements, replace `VOLTage` with `RESistance`.

Set the displayed digits using TSP commands

To set the number of displayed digits using TSP commands:

Send the commands:

```
smu.measure.func = mFunction
smu.measure.displaydigits = digits
```

Where *`mFunction`* is:

- `smu.FUNC_DC_CURRENT`: Selects current measurement
- `smu.FUNC_DC_VOLTAGE`: Selects voltage measurement
- `smu.FUNC_RESISTANCE`: Selects ohms measurement

And where *`digits`* is the number of digits:

- 6½ display digits: `smu.DIGITS_6_5`
- 5½ display digits: `smu.DIGITS_5_5`
- 4½ display digits: `smu.DIGITS_4_5`
- 3½ display digits: `smu.DIGITS_3_5`

Customizing a message for the User Display screen

You can customize the message that is displayed on the User Display screen.

You must use a remote interface to customize the User Display screen.

Creating the User Display message

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

The examples shown here switch the display to the User Display screen, set the first line to read "Test in process", and the second line to display "Do not disturb".

Using SCPI commands:

Send the commands:

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

Using TSP commands:

Send the commands:

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

Clearing the User Display screen

You can clear the message that is displayed on the User Display screen.

Using SCPI commands:

Send the command:

```
:DISPlay:CLEar
```

Using TSP commands:

Send the command:

```
display.clear()
```

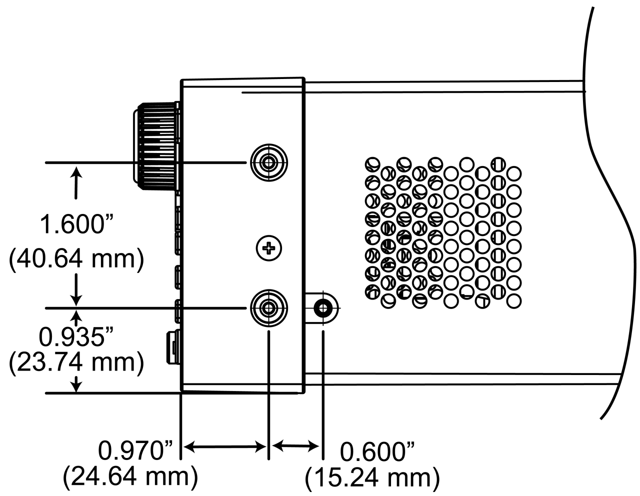
Dimensions

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

The instrument weighs 8.9 lb. (4.04 kg) with the bumpers and handle and 7.9 lb. (3.58 kg) without them.

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

Figure 22: Model 2450 mounting screw locations and dimensions



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

Figure 23: Model 2450 dimensions front and rear with handle and bumpers

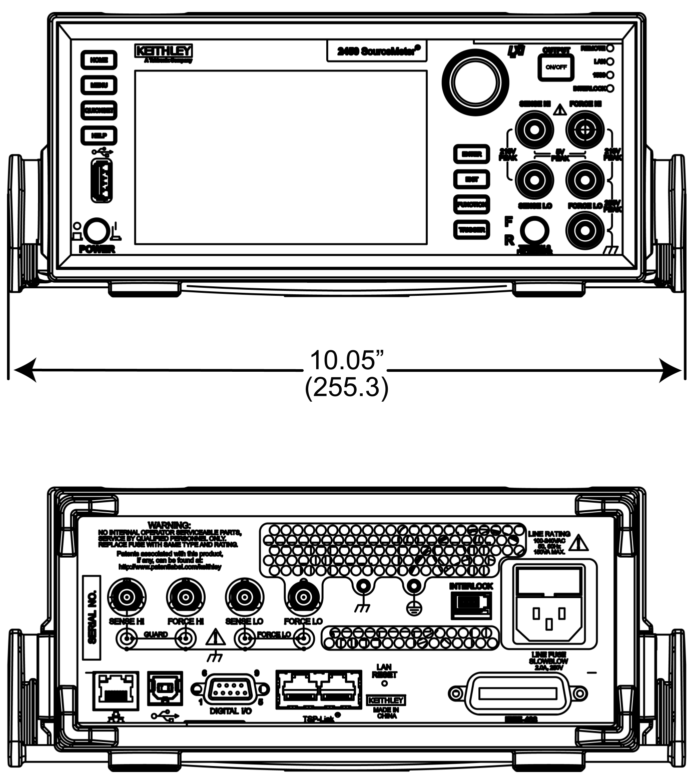
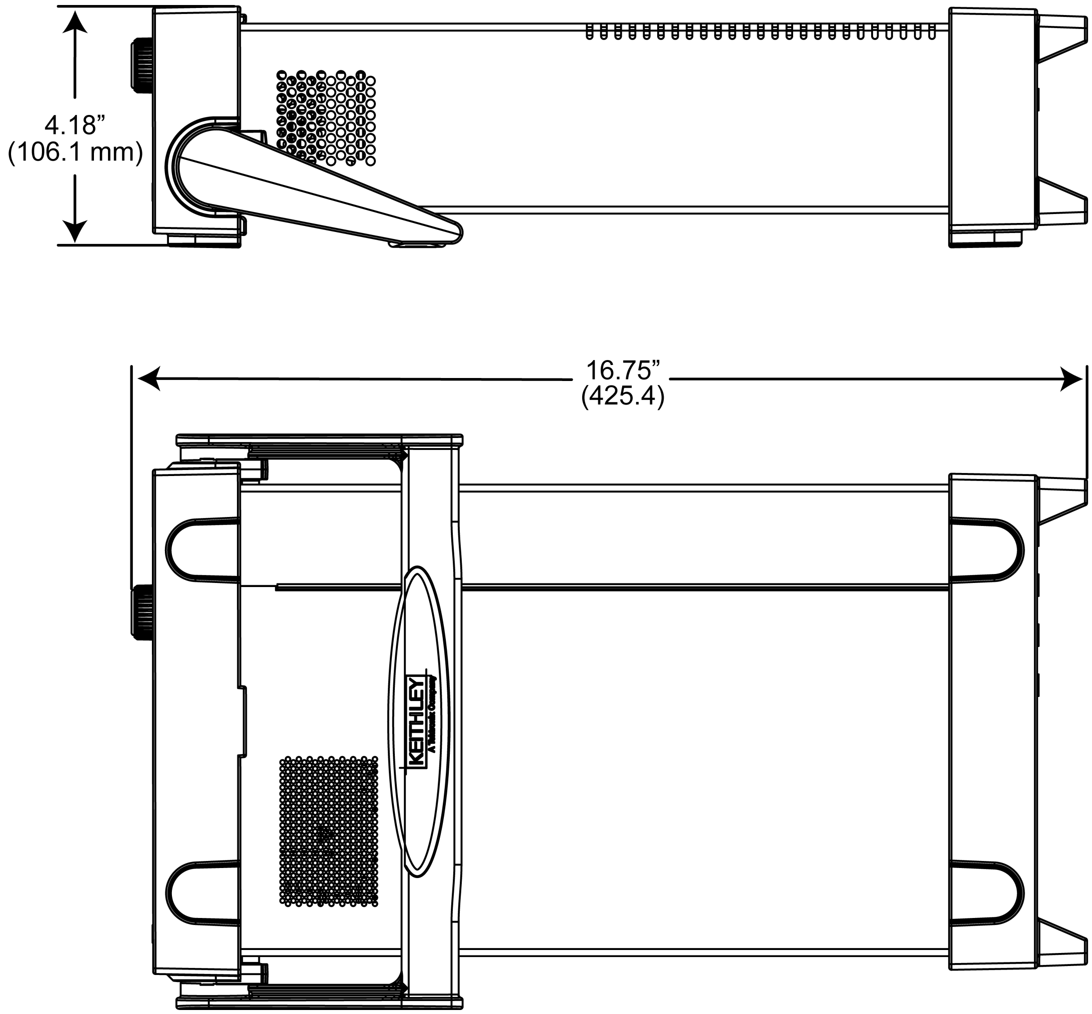


Figure 24: Model 2450 dimensions side and top with handle and bumpers



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

Figure 25: Model 2450 Front and rear panel dimensions with handle and bumpers removed

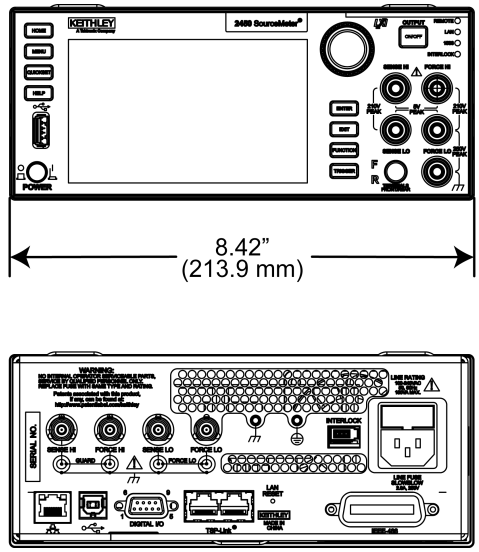
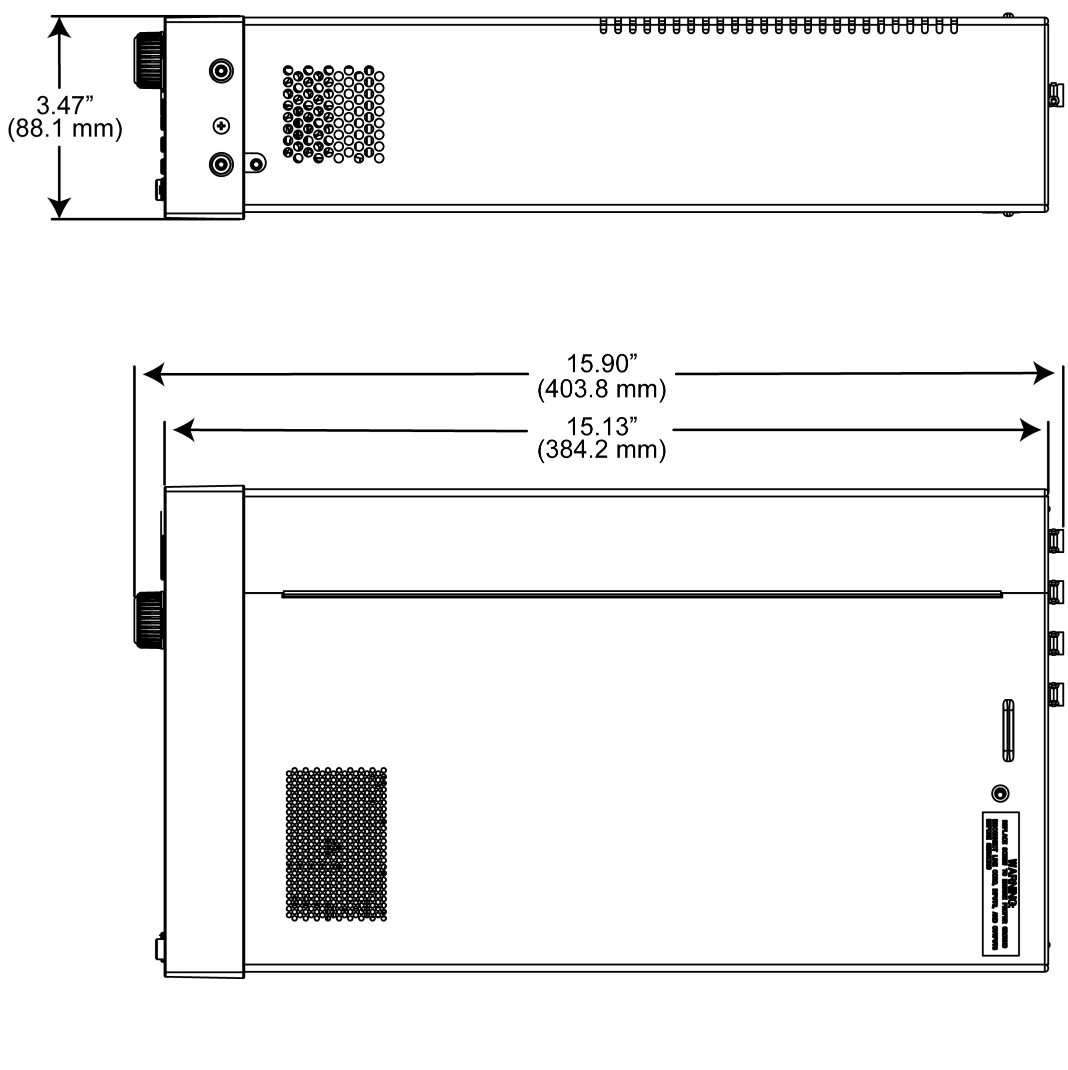


Figure 26: Model 2450 top and side dimensions with handle and bumpers removed



Handle and bumpers

The Model 2450 has a handle and front and rear bumpers for using the instrument on a benchtop. The handle rotates so that you can swing it below the bottom surface of the instrument to tilt the instrument up for easier front-panel viewing, or to carry the instrument from one location to another.

Removing the handle and bumpers

You can remove handle and bumpers on the Model 2450 if you want to mount the instrument in a rack.

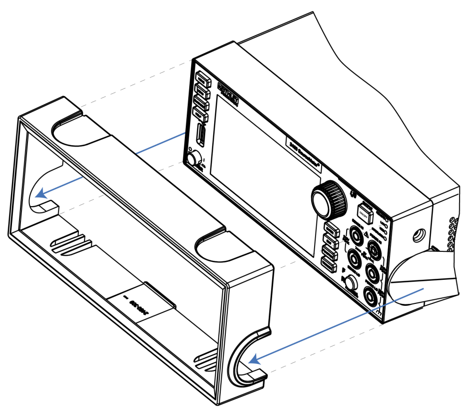
NOTE

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

To remove the bumpers:

1. Swivel the handle to a position above or below the instrument so that it will not interfere with the removal of the front bumper.
2. Grasp the front bumper on each side of the Model 2450 and gently pull it toward you until the bumper comes off of the instrument.

Figure 27: Removing the front bumper



NOTE

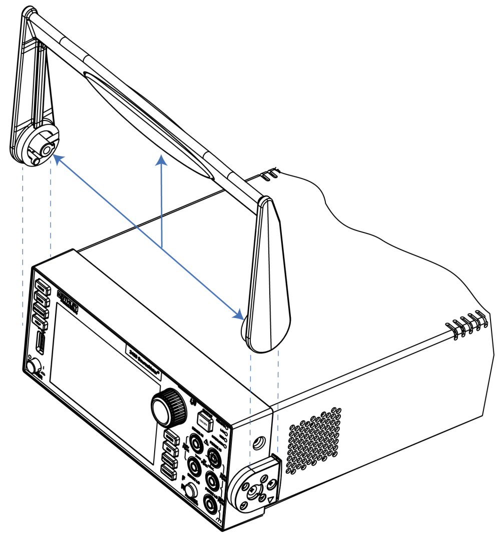
Remove all connections to the rear panel of the Model 2450 before removing the rear bumper.

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

To remove the handle assembly:

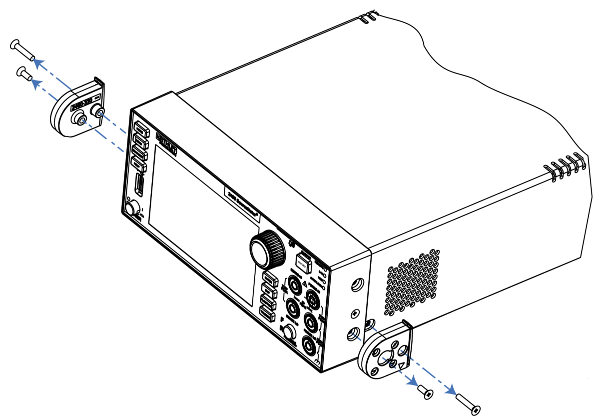
1. Grasp the sides of the handle near where it attaches to the instrument on both sides and gently pull the handle ends apart to widen the handle as you slide it over the instrument case.

Figure 28: Removing the handle



2. Using a Phillips screwdriver, loosen and remove the two screws holding the handle-mount assembly to one side of the Model 2450. The handle-mount assembly will fall away from the instrument chassis when the screws are removed.

Figure 29: Remove the handle-mount assembly



3. Repeat step 2 on the other side of the Model 2450.
4. Store the handle-mount assembly, screws, and handle together for future use.

Remote communication interfaces

You can choose from one of several communication interfaces to send commands to and receive responses from the Model 2450.

The Model 2450 can be controlled from only one communication interface at a time. The first interface on which it receives a message takes control of the instrument. If another interface sends a message, that interface takes control of the instrument.

Supported remote interfaces

The Model 2450 supports the following remote interfaces:

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

For details about TSP-Link, see TSP-Link System Expansion Interface (on page 3-118).

Comparison of the communication interfaces

The following topics discuss some of the advantages and disadvantages of the communication interfaces that are available for the Model 2450.

Simplicity

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

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

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

Triggering

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

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

Transfer rate

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

Instrument naming

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

Distance and instrument limitations

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

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

Expense

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

GPIB setup

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

The Model 2450 GPIB interface is IEEE Std 488.1 compliant and supports IEEE Std 488.2 common commands and status model topology.

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

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

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

Install the GPIB driver software

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

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

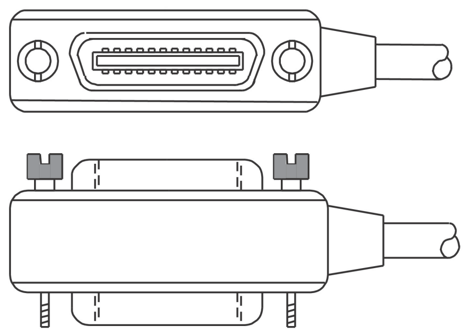
Install the GPIB cards in your computer

Refer to the manufacturer's documentation for information about installing the GPIB cards.

Connect the GPIB cables to your instrument

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

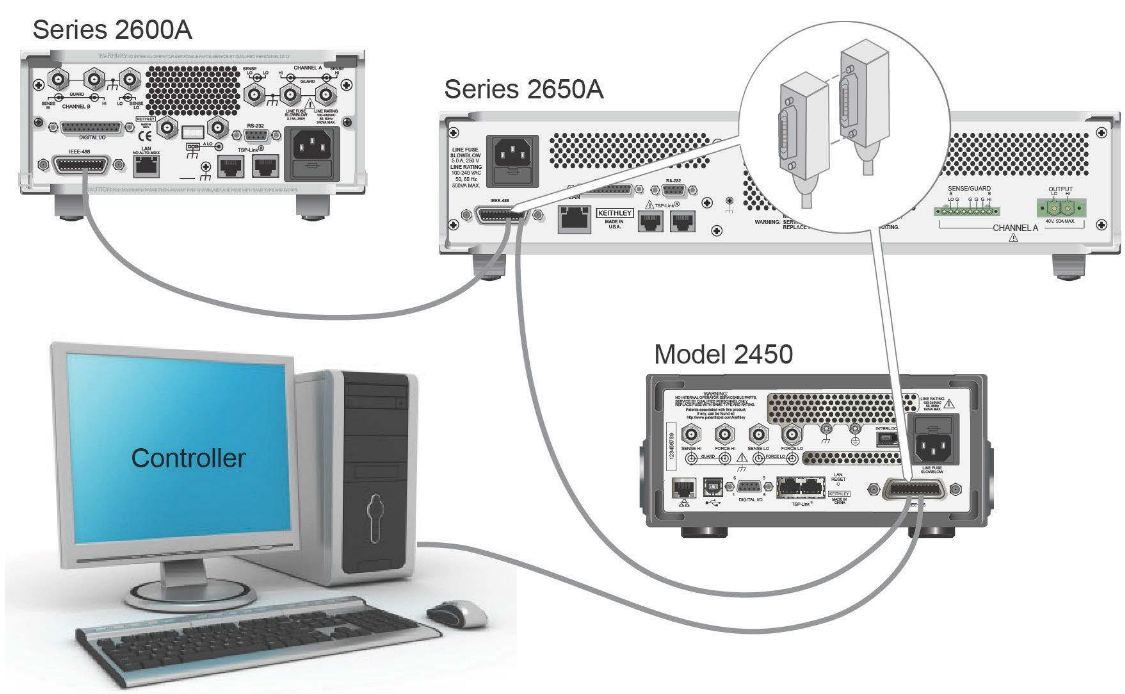
Figure 30: GPIB connector



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

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

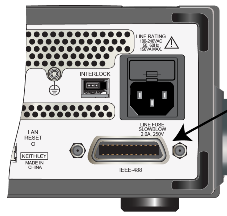
Figure 31: IEEE-488 connection example



To connect the instrument to the GPIB:

1. Align the cable connector with the connector on the Model 2450 rear panel. The location of the connector is shown in the following figure.
2. Attach the connector. Tighten the screws securely but do not overtighten them.

Figure 32: Rear panel GPIB location



3. Connect any additional connectors from other instruments, as required for your application.
4. Ensure the other end of the cable is properly connected to the controller.

Set the GPIB address

The GPIB address value is set to 18 at the factory. You can set the address to any address between 0 and 30 if it is unique in the system. This address cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

Quick Tip

GPIB controllers are usually set to 0 or 21. To be safe, do not configure any instrument to have an address of 0 or 21. To change the controller address, see the documentation for the controller.

The address is saved in nonvolatile memory, so it does not change when a reset is done or when the power is turned off and then turned on again.

From the front panel:

1. Press the **MENU** key.
2. Under System, select **Communication**. The SYSTEM COMMUNICATION window opens.
3. Select the **GPIB** tab.
4. Next to Address, select the number. The Set GPIB Address dialog box is displayed.
5. Enter the address.
6. Select **OK**.

NOTE

If you are using a Model 2450 with no front panel, you can set the GPIB address with the SCPI command :SYSTem:GPIB:ADDRess (on page 6-104) or the TSP command gpib.address (on page 8-65).

Effect of GPIB line events on Model 2450

The GPIB has control lines that allow predefined information, called events, to be transferred quickly. The following information lists some of the GPIB line events and how the Model 2450 reacts to them.

DCL

This event clears the GPIB interface. When the Model 2450 detects a device clear (DCL) event, it does the following actions:

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

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

GTL

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

IFC

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

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

LLO

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

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

REN

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

You should place the instrument into remote mode before you attempt to program it over the bus.

SDC

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

When the Model 2450 detects an SDC event, it will do the following for the selected instrument:

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

An SDC does not affect instrument settings and stored data.

SPE, SPD

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

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

LAN communications

You can communicate with the instrument using a local area network (LAN). The LAN interface can be used to build flexible test systems that include web access. This section provides an overview of LAN communications for the Model 2450.

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

The Model 2450 is a LXI version 1.4 Core 2011 compliant instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet LAN). There is one LAN port (located on the rear panel of the instrument) that supports full connectivity on a 10 Mbps or 100 Mbps network. Speed is automatically detected.

The Model 2450 also supports Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are useful on a LAN with no central administration.

NOTE

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

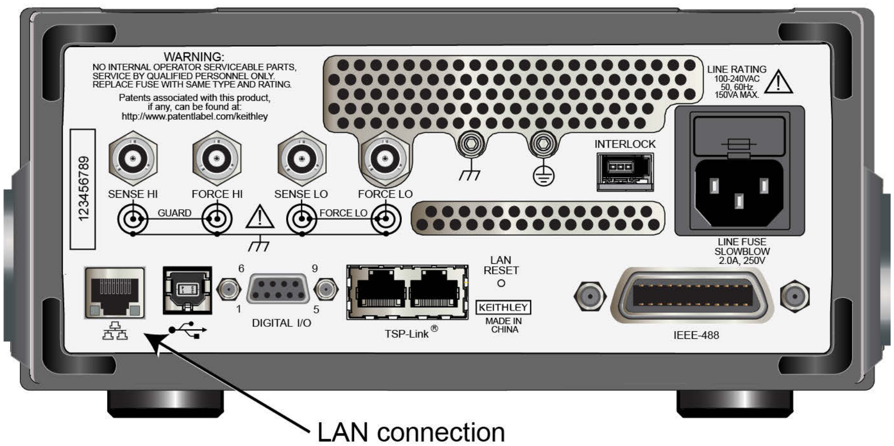
If you have problems setting up the LAN, see LAN troubleshooting suggestions (on page 2-60).

LAN cable connection

The Model 2450 includes two Model CA-180-3A cables (LAN crossover cables). One cable is for the TSP-Link® network and the other cable is for LAN communication. However, you can use any standard LAN crossover cable (RJ-45, male to male) or straight-through cable to connect your equipment. The instrument automatically senses which cable you have connected.

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

Figure 33: Model 2450 LAN connection



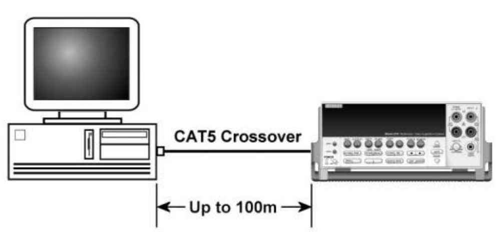
You can connect the instrument to the LAN in a one-to-one, one-to-many, two network card, or enterprise configuration, as described in the following topics.

One-to-one connection

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

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

Figure 34: One-to-one connection with a crossover cable



NOTE

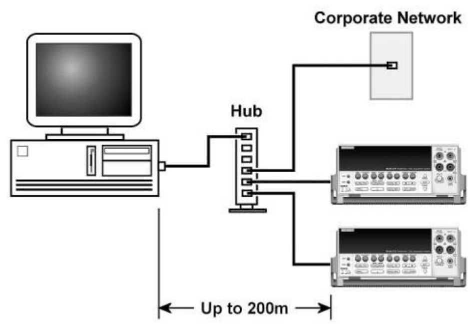
The Model 2450 supports Auto-MDIX and can use either normal LAN CAT-5 cables (patch) or crossover cables. The instrument automatically adjusts to support either cable.

One-to-many connection

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

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

Figure 35: One-to-many connection using a network hub or switch

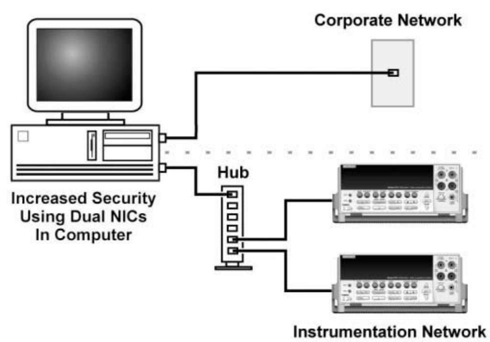


Use two network interface cards to connect to a corporate network and instrumentation hub

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

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

Figure 36: Use two network interface cards to connect to a corporate network and instrumentation hub

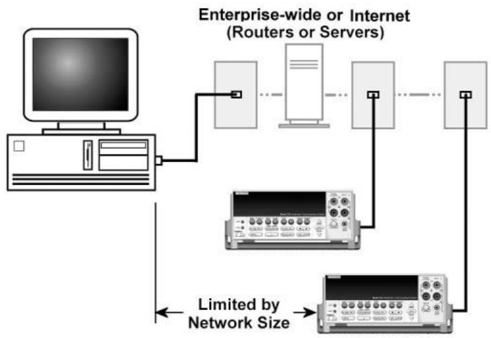


Instrumentation connection to enterprise routers or servers

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

Usually, the instruments are kept inside the corporate firewall, but the network administrator could assign resources that allow them to be outside the firewall. This would allow instruments to be connected to the Internet using appropriate security methods. Thus, data collection and distribution could be controlled from virtually any location.

Figure 37: Instrumentation connection to enterprise routers or servers



Set up LAN communications on the instrument

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

Check communication settings

Before setting up the LAN configuration, you can check the communication settings on the instrument without making any changes.

To check communication settings on the instrument:

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

NOTE

If you are using a Model 2450 with no front panel, you can check the settings with the SCPI command :SYSTem:COMMunication:LAN:CONFigure (on page 6-97) or the TSP command lan.ipconfig() (on page 8-66).

Set up automatic LAN configuration

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

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

NOTE

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

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

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

NOTE

If you are using a Model 2450 with no front panel, you can configure the LAN using SCPI or TSP commands. For details, see the SCPI command :SYSTem:COMMunication:LAN:CONFigure (on page 6-97) or the TSP command lan.ipconfig() (on page 8-66).

Set up manual LAN configuration

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

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

NOTE

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

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

To set up manual IP address selection on the instrument:

1. From the Home screen, press **MENU**.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. For TCP/IP Mode, select **Manual**.
5. Select the button next to Local IP and enter the LAN IP address. You can touch the number you want to change.
6. Select the button next to Gateway and enter the gateway address.
7. Select the button next to Subnet and enter the subnet mask.
8. Select **Apply Settings** to save your settings.

NOTE

If you are using a Model 2450 with no front panel, you can configure the LAN using SCPI or TSP commands. For details, see the SCPI command :SYSTem:COMMunication:LAN:CONFigure (on page 6-97) or the TSP command lan.ipconfig() (on page 8-66).

Set up LAN communications on the computer

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

NOTE

Do not change your IP address without consulting your system administrator. Entering an incorrect IP address can prevent your computer from connecting to your corporate network.

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

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

Wait for the LAN status indicator on the front panel to turn solid green

When the LAN status indicator on the front panel of the instrument turns solid green, it confirms that the instrument has been assigned an IP address. Note that it may take several minutes for the computer and instrument to establish a connection.

Install LXI Discovery Browser software on your computer

You can use the LXI Discovery Browser to identify the IP addresses of LXI-certified instruments. Once identified, you can double-click the IP address in the LXI Discovery Browser to open the web interface for the instrument.

The Keithley LXI Discovery Browser is available on the instrument CD.

To locate the Keithley LXI Discovery Browser on the Keithley website:

1. Select the **Support** tab.
2. In the model number box, type 2450.
3. From the list, select **Software** and click the search icon. A list of software applications for the instrument is displayed.
4. See the readme file included with the application for more information.

Run the LXI Discovery Browser

To run the LXI Discovery Browser software:

1. From the Windows Start menu, select Keithley Instruments.
2. Select LXI Discovery Browser, and then double-click **LXI Discovery Browser**. The Keithley LXI Discovery Browser window is displayed.

The Browser displays the instruments that are found on the network and their associated IP addresses.

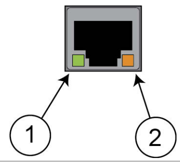
Double-click an IP address in the LXI Discovery Browser dialog box. The instrument web page for that instrument opens.

For information about using the web page, see Web interface (on page 2-57).

LAN status LEDs

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

Figure 38: LAN status LED



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

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

LAN interface protocols

You can use one of following LAN protocols to communicate with the Model 2450:

- Telnet
- VXI-11
- Raw socket

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

NOTE

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

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

LAN protocols

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

Raw socket connection

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

VXI-11 connection

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

Telnet connection

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

The Model 2450 supports the Telnet protocol, which you can use over a TCP/IP connection to send commands to the instrument. You can use a Telnet connection to interact with scripts or send real-time commands.

Dead socket connection

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

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

Web interface

When a connection between the LAN and the instrument has been established, you can open a web page for the instrument.

To access the web interface:

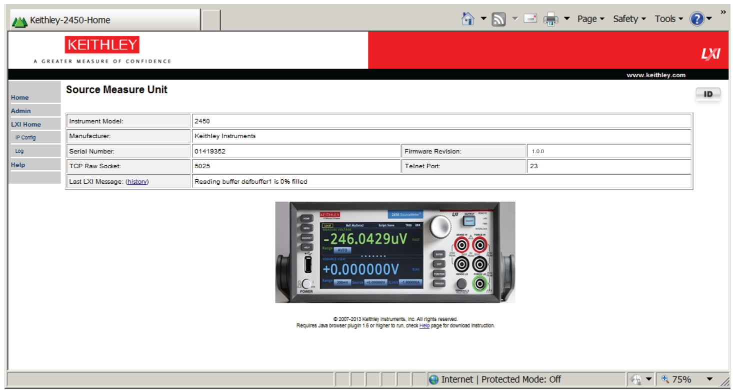
1. Open a web browser on the host computer.
2. Enter the IP address of the instrument in the address box of the web browser. For example, if the instrument IP address is 192.168.1.101, enter `192.168.1.101` in the browser address box.
3. Press **Enter** on the computer keyboard to open the instrument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

NOTE

If the web page does not open in the browser, see LAN troubleshooting suggestions (on page 2-60).

Web interface home page

Figure 39: Model 2450 web interface home page



The home page of the web interface gives you basic information about the instrument, including:

- The instrument model, serial number, firmware revision, and the last LXI message
- An **ID** button to help you locate the instrument
- Links to the instrument web options, including administrative options and LXI information

Identify the instrument

If you have a bank of instruments, you can click **ID** to determine which one you are communicating with.

Before trying to identify the instrument, make sure you have a remote connection to the instrument.

To identify the instrument:

In the upper right corner of the Home page, click .

The button turns green , and the LAN status indicator on the instrument blinks.

Click  again to return the button to its original color and return the LAN status indicator to steady on.

Change the IP configuration through the web interface

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

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

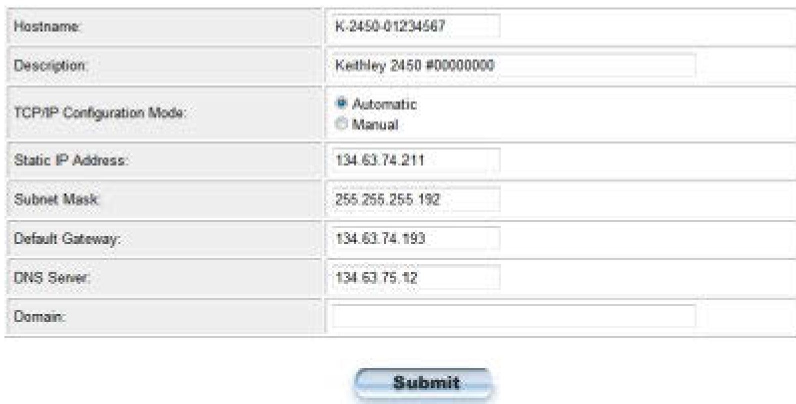
NOTE

You can also change the IP configuration through the front panel or with TSP and SCPI commands. See Set up LAN communications on the instrument (on page 2-53) for information.

To change the IP configuration using the instrument web page:

1. Access the internal web page as described in the previous topic.
2. From the navigation bar on the left, in the LXI Home menu, select **IP Config**.
3. Click **Modify**. The Modify IP Configuration page is displayed.

Figure 40: Modify IP Config web page



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

NOTE

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

Change the web interface password

You can change the instrument password from the web interface.

To change the password:

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

NOTE

The default password is `admin`.

Reviewing LAN trigger events in the event log

The event log records all LXI events generated and received by the instrument. You can view the event log using any command interface or the embedded web interface. The following information shows in the log:

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

LAN troubleshooting suggestions

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

- Verify that the network cable is in the LAN port on the rear panel of the instrument, not one of the TSP-Link® ports (see the description in Rear panel overview (on page 2-7)).
- 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 of 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.
- Restart your computer.
- Turn the instrument's power off, and then on. Wait at least 60 seconds for the network configuration to be completed. Verify that the correct settings are assigned to the instrument:
  1. Press the **MENU** key.
  2. Under System, select **Communication**.
  3. Select the LAN tab.
  4. Verify the settings.

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

USB communications

To use the rear-panel USB connection, you must have the Virtual Instrument Software Architecture (VISA) layer on the host computer. See How to install the Keithley I/O Layer (on page 2-65) for more information.

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

When a USB device that implements the USBTMC or USBTMC-USB488 protocol is plugged into the computer, the VISA driver automatically detects the device. It is important to note that only USBTMC and USBTMC-USB488 devices are automatically recognized by the VISA driver. Other USB devices, such as printers, scanners, and storage devices, are not recognized.

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

NOTE

The full version of National Instruments (NI®) VISA provides a utility to create a USB driver for any other kind of USB device that you want to communicate with VISA.

Using USB

A USB cable is shipped with the instrument. If the original cable is not available, you will need a USB cable with a USB Type B connector on one end and a USB type A connector on the other end for each instrument you plan to connect to the computer at the same time using the USB interface.

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

Communicate with the instrument

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

`USB0::0x05e6::0x2450::[serial number]::[INSTR]`

Where:

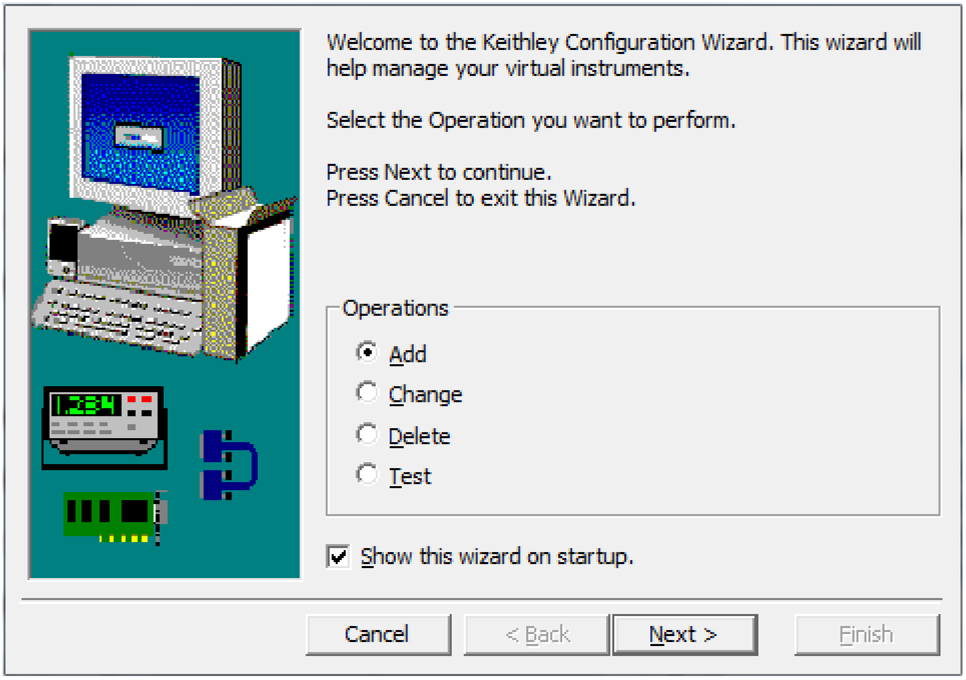
- `0x05e6`: The Keithley vendor ID
- `0x2450`: The instrument model number
- `[serial number]`: The serial number of the instrument (the serial number is also on the rear panel)
- `INSTR`: Use the USBTMC protocol

To determine these parameters, you can run the Keithley Configuration Panel, which automatically detects all instruments connected to the computer.

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

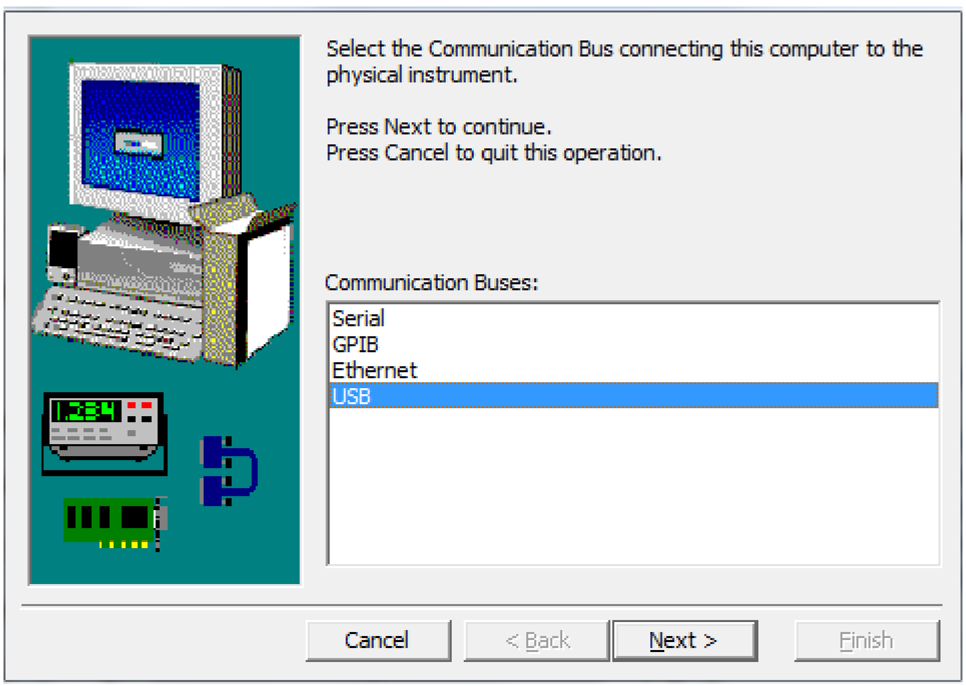
To use the Keithley Configuration Panel to determine the VISA resource string:

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



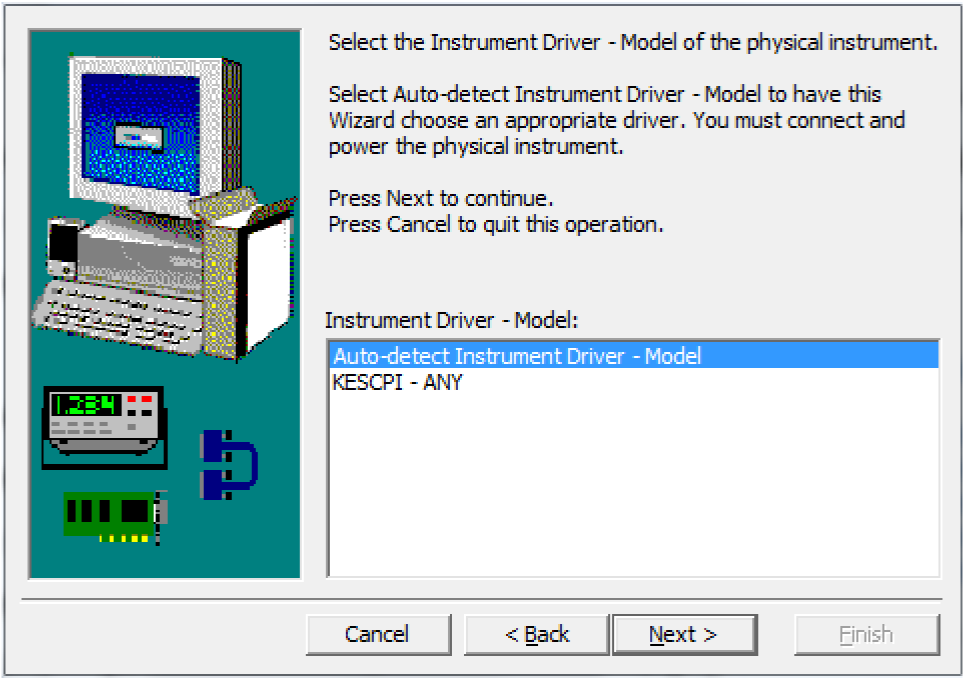
3. Click **Next**. The Select Communication Bus dialog box is displayed.

Figure 41: Select Communication Bus dialog box



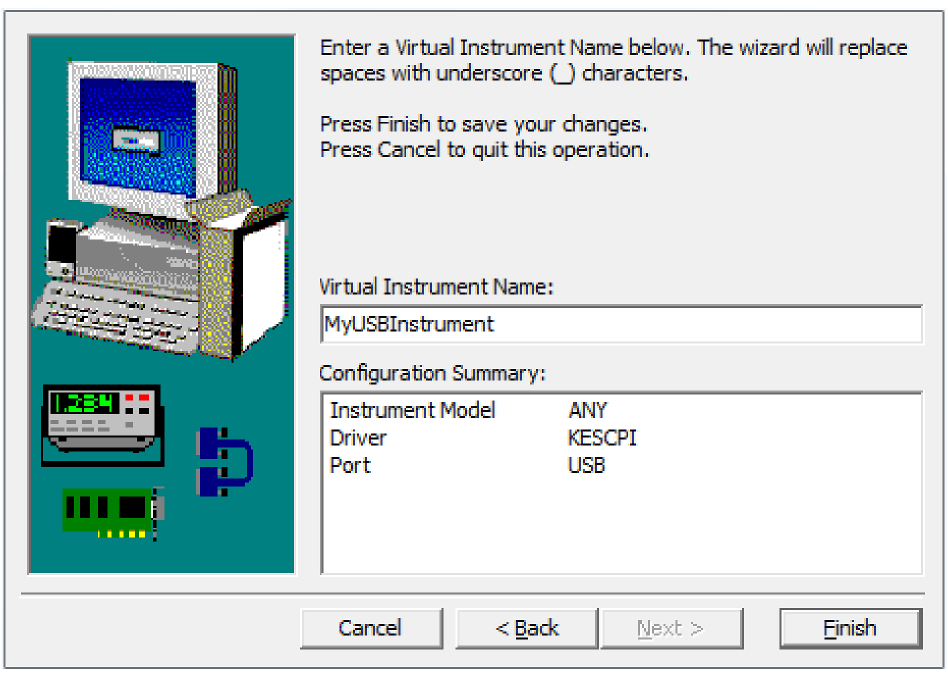
4. Select **USB**.
5. Click **Next**. The Select Instrument Driver dialog box is displayed.

Figure 42: Select Instrument Driver dialog box



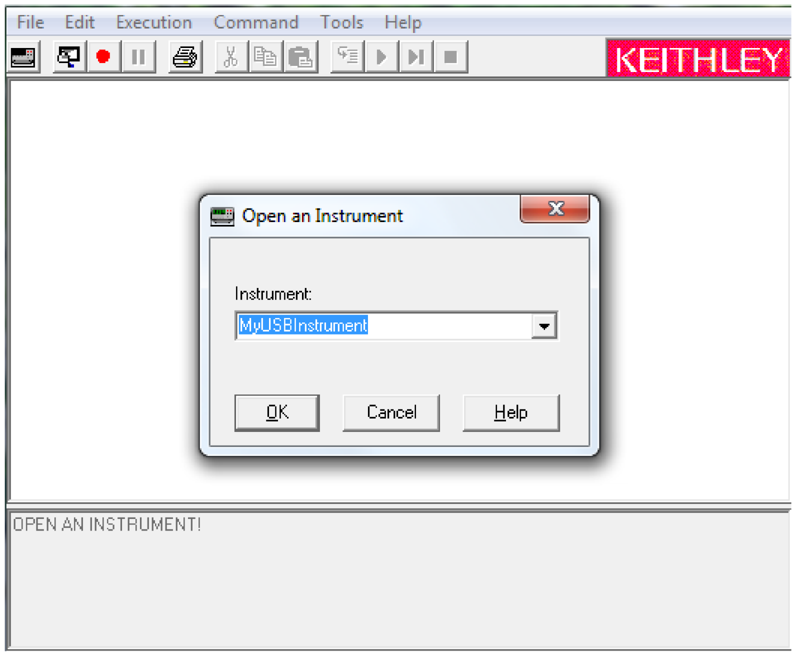
6. Select **Auto-detect Instrument Driver - Model**.
7. Click **Next**. The Configure USB Instrument dialog box is displayed with the detected instrument VISA resource string visible.
8. Click **Next**. The Name Virtual Instrument dialog box is displayed.

Figure 43: Name Virtual Instrument dialog box



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

Figure 44: Keithley Communicator Open Instrument



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

NOTE

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

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

How to install the Keithley I/O Layer

NOTE

You can install the Keithley I/O Layer from the CD-ROM that came with your instrument, or from the download from the Keithley website.

The software installs the following components:

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

To install the Keithley I/O Layer from the CD-ROM:

1. Close all programs.
2. Place the CD-ROM into your CD-ROM drive.
3. Your web browser should start automatically and display a screen with software installation links. If you need to manually open the web page, use a file explorer to navigate to the CD-ROM drive and open the file named `index.html`.
4. From the web page, select the **Software** category and click Keithley I/O Layer.
5. Accept all defaults.
6. Click **Next**.
7. Click **Install**.
8. Reboot your computer.

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

1.
2. Run the downloaded file from the temporary directory.
3. Follow the instructions on the screen to install the software.
4. Reboot your computer.

Determining the command set you will use

You can change the command set that you use with the Model 2450. The remote command sets that are available include:

- SCPI: An instrument-specific language built on the SCPI standard.
- TSP: A programming language that can be used to send individual commands or combine commands into scripts.
- SCPI 2400: Allows you to run code developed for earlier Series 2400 instruments.

You cannot combine the command sets.

As delivered from Keithley Instruments, the Model 2450 is set to work with the Model 2450 SCPI command set.

NOTE

If you choose the SCPI2400 command set, you will not have access to some of the extended ranges and other features that are now available using the SCPI command set. In addition, some Series 2400 code will work differently in the Model 2450 than it did in the earlier instrument. See Model 2450 in a Model 2400 application (on page D-1) for information about the differences.

To set the command set from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Command Set.
4. Select the command set.
5. You are prompted to reboot.

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

Send the command:

```
*LANG SCPI
```

Reboot the instrument.

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

Send the command:

```
*LANG TSP
```

Reboot the instrument.

To change to the SCPI 2400 command set from a remote interface:

Send the command:

```
*LANG SCPI2400
```

Reboot the instrument after changing the command set.

To verify which command set is selected:

Send the command:

```
*LANG?
```

System information

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

To view the system information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Information**. The system information displays.
3. To return to the Home page, select **Home**.

To view system information using SCPI commands:

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

```
*IDN?
```

To read the line frequency, send the command

```
SYStem:LFRequency?
```

The firmware build, memory available, and calibration date are not available with SCPI commands.

To view system information using TSP commands:

To read the model number, send the command:

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

To read the serial number, send the command:

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

To read the firmware version, send the command:

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

To read the line frequency, send the command

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

The firmware build and calibration date are not available with TSP commands.

You can also create user-defined strings to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location. See the TSP command reference (on page 8-1) for detail about the `userstring` functions.

Instrument sounds

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

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

To turn the off beeps when an error occurs (setting is only available from the front panel):

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

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

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

To generate an audible tone from the SCPI remote interface:

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

Where `frequency` is the frequency of the sound and `time` is the length of the sound in seconds.

To general an audible tone from the TSP command interface:

Send the following command:

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

Where *`duration`* is the length of the sound in seconds and *`frequency`* is the frequency of the sound.

Test connections

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 2450 while output is on.

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

There is no internal connection between protective earth (safety ground) and the LO terminals of the Model 2450. Therefore, hazardous voltages (more than 30 V_{rms}) can appear on LO terminals. This can occur when the instrument is operating in any mode. To prevent hazardous voltage from appearing on the LO terminals, connect the LO terminal to protective earth if your application allows it. You can connect the LO terminal to the chassis ground terminal on the front panel or the chassis ground screw terminal on the rear panel. Note that the front-panel terminals are isolated from the rear-panel terminals. Therefore, if you are using the front-panel terminals, ground to the front-panel LO terminal. If using the rear-panel terminals, ground to the rear panel LO terminal.

You can make test connections to the Model 2450 from the rear or front panel of the instrument.

The basic connection configurations for the Model 2450 include:

- Two-wire sensing
- Four-wire remote sensing
- Guard

Basic connections

WARNING

The front and rear terminals of the instrument 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 instrument terminals to CAT II, CAT III, or CAT IV circuits. Connection of the instrument terminals to circuits higher than CAT I can cause damage to the equipment and expose the operator to hazardous voltage.

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

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

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

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

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

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

When making or breaking connections, follow these guidelines:

- Power off the Model 2450 and all other instruments.
- Disconnect any devices that may deliver energy.
- Make connections to the DUT through a test fixture or other safe enclosure.
- Make sure the Model 2450 is properly connected to protective earth (safety ground).
- If the test fixture is conductive, make sure the test fixture is properly connected to protective earth (safety ground).
- Make sure the test fixture provides proper protection.
- Properly make interlock connections between the Model 2450, the test fixture, and any other instruments.
- Make sure to follow all warnings and cautions and to take adequate safety precautions for each set of connections.
- Properly terminate any triaxial cables. All unterminated cable ends must be in a safe enclosure.
- See Two-wire local sense connections (on page 2-75) and Four-wire remote sense connections (on page 2-76) for examples of connections.
- For information on the interlock, see Using the interlock (on page 2-70).

CAUTION

Combining two instruments to achieve greater currents in 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 instruments. To prevent damage to the instrument, connected instruments, and the device under test, make sure proper procedures are used.

instruments.

Using the interlock

An interlock circuit is provided on the rear panel of the instrument. This circuit must be closed to enable the Model 2450 to produce voltages greater than 38 V DC. When the safety interlock signal is asserted, the following actions occur:

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

However, when the safety interlock signal is not asserted, the following occurs:

- The nominal output is limited to ±37 V.
- The front-panel INTERLOCK indicator is not illuminated.

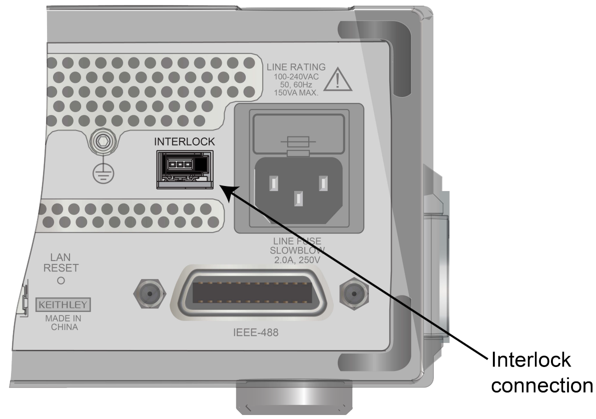
You can only use the high-voltage outputs when the interlock is asserted. If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, you see event code 5074, "Output voltage limited by interlock." Note that the SOURCE screen displays the value that was selected for the voltage source, but the source value is limited to ±37 V.

⚠ WARNING

The Model 2450 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

Location of the interlock connection

Figure 45: Rear panel interlock location



Interlock connector pins

An interlock circuit is provided on the rear panel of the instrument. This circuit must be closed to enable the Model 2450 to produce voltages greater than 38 V DC.

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

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

⚠ WARNING

Potentially hazardous voltages of up to ±210 V may be present at the High Force, High Sense, and Guard terminals when the interlock circuit is closed.
To prevent electrical shock, do not expose these lines.

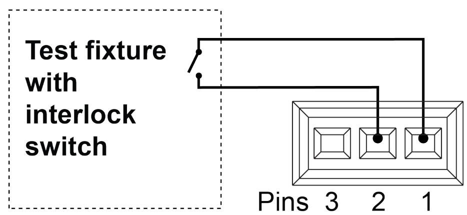
Keithley Instruments connector CS-1616-3, supplied with the Model 2450, can be used to make the interlock connection to the rear panel. You must supply connection wire.

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

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

- Pin 3: Earth and chassis ground
- Pin 2: Interlock
- Pin 1: +6 V DC out (current limited)

Figure 46: Model 2450 interlock pins



Front or rear panel test connections

You can use either the front-panel or rear-panel terminals to make connections to the device under test (DUT). You cannot make some connections to the front-panel terminals and some to the rear-panel terminals for the same test setup.

The instrument must be set to use the front or rear terminals.

Determining whether to use front or rear terminals

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

For the most precise measurements, use the rear-panel triaxial terminals. If you are making measurements or sourcing current in the 10 nA and 100 nA ranges, you must use the rear-panel triaxial terminals. You must also use the rear panel connections when making bipolar junction transistor (BJT) measurements.

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

Setting the instrument to use the front or rear terminals

NOTE

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

Using the front panel:

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

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

Using SCPI commands:

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

```
ROUTe:TERMinals FRONt
```

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

```
ROUTe:TERMinals REAR
```

Using TSP commands:

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

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

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

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

Two-wire compared to four-wire measurements

You can use 2-wire or 4-wire measurement techniques with the Model 2450.

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

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

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

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

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

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

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

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

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

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

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

NOTE

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

Two-wire local sense connections

Two-wire connections are shown in the following figure.

If your application results in impedances above 1 GΩ, you may need to also use guarding. This prevents leakage current from affecting measurement accuracy. For information, see Guarding (on page 4-14).

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

Two-wire local sense connection drawings

Figure 47: Two-wire DUT connections to rear panel

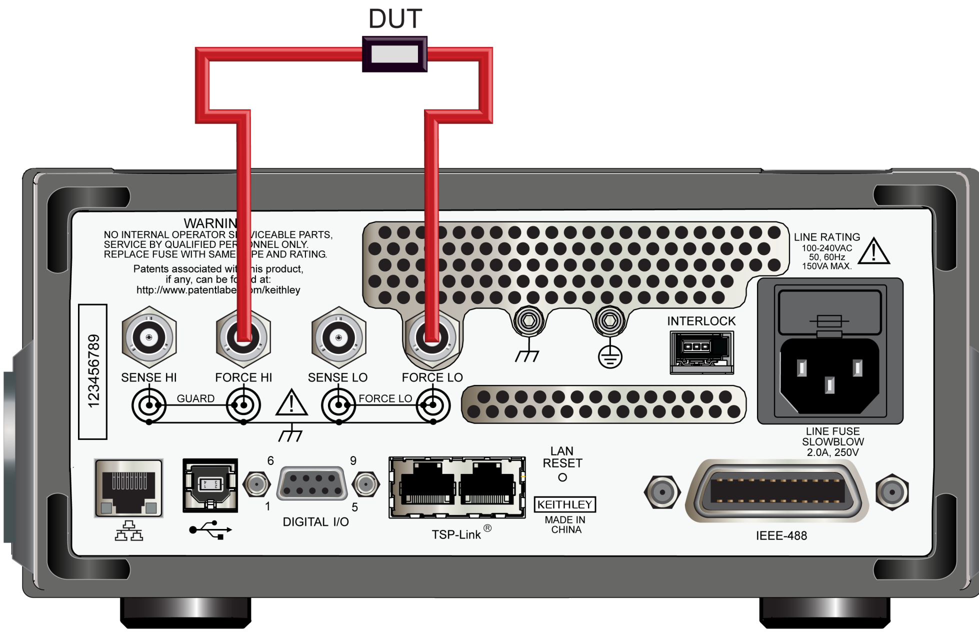


Figure 48: Two-wire DUT connections to the front panel

Using the front panel:

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

Using SCPI commands:

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

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

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

Using TSP commands:

For voltage measurements, send the commands:

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

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

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

Four-wire remote sense connections

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

If you use 4-wire remote sensing when you source voltage, the programmed voltage is delivered to the device under test (DUT). If you use 4-wire remote sensing when you source current and measure voltage, only the voltage drop across the DUT is measured.

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

NOTE

When you are sourcing voltage in 4-wire remote sense, connect the sense leads to the DUT. If a sense lead is disconnected, the instrument senses 0 V, which causes it to increase the output voltage to compensate. To further protect against overvoltage situations, you can set overvoltage protection. See Overvoltage protection (on page 2-98) for more information.

Four-wire remote sense connection drawings

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

Figure 49: Model 2450 rear panel 4-wire remote sense connections

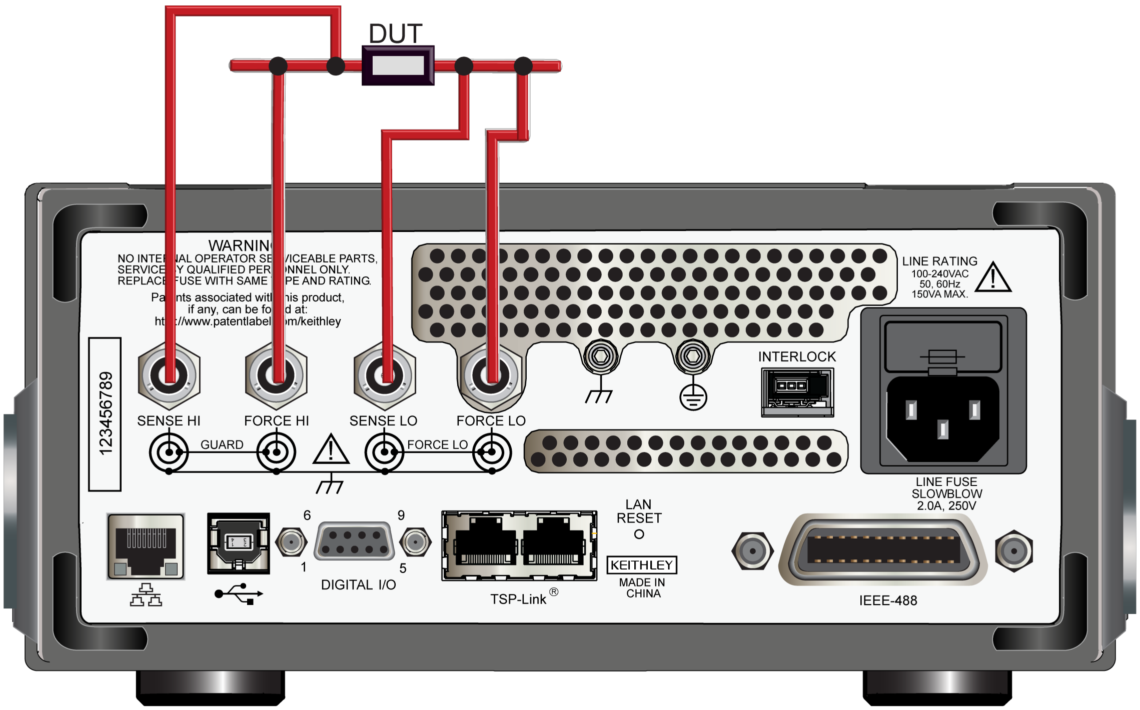
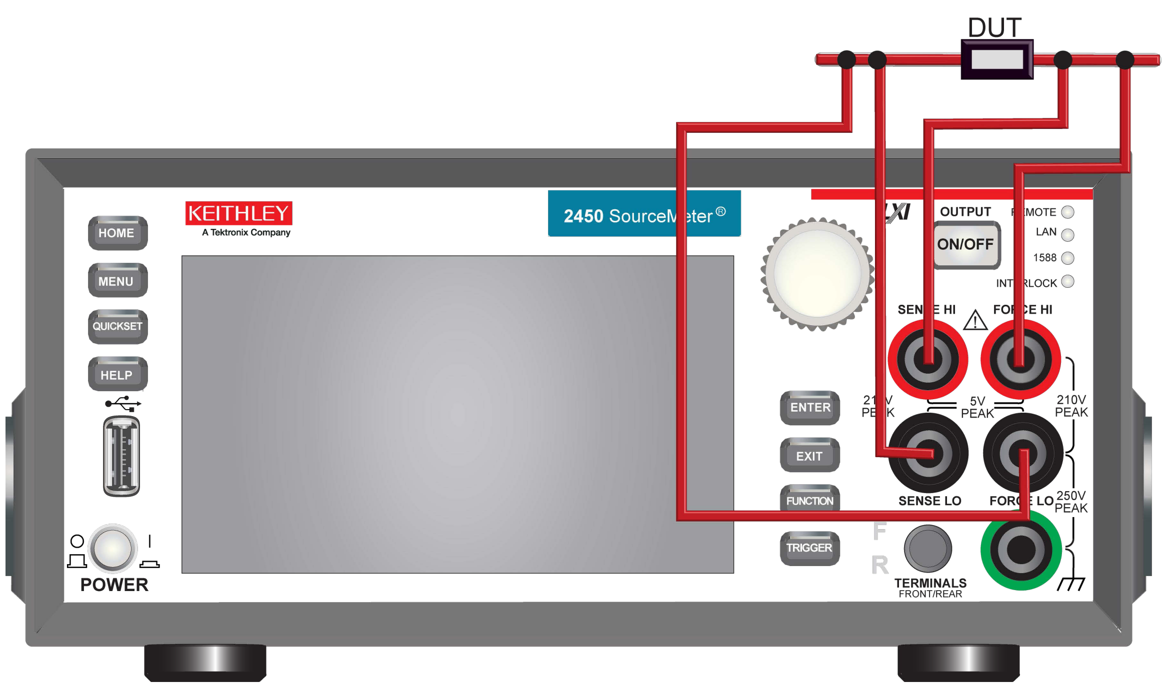


Figure 50: Model 2450 front panel 4-wire remote sense connections



Set the instrument to 4-wire sense

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

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

NOTE

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

Using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measurement combination.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. Select the button next to Sense Mode. The Sense Mode dialog box is displayed.
6. Select **4-Wire Sense**.
7. Select **HOME** to return to the operating display.

Using SCPI commands:

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

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

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

Using TSP commands:

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

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

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

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

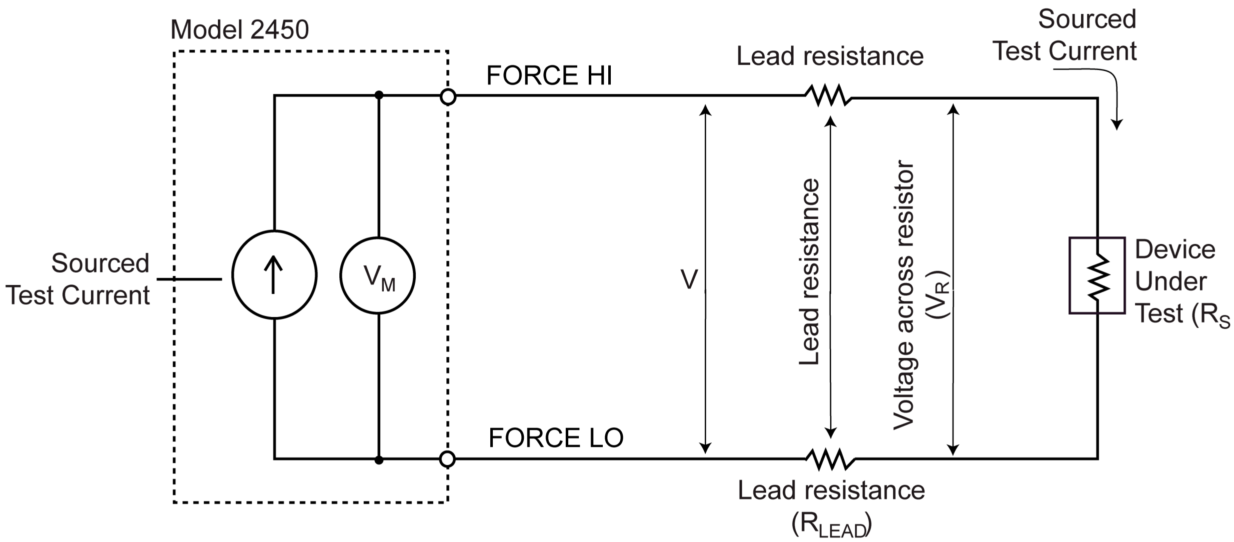
Ohms measurements

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

Accuracy of 2-wire resistance measurements

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

Figure 51: Two-wire resistance sensing for high impedance DUT

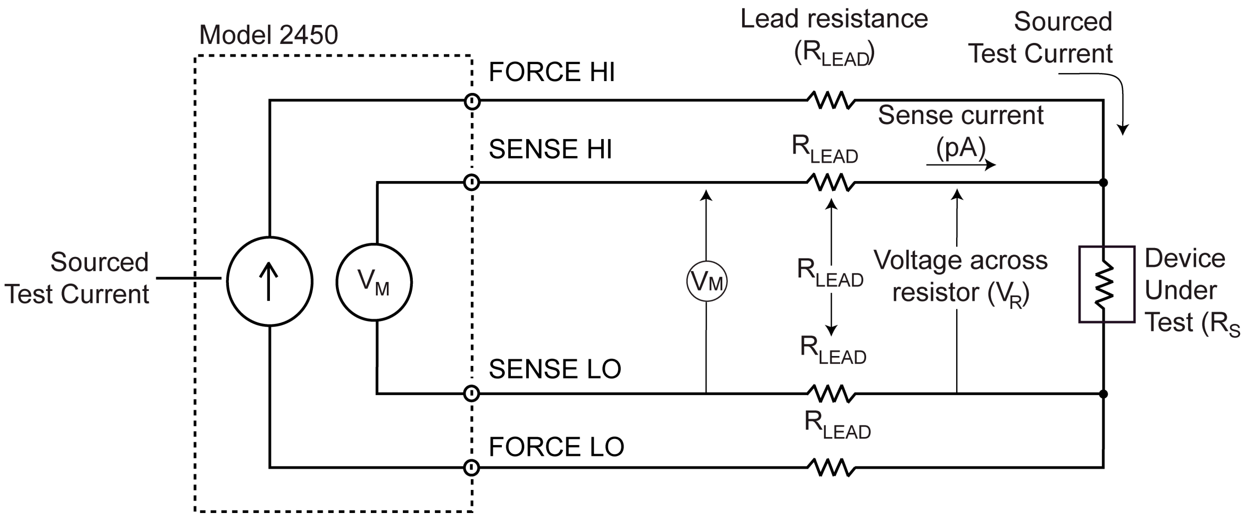


Measured resistance: $\frac{V_M}{I} = R_s + (2 \times R_{LEAD})$

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

Minimizing the effect of lead resistance with 4-wire testing

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



Sense current is negligible, therefore $V_M = V_R$

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

Test fixtures

A test fixture can be used to house a device or test circuit. The test fixture can be a metal or nonconductive enclosure, and is typically equipped with a lid. When the test fixture is correctly connected using the interlock, the output of the Model 2450 will turn off when the lid of the test fixture is opened. 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.

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 2450 includes an interlock connector on the rear panel of the instrument. When properly connected to a test fixture, the output of the Model 2450 turns off when the lid of the test fixture is opened.

Output-off state

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and limits before connecting the Model 2450 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

When the source of the instrument is turned off, it may not completely isolate the instrument from the external circuit. You can use the output-off setting to place the Model 2450 in a known, noninteractive state during idle periods, such as when changing the device under test. The output-off states that can be selected for a Model 2450 are normal, high-impedance, zero, or guard.

When you change the output-off state, the selected output-off state is changed immediately. On power-up, the instrument is momentarily in the high-impedance output-off state before going to the selected output-off state.

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

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

Normal output-off state

When the Model 2450 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10 percent of the full scale of the present current range
- OUTPUT OFF is displayed in the Home page Source area
- The Source button on the Home page shows the output that will be sourced when the output is turned on again

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

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

High-impedance output-off state

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

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

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

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

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

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

Zero output-off state

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

- The measurement sense is changed to 2-wire sense
- The source function is set to voltage
- The source voltage is set to 0
- Set the range to the presently selected range (turn off autorange)
- Program the voltage DAC to zero, and the current DAC to full scale of the present current range.

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

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

Guard output-off state

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

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

- The measurement sense is changed to 2-wire sense
- The current source is selected and set to 0 A
- The voltage limit is set to 10% full scale of the present voltage range

Output-off states and inductive loads

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

Setting the output-off state

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

Using the front panel:

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

Using SCPI commands:

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

```
:OUTPut:SMODe NORMal
```

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

```
:OUTPut:SMODe ZERO
```

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

```
:OUTPut:SMODe HIMPedance
```

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

```
:OUTPut:SMODe GUARd
```

Using TSP commands:

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

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

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

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

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

```
smu.source.offmode = smu.OFFMODE_HIGHZ
```

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

```
smu.source.offmode = smu.OFFMODE_GUARD
```

Source-measure overview

Using the Model 2450, you can perform the following operations:

- Source voltage and measure current, voltage, resistance, or power
- Source current and measure voltage, current, resistance, or power
- Measure voltage, current, resistance, or power

NOTE

Because many parameters you set are related to a specific function, make sure you select the source and measure functions before you make changes to other instrument settings. The settings options you have depend upon the active source and measure functions that are selected when you make the changes. If you make a change that is not compatible with the active source and measure functions, you may get unexpected results or you may receive an event message.

When you are making measurements, you can set the measurement range to a specific range or to automatic ranging. If you select a specific range, for the best accuracy, use the lowest possible range. When Auto is selected, the instrument selects the most sensitive range to make a measurement or will change automatically as the measured value of the device changes.

The operating boundaries for the source-measure operations are provided in Operating boundaries (on page 4-4).

WARNING

Hazardous voltages may be present on all output and guard terminals. To prevent electrical shock that could cause injury or death, never make or break connections to the Model 2450 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 2450 before handling cables. Putting the equipment into an output-off state does not guarantee that the outputs are powered off if a hardware or software fault occurs.

Source and measure through the front panel

You can source and measure through the options available on the front panel.

Using One-Touch Quick Setups

The One-Touch Quick Setups allow you to select predefined setups. The options include measurement choices that make measurements while sourcing a 0 value. The power supply option sources a voltage without making a measurement.

The Quick Setup options that are available include Voltmeter, Ammeter, Ohmmeter, and Power Supply.

For detail on the values that are set when you select these options, see QuickSet controls menu (on page 2-20).

CAUTION

When using the Model 2450 as a voltmeter, the voltage limit must be set higher than the voltage that is being measured. Failure to do this could result in damage to the instrument because of excessive current flow into the instrument.

Making a measurement with the QuickSet functions

The measurement-only functions available through the QuickSet option include Voltmeter, Ammeter, and Ohmmeter.

Using the front panel:

1. Make connections to the device under test before running the Quick Setup. The Voltmeter and Ammeter options use 2-wire connections. For the Ohmmeter option, you can select 2-wire or 4-wire connections. For information on making connections, see Test connections (on page 2-68).
2. Press **QUICKSET**.
3. Under One-Touch Quick Setups, select the type of measurement.
4. **Ohmmeter only:** Select 2-wire or 4-wire sense.
5. A message may be displayed. If the conditions of your test setup permit it, select **OK**.
6. The output turns on and the instrument begins making measurements.
7. Observe the readings.
8. You can adjust the source and measurement settings while the instrument makes measurements.
9. When finished, turn the output off by pressing the **OUTPUT ON/OFF** control. The OUTPUT indicator light switches off.

Sourcing a voltage with the QuickSet functions

The source-only function available through the QuickSet option includes Power Supply.

Using the front panel:

1. Make connections to the device under test before running the Quick Setup. You can select 2-wire or 4-wire connections. For information on making connections, see Test connections (on page 2-68).
2. Press **QUICKSET**.
3. Under One-Touch Quick Setups, select **Power Supply**.
4. Select the voltage level to be sourced. Select **OK**.
5. Select the maximum allowed source current. Select **OK**.
6. Select 2-wire or 4-wire sense.
7. The output turns on and the instrument begins sourcing voltage.
8. You can adjust the source and measurement settings while the instrument makes measurements.
9. When finished, turn the output off by pressing the **OUTPUT ON/OFF** control. The OUTPUT indicator light switches off.

Source voltage and make measurements

When the Model 2450 is sourcing voltage, you can make current, voltage, resistance, or power measurements.

Using the front panel:

1. Connect the device under test (DUT) as described in Test connections (on page 2-68).
2. Set the function for your measurement. Press **FUNCTION**. Under Source Voltage and Measure, select the type of measurement you want to make.
3. Select the source voltage range. Under VSOURCE VIEW on the home screen, select the button next to Range. Select the appropriate setting.
4. Set the source voltage. Under SOURCE VOLTAGE on the home screen, select the number next to Source. Use the displayed number pad to set the value. Select **OK**.
5. Set current limits for the source. Under SOURCE VOLTAGE on the home screen, select the number next to Limit. Set an appropriate value.
6. Select the measurement range. In the MEASURE area of the home screen, select the button next to Range and choose an appropriate range.
7. Turn on the output by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns on.
8. Hold the **TRIGGER** key for three seconds to change the measurement method. Select one of the following options:
   - Continuous Measurement: The instrument makes continuous measurements.
   - Manual Trigger Mode: The instrument makes measurements when you press the TRIGGER key.
9. Observe the readings.
10. You can adjust the settings while the instrument makes measurements.
11. When you are finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Source current and make measurements

When the Model 2450 is sourcing current, you can make current, voltage, resistance, or power measurements.

Using the front panel:

1. Connect the device under test (DUT) as described in Test connections (on page 2-68).
2. Set the function for your measurement. Press **FUNCTION**. Under Source Current and Measure, select the type of measurement you want to make.
3. Select the source current range. Under SOURCE CURRENT on the home screen, select the button next to Range. Select the appropriate setting.
4. Set the source current level. Under SOURCE CURRENT on the home screen, select the number next to Source. Use the displayed number pad to set the value. Select **OK**.
5. Set voltage limits for the source. Under SOURCE CURRENT on the home screen, select the number next to Limit. Set an appropriate value.
6. Select the measurement range. In the MEASURE area of the home screen, select the button next to Range and choose an appropriate range.
7. Turn on the output by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns on.
8. Hold the **TRIGGER** key for three seconds to change the measurement method. Select one of the following options:
   - Continuous Measurement: The instrument makes continuous measurements.
   - Manual Trigger Mode: The instrument makes measurements when you press the TRIGGER key.
9. Observe the readings.
10. When you are finished, turn the output off by pressing the **OUTPUT ON/OFF** switch. The OUTPUT indicator light turns off.

Source values

When you edit the source value, the source is updated immediately. This allows you to adjust the source value while the output is on.

In certain situations, you cannot change the source value immediately. These situations include:

- While the instrument is performing a sweep
- If offset compensation is enabled (ohms measurements only)
- If automatic ohms is enabled (ohms measurements only)

Making resistance measurements

When you make resistance measurements, the resistance is calculated using voltage and current measurements.

When source readback is on, the instrument measures both voltage and current and uses these values in the ohms calculations. When source readback is off, the instrument uses the programmed source value and the measured value in the ohms calculations. Note that the measured source value is more accurate than the programmed source value, so measurements made with source readback on are more accurate.

When you are measuring resistance, you can set the offset-compensated ohms option.

Resistance measurement methods

From the front panel, you can use one of the following methods to measure resistance with the Model 2450:

- Press **FUNCTION** and select source current and measure resistance
- Press **FUNCTION** and select source voltage and measure resistance
- Press **QUICKSET** and select **Ohmmeter**. When Ohmmeter is selected, the source current and source limit are set automatically.

From a remote interface, you can use one of following methods to measure resistance with the Model 2450:

- Source voltage, measure current, and set measure units to ohms
- Source current, measure voltage, and set measure units to ohms
- Set the measure function to resistance. This sets the instrument to auto-ohms, which causes the instrument to set the source current and source limit automatically.

Each of these methods is described in the following topics.

Source voltage, measure current, and read ohms

If you want to make resistance readings by sourcing voltage and measuring current, you can use this method.

The examples below use a 100 kΩ device under test. The code:

- Makes five readings by sourcing 5 V
- Measures current with autorange enabled
- Sets the measure units to ohms
- Uses offset compensation
- Retrieves the source and measure values

Even though the measurement units are in ohms, the measurement range is 10 μA.

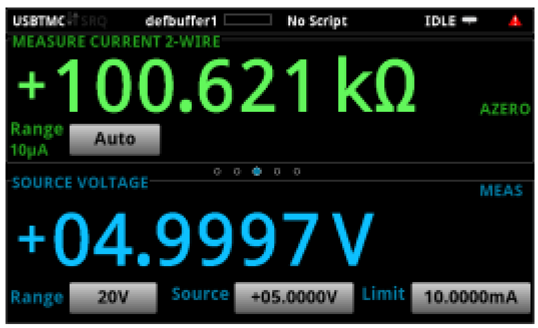
Using SCPI:

Send the following code:

```
*RST
SENSe:FUNCtion "CURR"
SENSe:CURRent:RANGe:AUTO ON
SENSe:CURRent:UNIT OHM
SENSe:CURRent:OCOM ON
SOURce:FUNCtion VOLT
SOURce:VOLT 5
SOURce:VOLT:ILIM 0.01
SENSe:COUNT 5
OUTPut ON
TRACe:TRIGger "defbuffer1"
TRACe:DATA? 1, 5, "defbuffer1", SOUR, READ
OUTPut OFF
```

The front-panel display will look similar to the following example.

Figure 52: Resistance measurement using SVMI and reading ohms



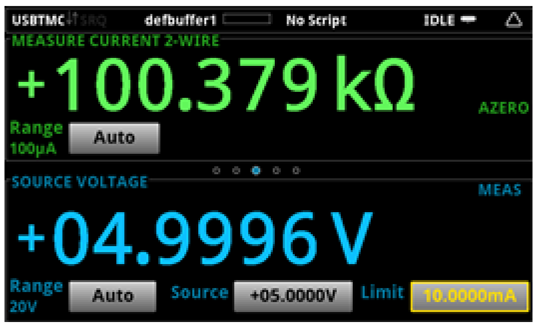
Using TSP commands:

Send the following code:

```
reset()
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.ON
smu.measure.unit = smu.UNIT_OHM
smu.measure.count = 5
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.level = 5
smu.source.ilimit.level = 0.01
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
for i=1, defbuffer1.n do
   print(defbuffer1.relativetimestamps[i], defbuffer1[i])
end
smu.source.output=smu.OFF
```

The front-panel display will look similar to the following example.

Figure 53: Resistance measurement SVMI and reading ohms



Source current, measure voltage, and set measure units to ohms

If you want to make resistance readings by sourcing current and measuring voltage, you can use this method.

The examples below use a 100 kΩ device under test. The code:

- Makes five readings by sourcing 5e-6 A
- Measures voltage with autorange enabled
- Sets the measure units to ohms
- Uses offset compensation
- Retrieves the source and measure values

Even though the measurement units are in ohms, the measurement range is 2 V.

Using SCPI:

Send the following code:

```
*RST
SENSe:FUNCtion "VOLT"
SENSe:VOLTage:RANGe:AUTO ON
SENSe:VOLTage:UNIT OHM
SENSe:VOLTage:OCOM ON
SOURce:FUNCtion CURR
SOURce:CURRent 5e-6
SOURce:CURRent:VLIM 10
SENSe:COUNT 5
OUTPut ON
TRACe:TRIGger "defbuffer1”
TRACe:DATA? 1, 5, "defbuffer1", SOUR, READ
OUTPut OFF
```

The front-panel display will look similar to the following example.

Figure 54: Resistance measurement SIMV SCPI example

Using TSP commands:

Send the following code:

```
reset()
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
smu.measure.unit = smu.UNIT_OHM
smu.measure.count = 5
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.level = 5e-6
smu.source.vlimit.level = 10
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
for i=1, defbuffer1.n do
   print(defbuffer1.relativetimestamps[i], defbuffer1[i])
end
smu.source.output=smu.OFF
```

The front-panel display will look similar to the following example.

Figure 55: Resistance measurement SIMV and read ohms

Measure resistance and use the auto-ohms (ohmmeter) mode

You can use the auto-ohms to measure resistances. When you use auto-ohms, the Model 2450 measures resistances by sourcing current. The instrument automatically sets the magnitude of the current source, voltage limit, and the measure range.

This mode is the same as the Ohmmeter One-Touch Quick Setup, which is available by pressing the QUICKSET button.

The examples below use a 100 kΩ device under test. The code makes five readings. Note that the measurement range is 200 kΩ.

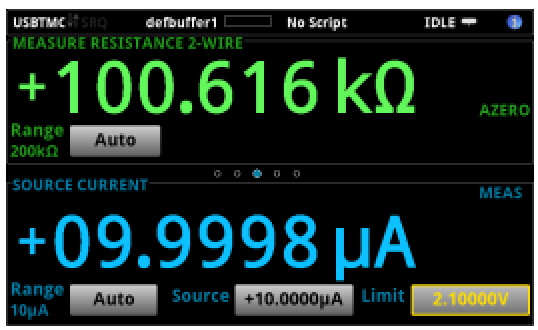
Using SCPI:

Send the following code:

```
*RST
SENSe:FUNCtion "RES"
SENSe:RESistance:RANGe:AUTO ON
SENSe:RESistance:OCOMpensated ON
SENSe:COUNt 5
OUTPut ON
TRACe:TRIGger "defbuffer1"
TRACe:DATA? 1, 5, "defbuffer1", SOUR, READ
OUTPut OFF
```

The front-panel display will look similar to the following example.

Figure 56: Resistance measurement using auto-ohms



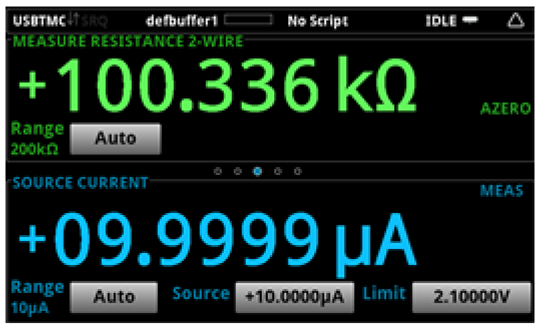
Using TSP:

Send the following code:

```
reset()
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.autorange = smu.ON
smu.measure.count = 5
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
for i=1, defbuffer1.n do
   print(defbuffer1.relativetimestamps[i], defbuffer1[i])
end
smu.source.output=smu.OFF
```

The front-panel display will look similar to the following example.

Figure 57: Resistance measurement autoohm



Offset-compensated ohms

The voltage offsets because of the presence of thermal EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

See Offset-compensated ohm calculations (on page 4-20) for additional detail on calculating offset-compensated ohms.

Setting offset-compensated ohms

Using the front panel:

NOTE

This setting is only available from the front panel when the resistance measurement function is selected.

1. Press the **MENU** key.
2. Under Measure, select **Settings**.
3. Next to Offset Comp, select **On**.
4. Select **HOME** to return to the operating display.

Using SCPI commands:

To enable offset-compensated ohms, send the command:

```
SENSe:RESistance:OCOMpensated ON
```

Using TSP commands:

To enable offset-compensated ohms, send the commands:

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.offsetcompensation = smu.ON
```

Source and measure using SCPI commands

The SCPI commands that set up the source functions are in the SOURce subsystem.

The source commands are specific to each source function (voltage or current). For example, to set the range to 100 mA for the current function, you would send:

```
:SOURce:FUNCtion CURRent
:SOURce:CURRent:RANGe 1e-01
```

To set the range to 20 V for the voltage function, you would send:

```
:SOURce:FUNCtion VOLTage
:SOURce:VOLTage:RANGe 20
```

The SCPI commands that set up the measurement functions are in the SENSe subsystem.

The sense commands are also specific to each measure function (voltage, current, or resistance). For example, to set the NPLC cycles to 0.5 for the current measurement function, you would send:

```
:SENSe:CURRent:NPLCycles .5
```

For the voltage measurement function, you would send:

```
:SENSe:VOLTage:NPLCycles .5
```

For the resistance measurement function, you would send:

```
:SENSe:RESistance:NPLCycles .5
```

To make a measurement, you send the `MEASure:<function>?` command. For example, to make a current measurement, send the command:

```
:MEASure:CURRent?
```

To make a voltage measurement, send the command:

```
:MEASure:VOLTage?
```

To make a resistance measurement, send the command:

```
:MEASure:RESistance?
```

For detailed application examples that use the SCPI command language, see the User's Manual.

Command descriptions are provided in the SCPI command reference (on page 6-1).

Source and measure using TSP commands

The TSP commands that set up the source functions begin with `smu.source`.

The source commands are specific to each source function (voltage or current). For example, to set the range to 100 mA for the current source function, you would send:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = .1
```

To set the range to 20 V for the voltage function, you would send:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
```

The TSP commands that set up the measurement functions begin with `smu.measure`.

The sense commands are also specific to each measure function (voltage, current, or resistance). For example, to set the NPLC cycles to 0.5 for the current measurement function, you would send:

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.nplc = .5
```

For the voltage measurement function, you would send:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.measure.nplc = .5
```

For the resistance measurement function, you would send:

```
smu.source.func = smu.FUNC_RESISTANCE
smu.measure.nplc = .5
```

To make a measurement, you set the measurement function and then send the `smu.measure.read()` command. For example, to make a current measurement, send the commands:

```
smu.measure.func = smu.FUNC_CURRENT
print(smu.measure.read())
```

To make a voltage measurement, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
print(smu.measure.read())
```

To make a resistance measurement, send the commands:

```
smu.measure.func = smu.FUNC_RESISTANCE
print(smu.measure.read())
```

For detailed application examples that use the TSP command language, see the User's Manual.

Command descriptions are provided in the TSP command reference (on page 8-1).

Protection

The Model 2450 provides several methods for ensuring that the source remains within certain values. This helps to protect the DUT from damage.

The protections that affect the source are the:

- Overvoltage protection. This is the voltage at the instrument terminals.
- Source limits. This is the sourced value at the device.

Overvoltage protection

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced. This protects the device under test (DUT) from high voltage levels.

For example, if a sense lead is disconnected or broken during a 4-wire sense measurement, the instrument can interpret the missing sense lead as a decrease in voltage and respond by increasing the source output. If overvoltage protection is set, the sourced output is not allowed to exceed the overvoltage protection limit.

The value set for overvoltage protection takes precedence over the source limit settings. When it is enabled, it is always in effect.

When overvoltage protection is set and the sourced voltage exceeds the setting:

- The output is clamped at the overvoltage protection value
- On the front panel, an indicator to the right of the voltage displays OVP

When overvoltage protection is used in a test sequence, it should be set before turning the source on.

⚠ WARNING

Even with the overvoltage protection set to the lowest value (2 V), never touch anything connected to the terminals of the Model 2450 when the output is on. Always assume that a hazardous voltage (greater than 30 V rms) is present when the output is on. To prevent damage to the DUT or external circuitry, do not set the voltage source to levels that exceed the value that is set for overvoltage protection.

Setting overvoltage protection levels

Overvoltage protection is set to preset value.

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Next to Overvoltage Protection Limit, select the button.
4. Select the limit.
5. Select **HOME** to return to the operating display.

Using SCPI commands:

Send the `:SOURce:VOLTage:PROTection` command with the value of the limit. For example, to set the overvoltage limit for the voltage source to 20 V, send the command:

```
SOURce:VOLTage:PROTection PROT20
```

See the command description for :SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73) for the full list of options.

Using TSP commands:

Set the source function and send the `smu.source.protect.level` command with the value of the limit. For example, to set the overvoltage limit to 20 V for the voltage source function, send the commands:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.protect.level = smu.PROTECT_20V
```

See the command description for smu.source.protect.level (on page 8-141) for the full list of options.

Source limits

The source limits (also known as compliance) prevent the instrument from sourcing a voltage or current over a set value. This helps prevent damage to the device under test (DUT).

The values that can be set for the limits must be below the setting for the overvoltage protection limit.

This limit can also be restricted by the measurement range. If a specific measurement range is set, the limit must be more than 0.1 percent of the measurement range. If not, an event is generated and the limit is automatically changed to an appropriate value for the selected range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

If you attempt to change the source limit to a value that is not appropriate for the selected source range, the source limit is not changed and a warning is generated. You must change the source range before you can select the new limit.

The lowest allowable limit is based on the load and the source value. For example, if you are sourcing 1 V to a 1 kohm resistor, the lowest allowable current limit is 1 mA (1 V/1 kohm = 1 mA). Setting a limit lower than 1 mA limits the source.

For example, assume the following conditions:

- Current limit 10 mA
- Instrument sources a voltage of 10 V
- DUT resistance of 10 Ω

With a source voltage of 10 V and a DUT resistance of 10 Ω, the current through the DUT should be: 10 V / 10 Ω = 1 A. However, because the limit is set to 10 mA, the current will not exceed 10 mA, and the voltage across the resistance is limited to 100 mV. In effect, the 10 V voltage source is transformed into a 10 mA current source.

In steady-state conditions, the set limit restricts the instrument output unless there are fast transient load conditions.

If the source output exceeds the source limit:

- On the Home screen, LIMIT is displayed to the right of the source voltage.
- The Source value changes to yellow.
- The Limit value on all swipe screens changes to yellow.

The source is clamped at the maximum limit value. For example, if the measurement limit is set to 1 V and the measurement range is 2 V, the output voltage is clamped at 1 V.

For additional details on using limits, see Operating boundaries (on page 4-4).

Setting source limits

Using the front panel:

1. Press **FUNCTION** and select the source and measurement combination.
2. On the Home page, select the button next to Limit.
3. Set the value.
4. Select **OK**.

Using SCPI commands:

To set the limit when sourcing current, send the command:

```
SOURCe:CURRent:PROTection <n>
```

Where `<n>` is the current limit value. To set for a voltage measurement, replace `CURRent` with `VOLTage`.

Using TSP commands:

To set the limit when sourcing current, send the commands:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.vlimit.level = limitValue
```

To set the limit when sourcing voltage, send the commands:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = limitValue
```

Where *`limitValue`* is the limit value.

Ranges

You can set ranges for the source and measurement values. You can set specific ranges or allow the instrument to choose the ranges automatically.

The source range determines how accurately the source output can be set.

The measurement range determines the full-scale input for the measurement. The measurement range also affects the accuracy of the measurements and the maximum signal that can be measured.

The highest available range is determined by the limit setting for the function that is being sourced or measured.

Source range

For most applications, you will select the automatic source range. This causes the instrument to set the source range to the best range for the present settings.

In most cases, the only reason you would select a specific source range is to reduce the time that is needed to make a measurement. Selecting a specific range eliminates the time that the instrument needs to select the range automatically.

If you set the source range manually through either the front panel or a remote command, the setting for automatic source range is set to disabled.

The selected source range must be within the limit set by the overvoltage protection limit. When the overvoltage protection level is exceeded, OVP is displayed in the SOURCE area of the Home screen.

To select the range, specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 3 V, send the command set to 0.05 (or 50e-3) to select the 200 mV range.

Considerations for the 10 nA and 100 nA ranges

If you are operating the Model 2450 using the 10 nA and 100 nA ranges, you need to take extra precautions to get accurate readings.

Be aware that these ranges may be selected if you have autoranging selected.

If you are using the 10 nA or 100 nA range, use the following precautions.

Never leave the FORCE HI terminal open to the environment. Current can couple into FORCE HI and cause measurable current readings from unlikely sources. If you are operating from the rear terminals, the triaxial connections will place guard around the FORCE HI and SENSE HI terminals so that outside current is unlikely to affect the measurement.

Place the device under test inside a shielded enclosure (LO) so that the guards surrounding FORCE HI and SENSE HI are inside the enclosure as close to the device under test as possible before dropped.

If you are using the front panel connections, shield the FORCE HI terminal with LO. While this will not prevent leakage, it will prevent noise and external signals from coupling into the FORCE HI terminal.

If you are using multiple instruments in an application, if possible, connect the LO terminals of each instrument together. This prevents common-mode current from flowing through the measurement circuits of the Model 2450. While the Model 2450 isolates measurement circuitry from earth ground, it is still possible to have hundreds of nanoamps of earth-referenced (common mode) current at the LO terminal. If the Model 2450 HI terminal is connected to the LO terminal of another instrument, the common-mode current will flow into the Model 2450 measurement circuits, possibly driving the Model 2450 into source limit.

Another method for preventing common-mode current from affecting measurements is to provide an earth reference for the measurement. The earth connection will allow the common-mode current of the offending instrument to flow directly to earth, not through the measurement circuits of the Model 2450.

Selecting a specific source range

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 μA, 10 μA, 100 μA, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 20 mV, 200 mV, 2 V, 20 V, and 200 V.

Using the front panel:

1. Press the **HOME** key.
2. Under SOURCE, select the button next to Range. The Select Range dialog box is displayed.
3. Select the range. You are returned to the Home screen.

Using SCPI commands:

To set the current source range, send the following command, with `<n>` set to the source range:

```
SOURce:CURRent:RANGe <n>
```

To set the voltage source range, send the following command, with `<n>` set to the source range:

```
SOURce:VOLTage:RANGe <n>
```

Using TSP commands:

To set the current source range, send the following commands, with *rangeValue* set to the source range:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = rangeValue
```

To set the voltage source range, send the following commands, with *rangeValue* set to the source range:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = rangeValue
```

Selecting the automatic source range

When the automatic range is selected, the source-measurement cycle is repeated to determine the correct range. This means that any source delay is applied each time the instrument has to set the automatic range. For example, if you program a one-second source delay, the instrument could take two or more seconds to complete a reading if it must change ranges.

The instrument changes ranges as follows:

1. If the reading reaches 105% of the present range, the instrument goes up three ranges or to the highest range possible.
2. The instrument takes another reading.
3. The instrument uses this reading to determine whether it needs to continue going up in range or if it picks the range based on the reading. If the reading is 10%, 1%, or 0.1% of the present range, it will go down by 1, 2, or 3 ranges, respectively.

Using the front panel:

1. Press the **HOME** key.
2. Under SOURCE, select the button next to Range. The Select Range dialog box is displayed.
3. Select **AUTO**. You are returned to the Home screen.

Using SCPI commands:

To set the current source range, send the command:

```
SOURce:CURRent:RANGe:AUTO ON
```

To set the voltage source range, send the command:

```
SOURce:VOLTage:RANGe:AUTO ON
```

Using TSP commands:

To set the source range when current is sourced, send the following commands:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.autorange = smu.ON
```

To set the source range when voltage is sourced, send the following commands, with *rangeValue* set to the source range:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.autorange = smu.ON
```

Measurement range

The measurement range determines the full-scale input for the measurement. The measurement range also affects the accuracy of the measurements and the maximum signal that can be measured.

A change to the measurement range can cause a change to the related current or voltage limit. When this occurs, an event message is generated.

The total range is 22 W; no combination of voltage and current ranges can exceed 22 W. For example, with 200 V source range, the highest current measurement range is 100 mA (200 V * 100 mA = 20 W).

Whether or not you can select a measurement range is affected by other settings on the instrument. You can only select a measurement range if you are sourcing one type of measurement and measuring another. For example, you can select a measurement range if you are sourcing voltage and measuring current. However, if you are sourcing voltage and measuring voltage, the measurement range is the same as the source range and cannot be changed.

You can only select a measurement range for ohms if you are using the instrument as an ohmmeter (sourcing current and measuring resistance).

Maximum limits for Model 2450

Measure range	Maximum source limit
20 mV	±21 mV
200 mV	±210 mV
2 V	±212 mV
20 V	±2.1 V
200 V	±21 V
10 nA	±10.5 nA
100 nA	±10.6 nA
1 µA	±1.05 µA
10 µA	±1.06 µA
100 µA	±10.6 µA
1 mA	±106 µA
10 mA	±1.06 mA
100 mA	±10.6 mA
1 A	±106 mA

Selecting a specific measurement range

NOTE

You need to set the measurement function before you can set the measurement range.

When selecting a measurement range, to ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow error.

NOTE

If you set the measurement range to a specific value, the setting for automatic measurement range is set to disabled.

Using the front panel:

1. Press **FUNCTION** and select the measurement function.
2. On the Home page, select the button next to Range for the measurement type. The Select Range dialog box is displayed.
3. Select the range. The Home page is displayed again with the selected value.

If the instrument displays an overflow message, select a higher range.

Using SCPI commands

For a current measurement, send the command:

```
SENSe:CURRent:RANGe:UPPer <n>
```

Where `<n>` is the positive full-scale value of the measurement range.

To set for a voltage measurement, replace `CURRent` with `VOLTage`.

To set for a resistance measurement, replace `CURRent` with `RESistance`.

Using TSP commands

For the current range, send the commands:

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.OFF
smu.measure.range = rangeValue
```

Where *`rangeValue`* is the positive full-scale value of the measurement range.

For a voltage measurement, replace `smu.FUNC_DC_CURRENT` with `smu.FUNC_DC_VOLTAGE`.

For a resistance measurement, replace `smu.FUNC_DC_CURRENT` with `smu.FUNC_RESISTANCE`.

Selecting the automatic measurement range

When automatic measurement range is selected, the instrument automatically selects the best range to measure the signal. If the measurement reaches 105% of the present range, the instrument changes the measurement range to the next higher range.

If you enable the automatic measurement range, the measurement range is changed when a measurement is made. To read the measurement that the instrument chose, you must query the range after a measurement is made.

If you set low or high limits for the measurement when the automatic range is disabled, the limits are not used until the limit is changed. The instrument only checks the upper limit when the limit is increased. If the instrument is on a higher range than the present upper limit, and the instrument automatically goes down a measurement range, it can still be on a range that is higher than the upper limit. The opposite is true for the lower limit.

The instrument does not evaluate the limits until the instrument automatically changes the range. This means that if the instrument is on a range that is above or below the upper or lower limits when the automatic range is enabled or when the limit is set, no range change will occur until the range needs to be changed.

NOTE

You need to set the measurement function before the measurement range can be set.

If you set the measurement range manually for a function, automatic measurement range is automatically turned off for that function and remains off until you re-enable it.

Choose automatic measurement range using the front panel

Using the front panel:

1. Press **FUNCTION** and select the source and measurement function.
2. On the Home page, select the button next to Range. The Select Range dialog box is displayed.
3. Select **AUTO**. The Home page is displayed again with the selected value shown as AUTO.

Using SCPI commands

For the current range, send the command:

```
SENSe:CURRent:RANGe:AUTO ON
```

To set for a voltage measurement, replace `CURRent` with `VOLTage`.

To set for a resistance measurement, replace `CURRent` with `RESistance`.

Using TSP commands

To set automatic range for the current measurement function, send the commands:

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.ON
```

To set automatic range for the voltage measurement function, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
```

To set automatic range for the resistance measurement function, send the commands:

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.autorange = smu.AUTORANGE_ON
```

Selecting low limits when automatic measurement range is used

You can set the low limit for the measurement range that is selected when the measurement range is set automatically.

Choose the lower limits for the automatic measurement range using the front panel

Using the front panel:

1. Press **FUNCTION** and select the measurement function.
2. Press **MENU**.
3. Under Measure, select **Settings**. The MEASUREMENT SETTINGS page is displayed.
4. Select the button next to Auto Range Low Limit.
5. Select the low limit. The MEASUREMENT SETTINGS page is displayed again.
6. Press **HOME** to return to the Home page.

Using SCPI commands:

To set the lower limit for current measurements, send the command:

```
SENSe:CURRent:RANGe:AUTO:LLIMit <n>
```

Where `<n>` is the lowest current measurement range that can be used.

To set the lower limit for voltage measurements, send the command:

```
SENSe:VOLTage:RANGe:AUTO:LLIMit <n>
```

Where `<n>` is the lowest voltage measurement range that can be used.

To set the lower limit for resistance measurements, send the command:

```
SENSe:RESistance:RANGe:AUTO:LLIMit <n>
```

Where `<n>` is the lowest resistance measurement range that can be used.

Set the lower limits for the automatic measurement range using the TSP commands

Using TSP commands:

Send the commands:

```
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.autorange = smu.ON
smu.measure.lowrange = lowRange
```

Where *`lowRange`* is the lowest current measurement range that can be used.

To set the lower limits for voltage, replace `smu.FUNC_DC_CURRENT` with `smu.FUNC_DC_VOLTAGE`.

To set the lower limits for resistance, replace `smu.FUNC_DC_CURRENT` with `smu.FUNC_RESISTANCE`.

Determining upper limits when automatic measurement range is used

For resistance measurements, you can define the upper limit that can be selected when the measurement range is set automatically. For voltage measurements, you can retrieve the value of the upper limit.

These options are only available for voltage and resistance measurements.

Using SCPI:

To query the upper limit for voltage measurements, send the command:

```
:SENSe:VOLTage:RANGe:AUTO:ULIMit?
```

To set the upper limit for resistance measurements, send the command:

```
:SENSe:RESistance:RANGe:AUTO:ULIMit <n>
```

Where `<n>` is the highest resistance measurement range that can be used.

Using TSP commands:

To view the upper limit for voltage measurements, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
print(smu.measure.highrange)
```

Where *highRange* is the highest voltage measurement range that can be used.

To set the upper limit for resistance measurements, send the commands:

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.autorange = smu.ON
smu.measure.highrange = highRange
```

Where *highRange* is the highest resistance measurement range that can be used.

Automatic reference measurements

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The Model 2450 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks the 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 readings, one for the internal ground and one for the voltage reference, before returning the result. This can cause some measurements to take longer 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, you can disable the automatic reference measurements.

When automatic reference measurements are turned off, the instrument may gradually drift out of specification. To prevent inaccurate readings, you can send a command that gets autozero information on a one-time basis.

Setting autozero

You can enable or disable automatic referencing, or request a one-time refresh of the reference values.

The reference setting is stored with the measure function.

To set autozero using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measurement combination.
3. Press the **MENU** key.
4. Under Measure, select Settings.
5. Next to Auto Zero, select **On** or **Off**.
6. If Off is selected, you can select the **Once Now** option to send a one-time refresh. When you select Once Now, Auto Zero is enabled after the refresh.
7. Select **HOME** to return to the operating display.

To set autozero using SCPI commands:

To turn autozero on, send the command:

```
:SENSe:VOLTage:AZERo ON
```

To turn autozero off, send the command:

```
:SENSe:VOLTage:AZERo OFF
```

To set autozero on or off for current measurements, replace `VOLTage` with `CURRent`. To set it for resistance measurements, replace it with `RESistance`.

To perform a one-time autozero update, send the command:

```
:SENSe:AZERo:ONCE
```

To set autozero using TSP commands:

To turn autozero on, select the measurement type, then send the command:

```
smu.measure.autozero.enable = smu.ON
```

To turn autozero off, select the measurement type, then send the command:

```
smu.measure.autozero.enable = smu.OFF
```

To perform a one-time autozero update, select the measurement type, and then send the command:

```
smu.measure.autozero.once()
```

Source readback

You can set the instrument to record and display the voltage or current of the configured source value or the actual source value. When you use the configured source value, the instrument records and displays the value that was configured. When you use the actual source value, the instrument measures the actual source value immediately before making the measurement of the device under test.

Using source readback results in more accurate measurements, at the cost of a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the device under test measurement. For example, if you have the source set to 60 V, you will see something like `+059.998 V` in the SOURCE VOLTAGE area of the Home screen.

When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the measurement of the device under test. For example, if the source is set to 60 V, you will see something like `+060.000 V` in the SOURCE VOLTAGE area of the Home screen.

Setting source readback

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Next to Source Readback, select **On** or **Off**.
4. Select **HOME** to return to the operating display.

Using SCPI commands:

To set readback off for the current source, send the command:

```
:SOURce:CURRent:READ:BACK OFF
```

To set readback for the voltage source, replace `CURRent` with `VOLTage`. To turn readback off, replace `OFF` with `ON`.

Using TSP commands:

To set readback off for the current source, send the commands:

```
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.readback = smu.OFF
```

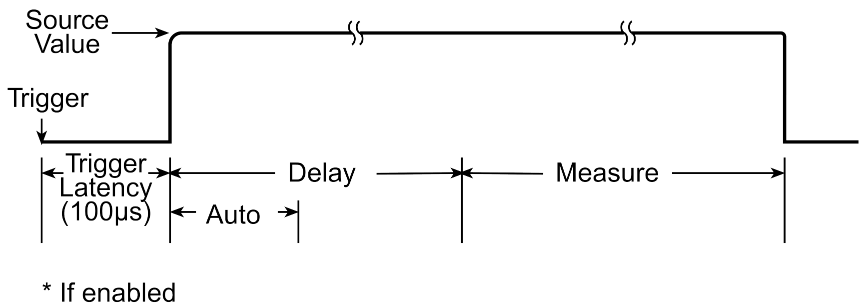
To set readback for the voltage source, replace `smu.FUNC_DC_CURRENT` with `smu.FUNC_DC_VOLTAGE`. To turn readback on, replace `smu.OFF` with `smu.ON`.

Source delay

When you use the instrument to source and measure, there is a delay between when the source is turned on and when the measurement is made. The delay provides a settling time for the source.

The amount of delay time depends on the settings that are made for the source delay. You can set a manual value, or use the auto delay.

Figure 58: Model 2450 settling and delay times



You can increase the amount of delay time by either setting a fixed amount of time or setting the instrument to include an automatic delay. The delay can be from 0 to 4 seconds.

If you select an automatic source delay, the delay time depends on the selected range. Values for the delay times for each range are shown in the following table.

Current range	Voltage source autodelay (ms)	With high capacitance (ms)	Current source autodelay (ms)	With high capacitance (ms)
10 nA	50	100	50	100
100 nA	50	100	50	100
1 µA	3	20	3	20
10 µA	2	10	2	10
100 µA	1	10	1	10
1 mA	1	10	1	10
10 mA	1	5	1	5
100 mA	1	5	1	5
1 A	1	5	2	5

For high impedance and high capacitive loads, more settling time is required for the source. The actual delay period you need can be calculated or determined by trial and error. For purely resistive loads and at higher current levels, the programmable delay can be set to 0 ms.

The measure time depends on the selected measurement speed. For example, if the speed is set at 0.01 PLC, the measure time would be 167 ms for 60 Hz operation (0.01/60).

Setting the source delay

Using the front panel:

1. Press **FUNCTION** and set the source and measurement function.
2. Press the **MENU** key.
3. Under Source, select **Settings**.
4. Select the button next to Source Delay and enter a value.
5. Click **OK**.

Using SCPI commands:

To set the source delay, set the command to a value. For example, to set the delay to 2 seconds, send:

```
SOURce:CURRent:DELay 2
```

To set an automatic source delay, send the command:

```
:SOURce:CURRent:DELay:AUTO ON
```

To set these commands for a voltage source, replace `CURRent` with `VOLTage`.

Using TSP commands:

To set a specific value, set the source function and send the command `smu.source.delay = sDelay` where *`sDelay`* is the delay in seconds. For example, to set a 0.5 s delay, send the command:

```
smu.source.delay = .5
```

To set an automatic source delay, send the command:

```
smu.source.autodelay = smu.ON
```

To turn off the automatic source delay, send the command:

```
smu.source.autodelay = smu.OFF
```

Saving setups

You can save the present settings and any source or measure configuration lists that you have defined for the Model 2450 to internal memory or an external USB flash drive.

After the settings are saved, you can recall the settings. You can also set them to be the default settings on power up.

If you are using TSP commands, saved setups are scripts and can be added, modified, and deleted like any other script. See Introduction to TSP operation (on page 7-1) for additional information on working with scripts.

Save a user setup to internal memory

From the front panel:

1. Configure the Model 2450 to the settings that you want to save.
2. Press the **MENU** key.
3. Under Scripts, select **Create Config**. The CREATE CONFIG SCRIPTS window is displayed.
4. Select **Create**. A keyboard is displayed.
5. Use the keyboard to enter the name of the script.
6. Select **ENTER**. The script is added to internal memory.

Using SCPI commands:

Configure the instrument to the settings that you want to save. To save the setup, send the command:

```
*SAV <n>
```

Where `<n>` is an integer between 0 and 4.

NOTE

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

Using TSP commands:

Configure the instrument to the settings that you want to save. To save the setup, send the command:

```
createconfigscript("setupName")
```

Where *`setupName`* is the name of the setup script that will be created.

Save a user setup to a USB flash drive

From the front panel:

1. Save the user setup to internal memory, as described in Save a user setup to internal memory (on page 2-113).
2. Insert the USB flash drive into the USB connector on the front panel.
3. Press the **MENU** key.
4. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
5. In the Internal Scripts list, select the script you want to copy to the USB flash drive.
6. Select **>**.

Using TSP commands:

1. Save the user setup to internal memory, as described in Save a user setup to internal memory (on page 2-113).
2. Insert the USB flash drive into the USB connector on the front panel.
3. Send the command:

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

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

Copy a user setup

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

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

Delete a user setup

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

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

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

Overwrite an existing setup with the new setup. See Save a user setup to internal memory (on page 2-113).

To delete a user setup to internal memory using TSP commands:

To delete the setup, send the command:

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

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

Recall a user setup

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

To recall a saved setup from the front panel:

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

To recall a user setup to internal memory using SCPI commands:

Send the command:

```
*RCL <n>
```

Where `<n>` is an integer between 0 and 4 that represents the saved script.

To recall a saved setup using TSP commands:

Send the command:

```
setupName()
```

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

Define the setup used for power-on

You can select a configuration to be used at power up.

From the front panel:

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

Set the power-on configuration using a SCPI command:

Send the command:

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

Where `<name>` is:

- `RST`: Power-up to *RST defaults
- `SAV0`: Power-up to setup stored at memory location 0
- `SAV1`: Power-up to setup stored at memory location 1
- `SAV2`: Power-up to setup stored at memory location 2
- `SAV3`: Power-up to setup stored at memory location 3
- `SAV4`: Power-up to setup stored at memory location 4

Using a TSP command:

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

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

Reset the instrument

You can reset many of the commands to their default values. For detail on what gets reset, see Reset default values (on page 4-24). Default values are also listed in the individual command descriptions.

If you are connected to a TSP-Link system, resetting the instrument resets all TSP-Link enabled instruments on the TSP-Link system.

Using the front panel:

1. Press the **MENU** key.
2. Under System, select **Manage**.
3. Select **System Reset**.
4. The commands are reset and a confirmation message is displayed.

Using TSP commands:

Send the command:

```
reset()
```

NOTE

If the instrument is connected to a TSP-Link system and you are using TSP commands, you can reset only the local instrument by sending `localnode.reset()` instead of `reset()`.

Using SCPI commands:

Send the command:

```
*RST
```

Using the event log

The event log records events that are reported by the instrument. The event log entries can be one of the following types:

- Error: An error occurred. This may indicate that a command was sent incorrectly.
- Warning: This message indicates that a change occurred that could affect operation.
- Information: The message is for information only. This is used to indicate status changes or information that may be helpful to the user. It also includes commands if the Log Command option is on.

The event log can hold up to 1000 events. When more than 1000 events are in the event log, the oldest event is removed when a new event is received.

Information provided for each event log entry

Each event log entry includes the following information:

- The type of event (informational, error, or warning)
- The time when the event occurred; this includes date for the first entry after power up
- The code number of the event; this number can be used with the status model to map events to bits in the event registers
- The description of the event

On the front panel, when you select an event from the System Events tab, a dialog box is displayed that shows you additional information about the event.

To access event log listing from the front panel:

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

Event log settings

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

To access event log settings from the front panel:

1. Press the **MENU** key.
2. Under System, select **Event Log**.
3. Select the **Settings** tab. A list of settings appears.
4. Press a box to select or clear a setting.

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

Settings tab settings	Description
Show Warning	Turns the logging of warnings on or off. If you turn this off, the instrument continues to record warning and display messages for them, but does not display them in the System Events tab.
Show Information	Turns the logging of information messages on or off. If you turn this off, the instrument continues to record information messages and display messages for them, but does not display them in the System Events tab.
Log Warning	Turns the logging of warnings on or off. If this is turned off, the instrument will not log or display messages for warnings.
Log Information	Turns the logging of information messages on or off. If this is turned off, the instrument will not log or display messages for information messages.
Log Command	Turns the logging of commands on or off. When this is turned on, the instrument records the commands that are sent to the instrument. It records commands sent from any interface (the front panel or remote interface).
Save to USB	Saves the event log to a .csv file on the flash drive. The filename is `eventlog.csv`.
Clear Log	Clear all entries from the event log.

Effects of errors on scripts

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

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

Section 3

Functions and features

In this section:

Relative offset

When making measurements, you may want to offset a known value that affects the measurement value. This is typically used to offset current leaks.

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 measurement function, the value is the same for all ranges for that measurement function. For example, if 5 V is set as the relative offset value on the 20 V range, the relative offset value is also 5 V on the 2 V and 200 mV ranges.

On the front panel, when relative offset is enabled, the REL indicator to the right of the measured value is displayed.

A relative offset value is saved for each function. If you change the measurement function, the relative offset value is changed to the setting for that measurement function.

The relative offset is applied to the measurement after any math functions but before the limit test functions. For more information on the order in which operations are performed, see Displayed measurements (on page 3-9).

Establishing a relative offset value

You can use the Model 2450 to automatically determine the relative offset, or you can assign a specific relative offset value.

Automatically acquiring a relative offset value

When you automatically acquire a relative offset value, the Model 2450 does the following actions:

- Makes a new measurement.
- Stores the measurement as the new relative offset level.

Using the front panel:

1. Press the **FUNCTION** key and select the measurement function. The relative offset will be applied to this function.
2. On the SETTINGS swipe screen, select the box next to **Rel**.

When the relative offset is selected, the REL annunciator to the right of the measurement is displayed.

Quick Tip

You can also enable the relative offset feature by selecting MENU > Filter/Math > Rel State.

Using SCPI commands:

Send the commands:

```
:FUNC "VOLT"
:SENSe:VOLTage:RELative:ACQuire
:SENSe:VOLT:REL:STATe ON
```

To acquire a relative offset value for another function, replace `VOLTage` with `CURRent` or `RESistance`.

Using TSP commands:

Send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.rel.acquire()
smu.measure.rel.enable = smu.ON
```

To set the relative offset for another function, replace `smu.FUNC_DC_VOLTAGE` with `smu.FUNC_DC_CURRENT` or `smu.FUNC_RESISTANCE`.

Setting a relative offset value

You can set a specific relative offset value using the remote commands. This option is not available through the front panel.

Using SCPI commands:

Send the commands:

```
:SENSe:FUNCtion "VOLTage"
:SENSe:VOLTage:RELative <n>
:SENSe:VOLTage:STATe ON
```

Where `<n>` is the amount of the offset.

To set the relative offset for another function, replace `VOLTage` with `CURRent` or `RESistance`.

Using TSP commands:

Send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.rel.level = relValue
smu.measure.rel.enable = smu.ON
```

Where *`relValue`* is the relative offset value.

To set the relative offset for another function, replace `smu.FUNC_DC_VOLTAGE` with `smu.FUNC_DC_CURRENT` or `smu.FUNC_RESISTANCE`.

Disabling the relative offset

Using the front panel:

1. Select the measurement function to which the relative offset is applied.
2. On the SETTINGS swipe screen, select the box next to **Rel**. An X should be displayed and the REL annunciator to the right of the measurement is no longer displayed.

Quick Tip

The relative offset feature can also be disabled by selecting MENU > Filter/Math > Rel State and selecting Off.

Using SCPI commands:

Send the command:

```
:SENSe:VOLTage:RELative OFF
```

To set the relative offset for another function, replace `VOLTage` with `CURRent` or `RESistance`.

Using TSP commands:

Send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.rel.enable = smu.OFF
```

To set the relative offset for another function, replace `smu.FUNC_DC_VOLTAGE` with `smu.FUNC_DC_CURRENT` or `smu.FUNC_RESISTANCE`.

Instrument access

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

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

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

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

When you set access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. To use another interface, log out of the present interface before logging into the new interface. You do not need a password with this access.

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

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

Changing the instrument access mode

To change the access mode from the front panel:

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

Using SCPI commands

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

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

Using TSP commands

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

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

Changing the password

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

The default password is `admin`.

To change the password from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Password. A keypad opens.
4. Enter the new password.
5. Select the **ENTER** button on the keypad. The password is reset.

NOTE

You can reset the password by pressing the **MENU** key, selecting **Manage** (under System), and then selecting the **LAN and Password Reset** button. Doing this returns the LAN settings (including the password) to the default settings.

To change the password using SCPI commands:

```
:SYSTem:PASSword:NEW "password"
```

Where `password` is the new password.

To change the password using TSP commands:

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

Where *`password`* is the new password.

Switching control interfaces

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

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

When you change interfaces from a remote interface, you must send the following command before sending commands:

```
login password
```

Where *`password`* is the password.

Calculations that you can apply to measurements

The Model 2450 has three built-in math calculations:

- mx+b
- percent
- reciprocal (1/X)

Math calculations are applied to the input signal after relative offset and before limit tests. For more detail on the order of operations, see Displayed measurements (on page 3-9).

Math operations apply to the selected measurement function. If you change the measurement function, the math operation for that measurement function becomes active.

mx+b

The mx+b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$mx + b = Y$

Where:

- **m** is a user-defined constant for the scale factor
- **x** is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- **b** is the user-defined constant for the offset factor
- **Y** is the displayed result

When the mx+b math operation is active, the unit of measure for the front-panel voltage and current readings is **X** and the MATH indicator is displayed to the right of the measurement. For resistance readings, the units of measure do not change. You cannot change this units designator.

Percent

The percent math function displays measurements as percent deviation from a specified constant. The percent calculation is:

$$Percent = \left(\frac{input - reference}{reference}\right) \times 100\%$$

Where:

- **Percent:** The result
- **Input:** The measurement (if relative offset is being used, this is the relative offset value)
- **Reference:** The user-specified constant

The result of the percent calculation is positive when the input is more than the reference. The result is negative when the input is less than the reference.

When the percent operation is active, the unit of measure for the front-panel voltage and current readings is **%** and the MATH indicator is displayed to the right of the measurement. For resistance readings, the units of measure do not change. You cannot change this unit's designator.

Reciprocal (1/X)

You can set math operation to reciprocal to display the reciprocal of a reading.

The reciprocal is 1/X, where X is the reading. If relative offset is on, the 1/X calculation uses the input signal with the relative offset applied.

The result of the 1/X calculation may be displayed in exponential notation. For example, a displayed reading of +2.500E+03 is equivalent to 2500.

Example:

Assume the normal displayed reading is 002.5000 Ω. The reciprocal of resistance is conductance. When the reciprocal math function is enabled, the following conductance reading is displayed:

```
0.400000
```

When the reciprocal math operation is active, the unit of measure for the front-panel voltage and current readings is **R** and the MATH indicator is displayed to the right of the measurement. For resistance readings, the units of measure do not change. You cannot change this units designator.

Setting percent math operations

From the front panel:

1. Press **FUNCTION** and select the measurement function.
2. Press the **MENU** key.
3. Under Measure, select **Filter/Math**.
4. Next to Math State, select **On**.
5. Next to Math Function, select **Percent**.
6. Next to Zero Percent Reference, enter the percentage.

Using SCPI commands:

To set the math operations to percent, send the commands:

```
:CALCulate:VOLTage:MATH:FORMat PERCent
:CALCulate:VOLTage:MATH:PERCent <n>
```

where `<n>` is the constant.

To set the percent math operation for another measurement function, replace `VOLTage` with `CURRent` or `RESistance`.

Using TSP commands:

Set the measurement function, and then send the commands:

```
smu.math.format = smu.MATH_PERCENT
smu.math.percent = value
smu.math.enable = smu.ON
```

where *`value`* is the constant

Setting mx+b math operations

From the front panel:

1. Press **FUNCTION** and select the measurement function.
2. Press the **MENU** key.
3. Under Measure, select **Filter/Math**.
4. Next to Math State, select **On**.
5. Next to Math Function, select **mx+b**.
6. Next to m (Scalar), select the value for m.
7. Next to b (Offset), select the value for b.

Using SCPI commands:

To set the math operations to mx+b, send the commands:

```
:CALCulate:CURRent:MATH:FORMat MXB
:CALCulate:CURRent:MATH:MMF <m>
:CALCulate:CURRent:MATH:MBF <b>
:CALCulate:CURRent:MATH:STATE ON
```

where `<m>` is the m factor and `<b>` is the b factor.

To set the math operations for a different function, change `CURRent` to `VOLTage` for voltage measurements or `RESistance` for resistance measurements.

Using TSP commands:

Set the measurement function, and then send the commands:

```
smu.measure.math.format = smu.MATH_MXB
smu.measure.math.mxb.mfactor = mvalue
smu.measure.math.mxb.bfactor = bvalue
smu.measure.math.enable = smu.ON
```

where *`mvalue`* is the m factor and *`bvalue`* is the b factor.

Setting reciprocal math operations

From the front panel:

1. Select the measurement function.
2. Press the **MENU** key.
3. Under Measure, select **Filter/Math**.
4. Next to Math State, select **On**.
5. Next to Math Function, select **Reciprocal**.

Using SCPI commands:

To set the math operations to reciprocal, send the command:

```
:CALCulate:VOLTage:MATH:FORMat RECiprocal
```

To set the percent math operation for another measurement function, replace `VOLTage` with `CURRent` or `RESistance`.

Using TSP commands:

Set the measurement function, and then send the commands:

```
smu.math.format = smu.MATH_RECIPROCAL
smu.math.enable = smu.ON
```

Switching math on the SETTINGS screen

Once you set the math operations settings for a measurement function, you can turn the math function on or off on the SETTINGS swipe screen.

From the front panel:

1. Select **HOME**.
2. Go the SETTINGS swipe screen.
3. Select the button next to **Math** to change the setting.

Displayed measurements

When you make measurements, the instrument may perform operations on the measured values that will affect what you see on the display and the measurements that are stored in the buffer.

The operations that can affect the measurement display are:

- Relative offset
- Math operations
- Limit tests

If none of these operations are set, the value that is displayed on the front panel is the actual measurement reading.

If any of these operations are set, the value that is displayed is the measurement reading with these operations applied. The operations are applied in the order shown above.

For example, if you made a measurement and had a relative offset and limit tests active, the measured value would have the relative offset applied, then have limit test results applied.

Reading buffers

Reading buffers capture measurements, ranges, instrument status, and the output state of the instrument. The Model 2450 has two default reading buffers. You can also create user-defined reading buffers.

Reading buffers provide the following statistics: average, minimum, maximum, and standard deviation. If you use SCPI commands over the remote interface, peak-to-peak statistics are also available.

You can perform the following operations on reading buffers from the front panel or the remote interface:

- Configure, store, and recall reading buffers. Only one reading buffer is active when you control buffers from the front panel.
- View reading buffer content.
- Choose to store readings in the default reading buffers or the user-defined reading buffers.
- Save reading buffer content to a USB flash drive.
- Set reading buffers to fill once or fill continuously.
- Change the capacity of reading buffers.
- Delete user-defined reading buffers. You cannot delete `defbuffer1` and `defbuffer2`.
- Clear reading buffers.
- Clear the default reading buffers and delete the user-defined reading buffers by turning the instrument off or sending an instrument reset command.

Getting started with buffers

The following sections provide you with information to help you start using reading buffers. The Remote buffer operation (on page 3-27) section provides additional information about accessing the reading buffers with remote commands.

Using default buffers

There are two default buffers:

- `defbuffer1`
- `defbuffer2`

If you do not select a specific buffer, all readings are stored in `defbuffer1`. If you want to store readings in `defbuffer2`, you need to select it. If you want to store readings in a user-defined buffer, you need to create the buffer and then select it.

Parameters in buffer-specific commands that require buffer names default to `defbuffer1` with the following exceptions:

- Print commands
- SCPI command `TRACE:MAKE`
- TSP command `buffer.make`

For information about default values, see Reset default values (on page 4-24).

Effects of reset and power cycle on buffers

The instrument clears the default buffers when a reset command is sent or when the power is turned off and then turned on again.

The instrument deletes all user-defined buffers when a reset command is sent or when the power is turned off and then turned on again.

Buffer fill status

There are several different ways to determine buffer fill status from the front panel.

As shown in the following figure, the Active buffer indicator (on page 2-14) displays buffer fill status and the STATISTICS swipe screen (on page 2-12) displays buffer statistics.

Figure 59: BUFFER STATISTICS swipe screen and active buffer indicator

You will receive the event code 4915, "Attempting to store past capacity of reading buffer," in a popup message when a buffer that is set to fill once is full.

NOTE

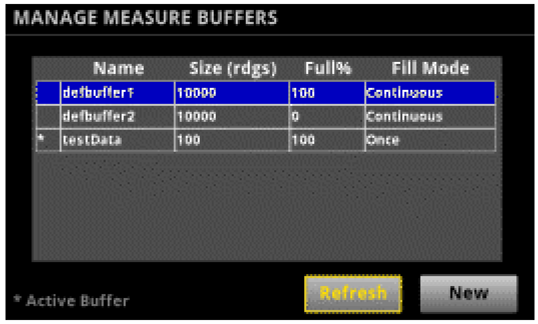
You can turn the display of messages on or off. Refer to System Event Log menu (on page 2-34) for information about turning the messages on or off.

The MANAGE MEASURE BUFFERS window displays buffer fill status. For example, the status in the following figure indicates that:

- `defbuffer1` is completely filled and is overwriting readings in the buffer as additional readings are made
- `defbuffer2` is empty
- `testData` is 100% full and no more readings are being stored in `testData`.

Select the **Refresh** button to update buffer fill status.

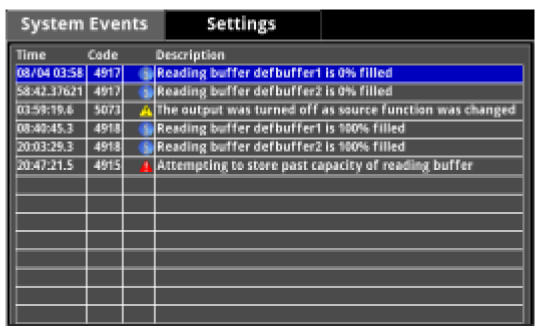
Figure 60: MANAGE MEASURE BUFFERS



The System Events tab on the System Event Log menu (on page 2-34) displays the following buffer events:

- Event code 4917, "The fill status of *bufferVar* is 0% filled"
- Event code 4918, "Reading buffer *bufferVar* is 100% filled"
- Event code 4915, "Attempting to store past the capacity of reading buffer," which occurs when a buffer that is set to fill once fills.

Figure 61: System Events tab



Creating buffers

To create a new user-defined reading buffer, you need to provide a name and capacity for the new buffer.

User-defined buffer names must start with an alphabetic character. The names cannot contain any periods or the underscore (_) character.

There is no fixed limit on the number of user-defined reading buffers you can create. However, you are limited by available memory in the instrument and the overall total capacity of all the buffers stored in the instrument cannot exceed 1,000,000 readings.

The following topics provide information about using the front panel to create buffers and introduce how to use remote commands to create buffers.

NOTE

The instrument automatically selects the reading buffer you just created as the buffer in which to store front-panel readings.

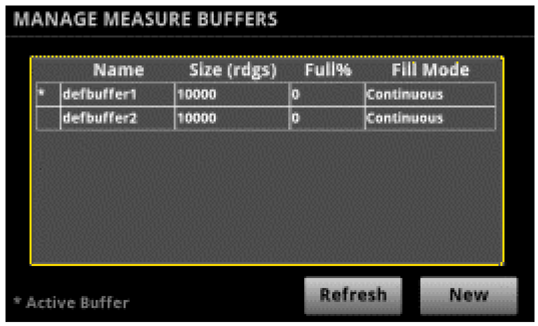
For additional information about using remote commands for buffer operations, see the following sections of this manual:

- Remote buffer operation (on page 3-27)
- SCPI commands see TRACe subsystem (on page 6-107)
- TSP command see TSP commands (on page 8-7)

Using the front panel to create a user-defined reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.

Figure 62: MANAGE MEASURE BUFFERS window

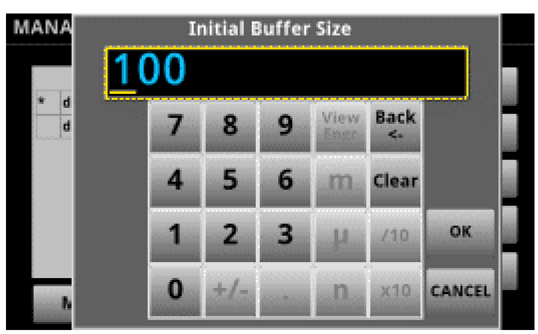


3. Select **New**. The keypad is displayed.
4. Enter a name for the buffer you are creating, for example, `testData`.

Figure 63: New Buffer Name

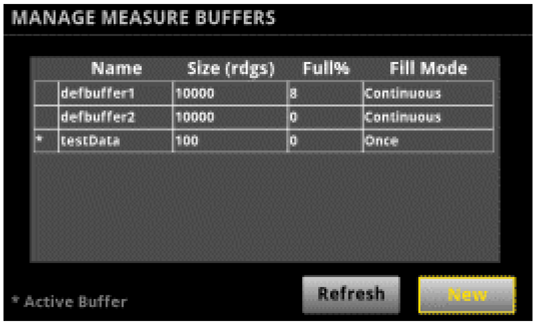
5. Select **Enter**. The Initial Buffer Size window is displayed, showing the default initial buffer size.

Figure 64: Initial Buffer Size window



6. Type the size of the new buffer.
7. Select **OK** to enter the size. The MANAGE MEASURE BUFFERS window is displayed showing the buffer you just created.

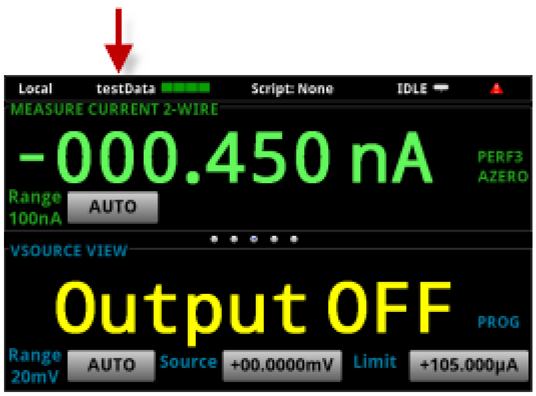
Figure 65: MANAGE MEASURE BUFFERS window



8. Press the **HOME** key to return to the **HOME** screen.

After you create a new reading buffer, the buffer becomes the active buffer. The active buffer indicator on the **HOME** screen displays the name of the active buffer. For example, the active buffer indicator shown in the following figure displays `testData`.

Figure 66: Active buffer indicator



Using SCPI commands to create a reading buffer:

To create a reading buffer named `testData` with a capacity of 200 readings, send the following command:

```
TRACe:MAKE "testData", 200
```

Using TSP commands to create a reading buffer:

To create a reading buffer named `testData` with a capacity of 200 readings, send the following command:

```
testData = buffer.make(200)
```

Selecting a buffer

The default reading buffer is `defbuffer1`. If you want to use a different buffer (`defbuffer2` or a user-defined reading buffer), use the information in this topic.

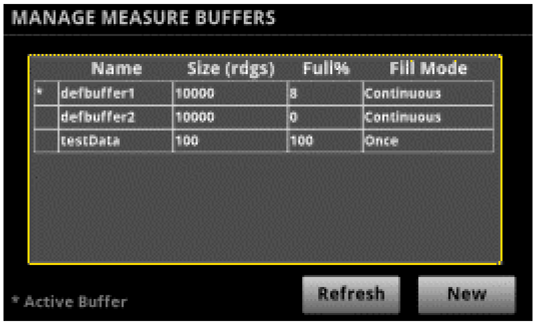
When you use remote commands to create buffers, the buffers are available to the system and can be used with any command that takes a buffer parameter. A newly created buffer automatically becomes the active buffer. If the active buffer is deleted, `defbuffer1` becomes the active buffer.

The following topics provide information about using the front panel to select buffers and provide an introduction to using remote commands to select buffers.

Using the front panel to select a reading buffer:

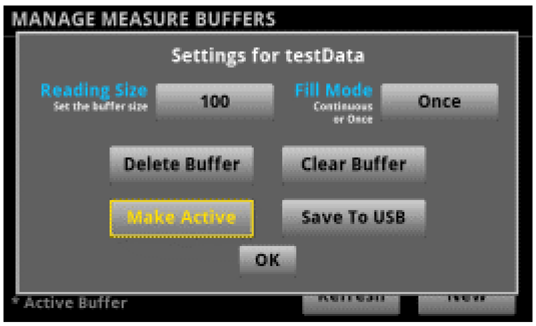
1. Press the **MENU** key.
2. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.

Figure 67: MANAGE MEASURE BUFFERS window



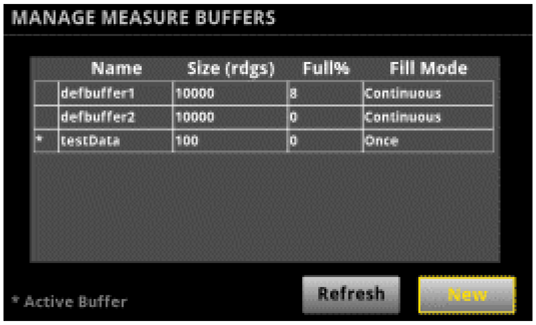
3. Select a reading buffer from the list. For example, select `testData`. The Settings for testData menu is displayed.

Figure 68: Settings for testData menu



4. Select the **Make Active** button. The "Are you sure" dialog box is displayed.
5. Select **Yes**. A list of available buffers is displayed. An asterisk in the first column of the buffer list indicates which buffer is the active reading buffer. For example, `testData` is the active buffer in the following figure.

Figure 69: MANAGE MEASURE BUFFERS window



After you select a reading buffer, the buffer becomes the active buffer. The active buffer indicator on the **HOME** screen displays the name of the active buffer.

You can also select reading buffers from the active buffer indicator on the HOME screen. Refer to Active buffer indicator (on page 2-14) for information about using the indicator to select buffers.

Using SCPI commands to select a reading buffer:

To make a measurement and store the readings in a specific reading buffer, send the `:READ? <bufferName>` command. If you do not specify a buffer when you send the `READ?` command, readings are stored in `defbuffer1`.

An alternative to sending the `:READ? <bufferName>` command is to send the `TRACe:TRIGger <buffer name>` command.

The `TRACe:TRIGger <buffer name>` command stores readings in the specified reading buffer. If no buffer is specified for the parameter, `defbuffer1` is used. To see the readings stored in the buffer after using this command, use the `FETCh?` command to see the last reading stored in the buffer or `TRACe:DATA?` command to see multiple readings from the buffer.

NOTE

When you specify a user-defined reading buffer, you must create the buffer before you can specify it.

To select current as the measurement function, measure current, and return the readings in the `testData` reading buffer, send the following commands:

```
SENSe:FUNCtion "CURRent"
READ? "testData"
```

To measure current and store the readings in the `defbuffer2` reading buffer, send the following command:

```
MEASure:CURRent? "defbuffer2"
```

To measure voltage and store the readings in the `defbuffer2` reading buffer, send the following command:

```
MEASure:VOLTage? "defbuffer2"
```

To measure current and return relative time and readings formatted as they appear on the front panel, send the following command:

```
MEASure:CURRent? "testData", READ, REL, SOURFORM
```

Buffer storage is consistent whenever readings are taken. The parameters, such as `REL` and `SOURFORM`, only affect what is included in the response. If you do not include parameters, the command only returns the reading.

Using TSP commands to select a reading buffer:

To make a measurement and store the readings in a specific reading buffer, use the `smu.measure.read()` function. If you do not specify a buffer when you use the `smu.measure.read()` function, readings are stored in `defbuffer1`.

If you select another buffer and you want to store additional readings in the previously selected buffer, you have to select the buffer again or the instrument will use the buffer you just selected.

To measure voltage and store the readings in the `voltMeasBuffer`, send the following commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.read(voltMeasBuffer)
```

To measure voltage, store the readings in the `voltMeasBuffer`, and print the last reading in the buffer, send the following command:

```
print(smu.measure.read(voltMeasBuffer))
```

To measure current, store the readings in `defbuffer1`, and print the last reading in the buffer, send the following commands:

```
smu.measure.func = smu.FUNC_DC_CURRENT
print(smu.measure.read())
```

Storing readings in buffers

Before you store readings, make sure the correct reading buffer is selected. See Selecting a buffer (on page 3-15) for more information.

To store a reading from the front panel, perform any operation that instructs the instrument to make a measurement. The buffer-fill-status indicators light up to indicate that the buffer is filling. Depending on the size of the buffer, the lit indicator may be difficult to observe. When all four indicators are lit, the buffer is completely filled. All of the indicators will never be lit as long as the number of readings stored is less than the selected buffer capacity.

To stop storing readings in a buffer when you are taking continuous readings, select the Trigger mode indicator and select the **Manual Trigger Mode**.

You can press and hold the **Trigger** key for about 3 seconds to display the trigger mode window.

NOTE

Stored readings are lost when the instrument is turned off. Stored readings are also lost when you resize a reading buffer.

Viewing and saving buffer content

You can view the content of buffers from the front panel.

However, the front panel may not be flexible enough for your particular type of data analysis. For further analysis, save the contents of the reading buffer to a USB flash drive. The stored file can be loaded directly into Microsoft® Excel or another tool. The file contains all the information the instrument records about each data point in the reading buffer. When you save the buffer data, you may indicate a starting or ending point to save only a portion of the data. If you do not specify a starting and ending point, the entire buffer data is saved. You may also specify the how you want the time saved with the time format parameter.

You can append the contents of a reading buffer to a file that is already on the USB flash drive. When you append data, you can specify the starting and ending point in the buffer to save only a portion of the data and time format as you do when you save the buffer.

All readings are saved in the comma-separated value (.csv) file format. This format stores tabular data (numbers and text) in plain-text form. You can import the .csv file into a spreadsheet. See the following figures for an example of a buffer .csv file imported into a spreadsheet.

Figure 70: Example of spreadsheet with reading buffer content Sheet 1

Append Mode	1									
Fill Mode	0									
Capacity	600									
Count	600									
Base Time Seconds	17719548									
Base Time Fractional	6964500									
Index	Reading	Unit	Range Digits	Disp Digits	Math	Limit1 High	Limit1 Low	Limit2 High	Limit2 Low	Terminal
1	209997792	O	100000000	5.5	F	F	F	F	F	Front
2	209999392	O	100000000	5.5	F	F	F	F	F	Front
3	209998208	O	100000000	5.5	F	F	F	F	F	Front
4	209999200	O	100000000	5.5	F	F	F	F	F	Front
5	209996112	O	100000000	5.5	F	F	F	F	F	Front
6	209999200	O	100000000	5.5	F	F	F	F	F	Front
7	209999520	O	100000000	5.5	F	F	F	F	F	Front
8	209999392	O	100000000	5.5	F	F	F	F	F	Front
9	209995408	O	100000000	5.5	F	F	F	F	F	Front
10	209997664	O	100000000	5.5	F	F	F	F	F	Front
11	209999440	O	100000000	5.5	F	F	F	F	F	Front
12	209998336	O	100000000	5.5	F	F	F	F	F	Front
13	209999504	O	100000000	5.5	F	F	F	F	F	Front

Figure 71: Example of spreadsheet with reading buffer content Sheet 2

Questionat	Origin	Value	Unit	Digits	Output	Sense	Source Lim	Overtemp	Date	Time	Fractional S
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.006965
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.156923
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.30693
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.456915
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.606918
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.756925
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:48	0.906918
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:49	0.056918
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:49	0.206914
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:49	0.356918
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:49	0.50692
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:49	0.656914
F	Main	0.0000001	A	0.000001	T	2W	T	F	5/2/2013	2:05:49	0.806923

The following table describes the information that is stored in each column of the spreadsheet.

An F in the column indicates the corresponding heading item is false for that reading. For example, if an F is listed in the Math column, the F indicates that the item was not used or did not occur on that reading.

A T in the column indicates that the corresponding heading item is true for that reading. For example, if a T is listed in the Math column, the T indicates that the item was applied to that reading.

Heading	Description
Index	Provides an identifier for each reading
Reading	Measured value for each reading
Unit	Indicates the unit of measure for the reading; values may be any of the following: V (Volts), A (Amps), O (Ohms)
Range Digits	Positive full-scale value of the measurement range that the instrument is presently using; values may be any of the following: ".000000001", ".00000001", ".0000001", ".000001", ".00001", ".0001", ".001", ".01", ".1", "1", "10", "100", "1000", "10000", "100000", "1000000", "10000000", "100000000", "1000000000", "10000000000", "100000000000"
Disp Digits	The number of digits that are displayed for measurements on the front panel. Values may be 3.5, 4.5, 5.5, or 6.5
Math	T when Math is ON; F when Math is off
Limit1 High	Specifies that the upper limit for limit 1 has been exceeded
Limit1 Low	Specifies that the lower limit for limit 1 has been exceeded
Limit2 High	Specifies that the upper limit for limit 2 has been exceeded
Limit2 Low	Specifies that the lower limit for limit 2 has been exceeded
Terminal	Specifies which set of input and output terminals the instrument was using when the measurements were made; values may be any of the following: Front or Rear
Questionable	T or F
Origin	The A/D converter from which the reading originated; for the Model 2450, this will always be Main.
Value	Value is the source value when the reading was taken; if readback is ON, it is the measured source value; otherwise, it is the programmed value
Unit	Units of measure of the source; values may be any of the following: V (Volts), A (Amps), O (Ohms), W (Watts).
Output	On or Off
Sense	2W or 4W
Source Limit	T indicates that the source limit was exceeded; F indicates that the source limit was not exceeded
Overtemp	T indicates that the instrument is in over temperature; F indicates that the instrument is not in over temperature
Date	Date the readings were made
Time	Time the readings were made
Fractional Seconds	Fractional portion of the timestamp (in seconds) when each reading was made

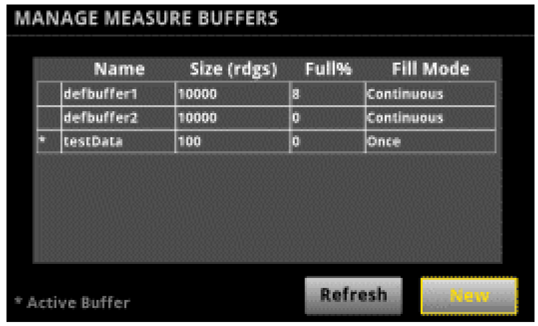
Using the front panel to view the contents of a reading buffer:

1. Press the **MENU** key.
2. Under **Views**, select **Sheet**.
3. Next to Buffer, select the buffer you want to view. The data is displayed.
4. If you want to view the DATA SHEET window for a different buffer, select the **Buffer** button to display the **Select Buffer to View** menu.
5. To view a specific data point, select **Jump To** and enter the number of the data point.
6. Press the **HOME** key to return to the **HOME** screen.

Using the front panel to save or append buffer content to files:

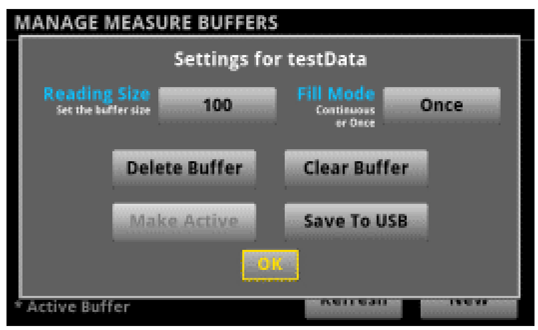
1. Insert a USB drive into the USB port.
2. Press the **MENU** key.
3. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.

Figure 72: MANAGE MEASURE BUFFERS window



4. Select the reading buffer that you want to save. For example, select `testData`. The Settings for testData menu is displayed.

Figure 73: MANAGE MEASURE BUFFERS



5. Select the **Save To USB** button. A keypad is displayed.
6. Enter the name of the file in which to save the readings.

NOTE

You only have to enter the name of the file you want to save. It is not necessary to enter the file extension. All files are saved as `.csv` files.

7. Select **Yes** to confirm saving the file. When the `MANAGE MEASURE BUFFERS` window is displayed again, the file is saved.
8. Press the **HOME** key to return to the **HOME** screen.

Using SCPI commands to save or append buffer content to files:

Before using any of these commands, insert a USB flash drive into the USB port.

To save readings and formatted timestamps from the default buffer to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE "/usb1/myData.csv", "defbuffer1"
```

To save readings and formatted timestamps from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE "/usb1/myData.csv", "testData"
```

To append readings and formatted timestamps from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE:APPend "/usb1/myData.csv", "testData"
```

To append readings and formatted timestamps from a reading buffer named `testData` from point 6 to point 10 in file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE:APPend "/usb1/myData.csv", "testData", FORM, 6, 10
```

Using TSP commands to save or append buffer content to files:

Before using any of these commands, insert a USB flash drive into the USB port.

To save readings from the default buffer to a file named `myData.csv` on a USB flash drive, send the following command:

```
buffer.save(defbuffer1, "/usb1/myData.csv")
```

To save readings from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

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

To append readings from a reading buffer named *`testData`* with default time information to a file named `myData.csv` on the USB flash drive, send the following command:

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

Setting buffer capacity and fill mode

To configure the buffer, you need to make the following settings:

- Setting the buffer capacity
- Setting the buffer fill mode

Buffer capacity

The initial buffer capacity for user-defined buffers is set when the buffer is created. You can resize a user-defined buffer.

The initial buffer capacity for `defbuffer1` and `defbuffer2` is 10,000 readings. You can resize the default buffers.

NOTE

Stored readings and statistics are deleted when you change the capacity of a buffer.

Fill mode

The fill mode setting for the reading buffer controls how the incoming data is managed as the buffer fills. The options are:

- Fill once: The buffer stops accepting data once it fills to capacity and no new data is stored in the buffer.
- Fill continuously: Data fills the buffer normally until the end of the buffer is reached. When the end is reached, the data returns to the beginning of the buffer and overwrites the oldest reading. This is a traditional circular buffer. In this case, the buffer never technically fills.

The following topics provide information about using the front panel to configure buffers and provide an introduction to using remote commands to configure buffers.

For additional information about using remote commands for buffer operations, see the following sections of this manual:

- Remote buffer operation (on page 3-27)
- SCPI commands see TRACe subsystem (on page 6-107)
- TSP command see TSP commands (on page 8-7)

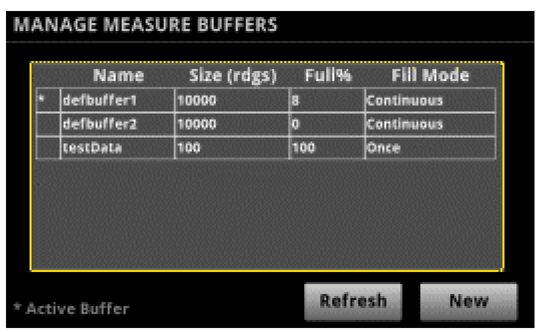
Using the front panel to set buffer capacity:

NOTE

Resizing reading buffers also clears them.

1. Press the **MENU** key.
2. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.

Figure 74: MANAGE MEASURE BUFFERS window



3. Select a reading buffer from the list. For example, touch `testData` to select it. The Settings for testData dialog box is displayed.
4. Select the **Reading Size** button. The New Buffer Size screen is displayed.
5. Enter the new size for the buffer. For example, enter 600, as shown in the following figure.

Figure 75: New Buffer Size

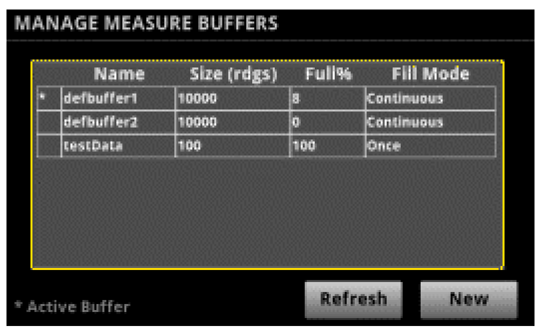
6. Select **OK**. The MANAGE MEASURE BUFFER window is displayed, indicating the new buffer size and that **Full%** = 0.
7. Press the **HOME** key to return to the operating display.

After you complete this procedure, the buffer that you resized becomes the active buffer. The new active buffer name is displayed as the active buffer indicator on the HOME screen.

Using the front panel to set fill mode:

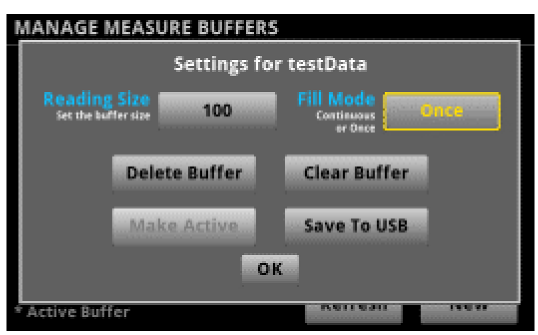
1. Press the **MENU** key.
2. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.

Figure 76: MANAGE MEASURE BUFFERS window



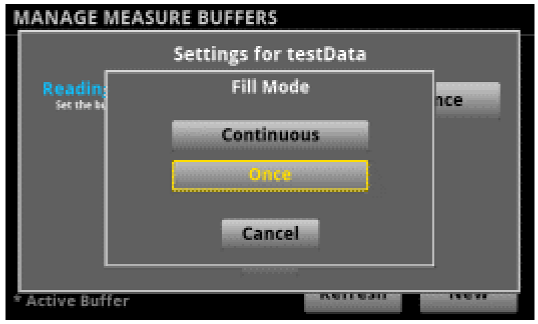
3. Select a reading buffer from the list. For example, touch `testData` to select it. The Settings for testData menu is displayed.

Figure 77: Settings for testData menu



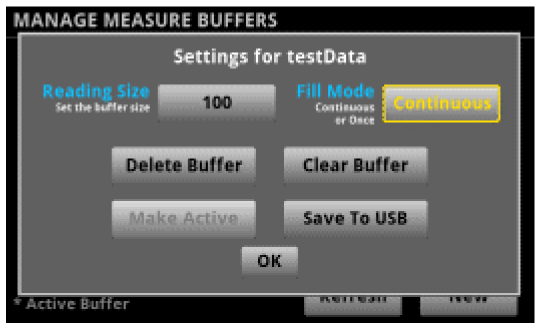
4. Select the **Fill Mode** button. The Fill Mode menu is displayed.

Figure 78: Fill Mode menu



5. Select the Fill Mode. For example, select **Continuous**. The Settings for testData menu is displayed, indicating that the Fill Mode is Continuous.

Figure 79: Settings for testData menu



6. Press the **HOME** key to return to the **HOME** screen.

Using SCPI commands to set the buffer fill mode:

To set the `testData` reading buffer fill mode to continuous, send the following command:

```
TRACe:FILL:MODE CONT, "testData"
```

To set the `defbuffer1` reading buffer fill mode to fill once, send the following command:

```
TRACe:FILL:MODE ONCE, "defbuffer1"
```

Using SCPI commands to set buffer capacity:

To set the `testData` reading buffer to hold 300 readings, send the following command:

```
TRACe:POINts 300, "testData"
```

Using TSP commands to set a buffer fill mode:

To set the `testData` reading buffer fill mode to continuous, send the following command:

```
testData.fillmode = buffer.FILL_CONTINUOUS
```

To set the `defbuffer1` reading buffer fill mode to fill once, send the following command:

```
defbuffer1.fillmode = buffer.FILL_ONCE
```

To print the `defbuffer1` fill mode setting, send the following command:

```
print(defbuffer1.fillmode)
```

Where a return of `0` indicates the buffer is set to fill once and a return of `1` indicates the buffer is set to fill continuously.

Using TSP commands to set buffer capacity:

To set the `testData` reading buffer to hold 300 readings, send the following command:

```
testData.capacity = 300
```

Clearing buffers

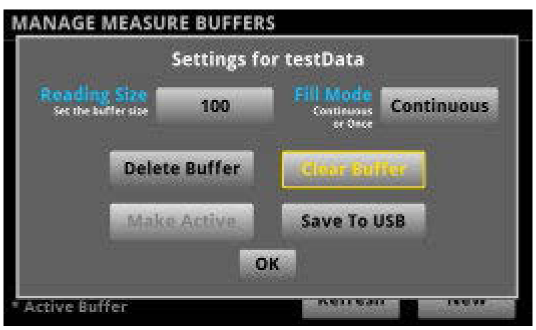
You can clear all readings and statistics from buffers.

The following topics provide information about using the front panel to clear buffers and provide an introduction to using remote commands to clear buffers.

Using the front panel to clear a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.
3. Select a reading buffer from the list. For example, touch `testData` to select it. The Settings for testData menu is displayed.

Figure 80: Settings for testData menu



4. Select **Clear Buffer** to clear the buffer.
5. When the "Are you sure you want to clear testData" prompt is displayed, select **Yes**.
6. Press the **HOME** key to return to the **HOME** screen.

After you clear a reading buffer, the buffer becomes the active buffer. The active buffer indicator on the **HOME** screen displays the name of the active buffer.

Using SCPI commands to clear a buffer:

To clear a user-defined buffer named `testData`, send the following command:

```
TRACE:CLEar "testData"
```

Using TSP commands to clear a buffer:

To clear a user-defined buffer named `testData`, send the following command:

```
testData.clear()
```

Deleting buffers

If you want to save the readings in a buffer before deleting the buffer, save the buffer to USB, see Viewing and saving buffer content (on page 3-18).

You cannot delete the default buffers `defbuffer1` or `defbuffer2`. However, the data in the default buffers is lost when the instrument is reset or the power is turned off.

Using the front panel to delete a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.
3. Select a reading buffer from the list. For example, touch `testData` to select it. The Settings for testData menu is displayed.
4. Select **Delete** to delete the buffer.
5. When the "Are you sure you want to delete testData" prompt is displayed, select **Yes**.
6. Press the **HOME** key to return to the **HOME** screen.

Using SCPI commands to delete a buffer:

To delete a user-defined buffer named `testData`, send the following command:

```
:TRACe:DELete "testData"
```

Using TSP commands to delete a buffer:

To delete a user-defined buffer named `testData`, send the following command:

```
buffer.delete(testData)
```

Remote buffer operation

You can control the Model 2450 buffers through a remote interface using SCPI or TSP remote commands.

This section provides a summary of some of the remote commands available to control and access data stored in buffers; however, this section does not describe all of the available commands. See the following sections for command descriptions:

- For information about SCPI commands, see the SCPI command reference (on page 6-1)
- For information about TSP commands, see the TSP command reference (on page 8-1)

Storing data in buffers

Using SCPI commands:

The following SCPI commands are associated with data storage operation:

Command	Description
`:TRACe:MAKE`	This function creates a user-defined reading buffer. You cannot use this command on the default buffers. See Creating buffers (on page 3-12). Also see :TRACe:MAKE (on page 6-114).
`:TRACe:SAVE`	This function saves data from the specified reading buffer to a USB flash drive. See Storing readings in buffers (on page 3-17). Also see :TRACe:SAVE (on page 6-116).
`:TRACE:SAVE:APPend`	This function appends data from the reading buffer to a file on the USB flash drive. See :TRACE:SAVE:APPend (on page 6-118).
`:TRACe:POINts`	This command reads the number of readings a buffer can store. This allows you to change the number of readings the buffer can store. See :TRACe:POINts (on page 6-115).
`:TRACe:CLEar`	This command clears all readings and statistics from the specified buffer. See Clearing buffers (on page 3-26). Also see :TRACe:CLEar (on page 6-108).
`:TRACe:STATistics:CLEar`	This command clears the statistical information associated with the specified buffer. This command does not clear the readings.
`:TRACe:FILL:MODE`	This command determines if a reading buffer is filled continuously or is filled once and stops. See :TRACe:FILL:MODE (on page 6-112).
`:TRACe:LOG:STATe`	This command indicates whether the reading buffer should log informational events. See :TRACe:LOG:STATe (on page 6-113).
`:TRACe:ACTual?`	This command contains the number of readings in the specified buffer. See :TRACe:ACTual? (on page 6-107)
`:TRACe:DELete`	This command deletes a buffer. See :TRACe:DELete (on page 6-111).

Using TSP commands:

CAUTION

Once you create a reading buffer using TSP commands, if you use that buffer name for another buffer or variable, you can no longer access the original buffer.

The following TSP commands are associated with data storage operation:

Command	Description
`buffer.clearstats()`	This function clears all statistics from the specified buffer. This function does not clear the readings. See buffer.clearstats() (on page 8-8).
`buffer.delete()`	This function deletes a user-defined reading buffer. See buffer.delete() (on page 8-9).
`buffer.make()`	This function creates a user-defined reading buffer. You cannot use this command on the default buffers. See Creating buffers (on page 3-12). Also see buffer.make() (on page 8-11).
`buffer.save()`	This function saves data from the specified reading buffer to a USB flash drive. See Storing readings in buffers (on page 3-17). Also see buffer.save() (on page 8-12).
`buffer.saveappend()`	This function appends data from the reading buffer to a file on the USB flash drive. See buffer.saveappend() (on page 8-14).
bufferVar`.capacity`	This command reads the number of readings a buffer can store. This allows you to change the number of readings the buffer can store. See bufferVar.capacity (on page 8-15).
bufferVar`.clear()`	This function clears all readings and statistics from the specified buffer. See Clearing buffers (on page 3-26). Also see bufferVar.clear() (on page 8-17).
bufferVar`.fillmode`	This attribute determines if a reading buffer is filled continuously or is filled once and stops. See bufferVar.fillmode (on page 8-19).
bufferVar`.logstate`	This attribute indicates whether the reading buffer should log informational events. See bufferVar.logstate (on page 8-22).
bufferVar`.n`	This attribute contains the number of readings in the specified buffer. See bufferVar.n (on page 8-23).

Accessing the data in buffers

Using SCPI commands:

To access a buffer, include the buffer name in the respective command. For example, the following commands return five readings (including the measurement, source value, and relative time) from a user-defined buffer named `testData`:

```
-- Create a buffer named testData to store 100 readings.
TRAC:MAKE "testData", 100
-- Set the instrument to make 5 readings for all measurement requests.
SENS:COUN 5
-- Make the readings and store them in the buffer
TRAC:TRIG "testData"
-- Read the 5 readings including the measurement, source value, and relative time
   for each point.
TRAC:DATA? 1, 5, "testData", MEAS, SOUR, REL
```

Using TSP commands:

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

To access a buffer, include the buffer name in the respective command. For example, the following commands return five readings (including the measurement, source value, and relative time) from a user-defined buffer named `testData`:

```
-- Create a buffer named testData to store 100 readings.
testData = buffer.make(100)
-- Set the instrument to make 5 readings and store them in the buffer.
trigger.model.load("SimpleLoop", 5, 0, testData)
-- Make the readings
trigger.model.initiate()
waitcomplete()
-- Read the 5 readings and print them including the measurement, source value, and
   relative time for each reading.
printbuffer(1, 5, testData.readings,
   testData.sourcevalues,testData.relativetimestamps)
```

Buffer read-only attributes

Use buffer read-only attributes to access the information contained in an existing buffer.

Using SCPI commands:

The following attributes are available for each reading buffer.

Attribute	Description
`:TRACe:ACTual?`	This attribute contains the number of readings in the specified buffer. See :TRACe:ACTual? (on page 6-107)
`:TRACe:DATA`	This attribute contains the readings stored in a specified reading buffer. See .:TRACe:DATA? (on page 6-109).
`:TRACe:STATistics:AVERage`	This command returns average of all readings added to the buffer. See :TRACe:STATistics:AVERage? (on page 6-119)
`:TRACe:STATistics:MAXimum?`	This command returns the maximum reading value added to the buffer. See :TRACe:STATistics:MAXimum? (on page 6-121)
`:TRACe:STATistics:MINimum?`	This command returns the minimum reading value added to the buffer. :TRACe:STATistics:MINimum? (on page 6-121)
`:TRACe:STATistics:PK2Pk?`	This command returns the peak-to-peak value of all readings (samples) added to the buffer. See :TRACe:STATistics:PK2Pk? (on page 6-122).
`:TRACe:STATistics:STDDev?`	This command returns the standard deviation of all readings (samples) added to the buffer. See :TRACe:STATistics:STDDev? (on page 6-123).

Using TSP commands:

The following attributes are available for each reading buffer (for example, `rb.dates` accesses dates for reading buffer `rb`, and the number of readings in the reading buffer is accessed as `rb.n`).

Attribute	Description
bufferVar.n	This attribute contains the number of readings in the specified buffer. See bufferVar.n (on page 8-23).
bufferVar.readings	This attribute contains the readings stored in a specified reading buffer. See bufferVar.readings (on page 8-24).
bufferVar.dates	This attribute contains the dates of readings stored in the reading buffer. See bufferVar.dates (on page 8-18).
bufferVar.statuses	This attribute contains the status values of readings in the reading buffer. See bufferVar.statuses (on page 8-33).
bufferVar.formattedreadings	This attribute contains the stored readings formatted as they appear on the front-panel display. See bufferVar.formattedreadings (on page 8-20).
bufferVar.sourceformattedvalues	This attribute contains the source levels formatted as they appear on the front-panel display when the readings in the reading buffer were acquired. See bufferVar.sourceformattedvalues (on page 8-28).
bufferVar.sourcevalues	This attribute contains the source levels being output when readings in the reading buffer were acquired. See bufferVar.sourcevalues (on page 8-32).
bufferVar.sourcestatuses	This attribute contains the source status conditions of the instrument for the reading point. bufferVar.sourcestatuses (on page 8-29).
bufferVar.times	This attribute contains the time when the instrument made the readings. See .bufferVar.times (on page 8-34).
bufferVar.timestamps	This attribute contains the timestamps of readings stored in the reading buffer. See bufferVar.timestamps (on page 8-35).
bufferVar.relativetimestamps	This attribute contains the timestamps, in seconds, when each reading occurred relative to the timestamp of reading buffer entry number 1. See bufferVar.relativetimestamps (on page 8-25).
bufferVar.sourceunits	This attribute contains the units of measure of the source. See bufferVar.sourceunits (on page 8-30).
bufferVar.seconds	This attribute contains the nonfractional seconds portion of the timestamp when the reading was stored in UTC format. See bufferVar.seconds (on page 8-26).
bufferVar.fractionalseconds	This attribute contains the fractional portion of the timestamp (in seconds) when each reading occurred. See bufferVar.fractionalseconds (on page 8-21).

Reading buffer time and date values

Time and date values are represented as a number of UTC seconds since 12:00 a.m. Jan. 1, 1970. Use the following commands to return values in the following formats:

- Hours and minutes: bufferVar.times (on page 8-34)
- UTC seconds: bufferVar.seconds (on page 8-26)
- Month, day, year, format, or to access the timestamp table: bufferVar.dates (on page 8-18)

Reading buffer for . . . do loops

The following TSP examples illustrate the use of for . . . do loops when recalling data from a reading buffer called `mybuffer`. The following code may be sent as one command line or as part of a script. Sample outputs follow the line of code. Also see the printbuffer() (on page 8-78) command.

This example loop uses the `printbuffer()` command to show the reading, units, and relative timestamps for all readings stored in the reading buffer. The information for each reading (reading, units, and relative timestamps) is shown on a single line with the elements comma-delimited.

```
for x = 1, mybuffer.n do
   printbuffer(x,x,mybuffer, mybuffer.units, mybuffer.relativetimestamps)
end
```

Example comma-delimited output of above code:

```
-1.5794739960384e-09, Amp DC, 0
-1.5190692453926e-11, Amp DC, 0.411046134
-2.9570144943758e-11, Amp DC, 0.819675745
-2.9361919146043e-11, Amp DC, 1.228263492
-3.0666566508408e-11, Amp DC, 1.636753752
-4.0868204653766e-11, Amp DC, 2.034403917
```

The following loop uses the `print` command instead of the `printbuffer` command. This loop shows the same information described in the previous example (reading, units, and relative timestamps for all readings stored in the buffer). However, because the `print()` command is used instead of `printbuffer()`, each line is tab-delimited (rather than comma-delimited) to produce a columnar output, as shown below:

```
for x = 1, mybuffer.n do
   print(mybuffer.readings[x], mybuffer.units[x], mybuffer.relativetimestamps[x])
end
```

Example columnar-delimited output of above code:

```
-1.5794739960384e-09 Amp DC      0
-1.5190692453926e-11 Amp DC      0.411046134
-2.9570144943758e-11 Amp DC      0.819675745
-2.9361919146043e-11 Amp DC      1.228263492
-3.0666566508408e-11 Amp DC      1.636753752
-4.0868204653766e-11 Amp DC      2.034403917
```

Exceeding reading buffer capacity

When the reading buffer fill mode is set to fill once and the reading buffer count is not exceeded, readings are stored as expected. But if new readings would exceed reading buffer capacity when they are added to the active buffer index, the count is lowered to a new count so it does not exceed the reading buffer capacity. Once the reading buffer is full (to the new count), no more readings are taken and error message 4915 is displayed, stating that you attempted to exceed the capacity of the reading buffer. If you attempt to store additional readings in a full reading buffer, the same message appears, and no readings are taken.

Configuration lists

Instrument configuration

An instrument configuration is a collection of settings that can be applied to the instrument.

Active setting
At any given time, the instrument is operating using its active settings. For example, if you set the measurement NPLC to 1.0, the active NPLC setting is 1.0.

Active state
At any given time, the complete set of active settings of the instrument is the active state. These active settings can be subdivided into the following groups:

- Measure settings
- Source settings
- All other general settings

The active state of the instrument changes when:

- You use the front panel of the instrument to change settings.
- You send commands to the instrument that change settings.
- You use a configuration list to recall source or measure settings.
- You run a configuration script (TSP or front panel) or use `*RCL` (SCPI) to recall all instrument settings.

When you create a new source or measure configuration list, it is important to remember that all instrument source or measure settings are included in its active state, not just the specific settings that changed immediately before setting up the configuration list.

What is a configuration list?

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

Configuration lists allow you to record the function settings of the instrument, store them, and then return the instrument to those settings as needed.

You can recall configuration lists from the front panel, using remote commands, or as part of a trigger model.

The following figure shows an example of a three-point source configuration list. Where:

- Each tab represents a configuration list
- Each row represents a configuration point
- Each column holds the stored setting corresponding to that configuration point.

When you recall a configuration point, you recall the settings in one row.

Figure 81: Database of configuration points

	A	B	C	D	E	F	G
1	**Point**	**Function**	**Auto range**	**Delay**	**Level**	**...**	**User Delay5**
2	1	Voltage	Off	3	100	...	0.000000
3	2	Current	Off	3	1.0 A	...	0.000000
4	3	Current	Off	3	0.5 A	...	0.000000

MySourceList MyTestList

If you want to use the same configuration list on multiple Model 2450 instruments, you have to recreate it on each instrument. You can do this using one of the following methods:

- Define the commands to create the configuration list, then send the commands to multiple instruments.
- Create a user-defined saved setup and run it on the other instruments.

Configuration list types

There are different types of configuration lists for different types of instruments. The Model 2450 supports source configuration lists and measure configuration lists, making it possible to sequence through defined source settings, measurement settings, or both. If you are familiar with the Model 2400, using configuration lists offers you similar functionality to using Source Memory.

Configuration lists and the trigger model

If you think of the trigger model as the execution engine that makes the instrument do things, configuration lists provide a database of stored settings that the trigger model can recall to change the settings of the instrument at any time during trigger model execution. Refer to Trigger model (on page 3-95) for more information.

What is a configuration point?

A configuration point contains a copy of all instrument source or measure active settings at a specific point in time. You store configuration points in a specific configuration list. Only the amount of available memory limits the number of configuration points that you can store in a configuration list. Lists can typically exceed 100,000 points.

Each configuration point is identified by an index. To overwrite an existing point, you can provide the index when you store the configuration point. Otherwise, the instrument appends the configuration point to the end of the list. Configuration points are chronologically numbered with the index starting from 1.

The first time you create a configuration point and store into it, the instrument stores the active settings to configuration point 1. Each time you store another set of active settings to the same list, the instrument creates a new configuration point and appends it to the list using the next chronological index.

You can use the index to identify a specific configuration point and perform operations on it when necessary.

Although you can specify a specific configuration point index when you store active settings to a configuration list, this is only necessary if you wish to overwrite an existing point. Normally, you can build up the configuration points in a configuration list by appending (no index specified) subsequent configuration points to the list.

If you only store one configuration point to a list, the list will consist of configuration point 1.

What settings are stored in a configuration list?

Specific instrument settings are stored in each type of configuration list. The same settings are recalled to overwrite the active state when you recall a configuration list.

The first time you use the front panel or remote commands to store a configuration list, the instrument stores the active settings to configuration point 1. Each time you append a configuration point to the configuration list, the instrument saves the active values for active settings to settings in a configuration point.

When you recall a configuration point on the list, the instrument restores the settings to the values that were stored. The recall operation overwrites the active settings with the stored settings.

It does not matter how many configuration points are on a configuration list, you can only recall one configuration point at a time.

To see actual values for the settings saved to a configuration point refer to Viewing configuration list contents (on page 3-42).

Instrument settings stored in a measure configuration list

When you save a configuration point to a measure configuration list, the instrument saves the values for each setting listed in the following table. These settings can be set from the front panel or by using remote commands. The table shows front panel, TSP, and SCPI settings that can be used to set these settings.

Measure configuration list settings		
Front-panel setting	**TSP command**	**SCPI command**
Function FUNCTION key	smu.measure.func (on page 8-103)	[:SENSe[1]]:FUNCtion[:ON] (on page 6-49)
Auto range HOME > Range	smu.measure.autorange (on page 8-86)	[:SENSe[1]]:<function>:RANGe:AUTO (on page 6-50)
Auto range low MENU > Measure > Settings > Auto Range Low Limit	smu.measure.autorangelow (on page 8-88)	[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 6-51)
Display digits SETTINGS swipe screen > Display Digits	smu.measure.displaydigits (on page 8-100)	:DISPlay:<function>:DIGits (on page 6-23)
Filter MENU > Measure > Filter/Math > Filter State	smu.measure.filter.enable (on page 8-101)	[:SENSe[1]]:<function>:AVERage[:STATe] (on page 6-44)
Filter type MENU > Measure > Filter/Math > Filter State Type	smu.measure.filter.type (on page 8-102)	[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 6-45)
Filter count MENU > Measure > Filter/Math > Filter Count	smu.measure.filter.count (on page 8-100)	[:SENSe[1]]:<function>:AVERage:COUNt (on page 6-43)
NPLC SETTINGS swipe screen > NPLCs	smu.measure.nplc (on page 8-116)	[:SENSe[1]]:<function>:NPLCycles (on page 6-48)
Offset compensation MENU > Measure > Settings > Offset Comp (when function is set to source current and measure resistance)	smu.measure.offsetcompensation (on page 8-117)	[:SENSe[1]]:<function>:OCOMpensated (on page 6-49)
Relative offset SETTINGS swipe > Rel	smu.measure.rel.enable (on page 8-121)	[:SENSe[1]]:<function>:RELative:STATe (on page 6-57)
Relative offset value Not available from front panel	smu.measure.rel.level (on page 8-122)	[:SENSe[1]]:<function>:RELative:ACQuire (on page 6-56)
Terminals TERMINALS button	smu.measure.terminals (on page 8-124)	:ROUTe:TERMinals (on page 6-35)
Autozero enable MENU > Measure > Settings > Auto Zero	smu.measure.autozero.enable (on page 8-89)	[:SENSe[1]]:<function>:AZERo[:STATe] (on page 6-46)
Math MENU > Measure > Filter/Math > Math State	smu.measure.math.enable (on page 8-111)	:CALCulate[1]:<function>:MATH:STATe (on page 6-12)
Math format MENU > Measure > Filter/Math > Math Function	smu.measure.math.format (on page 8-112)	:CALCulate[1]:<function>:MATH:FORMat (on page 6-7)
Math mx+b b factor MENU > Measure > Filter/Math > b (Offset)	smu.measure.math.mxb.bfactor (on page 8-113)	:CALCulate[1]:<function>:MATH:MMFactor (on page 6-10)

Math mx+b m factor MENU > Measure > Filter/Math > m (Scalar)	smu.measure.math.mxb.mfactor (on page 8-114)	:CALCulate[1]:<function>:MATH:MBFactor (on page 6-8)
Math percent MENU > Measure > Filter/Math > Percent	smu.measure.math.percent (on page 8-115)	:CALCulate[1]:<function>:MATH:PERCent (on page 6-11)
Units Not available from front panel	smu.measure.unit (on page 8-125)	[:SENSe[1]]:<function>:UNIT (on page 6-59)
Limit 1 and Limit 2 MENU > Measure > Settings > Limits > State	smu.measure.limit[Y].enable (on page 8-106)	:CALCulate2:<function>:LIMit<y>:STATe (on page 6-17)
Limit auto clear MENU > Measure > Settings > Limits > Auto Clear	smu.measure.limit[Y].autoclear (on page 8-104)	:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 6-13)
Limit low value MENU > Measure > Settings > Limits > Low Value	smu.measure.limit[Y].low.value (on page 8-110)	:CALCulate2:<function>:LIMit<y>:LOWer[:DATA] (on page 6-16)
Limit high value MENU > Measure > Settings > Limits > High Value	smu.measure.limit[Y].high.value (on page 8-109)	:CALCulate2:<function>:LIMit<y>:UPPer[:DATA] (on page 6-18)
Limit fail Displayed on home screen	smu.measure.limit[Y].fail (on page 8-107)	:CALCulate2:<function>:LIMit<y>:FAIL? (on page 6-15)
User delay (1 through 5) Not available from front panel	smu.measure.userdelay[N] (on page 8-126)	[:SENSe[1]]:<function>:DELay:USER<n> (on page 6-47)
Performance QUICKSET > Performance	smu.measure.performance	Not available from SCPI

Instrument settings stored in a source configuration list

When you save a configuration point to a source configuration list, the instrument saves the values for each setting listed in the following table. These settings can be set from the front panel or by using remote commands. The table lists SCPI and TSP commands that can be used to set these settings.

Source configuration list settings		
Front-panel setting	**TSP command**	**SCPI command**
Function FUNCTION key	smu.source.func (on page 8-134)	:SOURce[1]:FUNCtion[:MODE] (on page 6-72)
AutoRange HOME > Source Range	smu.source.autorange (on page 8-127)	:SOURce[1]:<function>:RANGe:AUTO (on page 6-75)
Delay MENU > Source > Settings > Source Delay	smu.source.delay (on page 8-133)	:SOURce[1]:<function>:DELay (on page 6-66)
Auto delay Not available from the front panel	smu.source.autodelay (on page 8-128)	:SOURce[1]:<function>:DELay:AUTO (on page 6-67)
Source level HOME > Source	smu.source.level (on page 8-135)	:SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 6-71)
Overvoltage protection MENU > Source > Settings > Overvoltage Protection Limit	smu.source.protect.level (on page 8-141)	:SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73)
Output off state MENU > Source > Settings > Output Off State	smu.source.offmode (on page 8-139)	:OUTPut[1]:<function>:SMODe (on page 6-31)
Range MENU > Source > Settings > Source Range	smu.source.range (on page 8-143)	:SOURce[1]:<function>:RANGe (on page 6-74)
Limit HOME > Limit	smu.source.xlimit.level (on page 8-137)	:SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 6-71)
High capacitance MENU > Source > Settings > Hi Capacitance	smu.source.highc (on page 8-135)	:SOURce[1]:<function>:HIGH:CAPacitance (on page 6-69)
Readback MENU > Source > Settings > Source Readback	smu.source.readback (on page 8-144)	:SOURce[1]:<function>:READ:BACK (on page 6-77)
User delays Not available from the front panel	smu.source.userdelay[N] (on page 8-153)	:SOURce[1]:<function>:DELay:USER<n> (on page 6-68)
Source output Output on/off switch	smu.source.output (on page 8-140)	:OUTPut[1][:STATe] (on page 6-34)

Creating, storing, and performing operations on configuration lists and points

To create a configuration point, you need to:

- Create a new configuration list and give it a name or use a specific configuration list that already exists on the instrument
- Configure the instrument with the settings that you want to store in a configuration point
- Store the active settings into a configuration point on the specified configuration list

After you store configuration points to a configuration list, you can perform the following operations on a specific configuration point:

- Recall a configuration point and restore the stored settings to the active state
- View the contents of a configuration point
- Delete a configuration point or delete the entire configuration list

You can perform configuration lists operations from the front panel (see Using the front panel for configuration list operations (on page 3-39)) or by using remote commands (see Using remote commands for configuration list operations (on page 3-43)).

Using the front panel for configuration list operations

This section describes how to store the active settings to a specific point on a configuration list.

See What is a configuration point? (on page 3-35) for information about how the instrument adds configuration points to configuration lists before reading this section.

The following topics provide information to:

- Create an example measure configuration list named `MyMeasList`.
- Store two example configuration points on `MyMeasList`. The following table provides information about specific settings for each configuration point.
- View the contents of a specific configuration point.
- Recall a configuration point from `MyMeasList`.

NOTE

This example creates a measure configuration list; however, you can use the same process to create a source configuration list.

Front panel configuration list menu overview

To display the configuration list menus from the main menu:

1. Press the **MENU** key.
2. Under Measure, select **Config List**. The MEASURE CONFIGURATION LISTS screen is displayed. Use this menu for operations on measure configuration lists.
3. Under Source, select **Config List**. The SOURCE CONFIGURATION LISTS screen is displayed. Use this menu for operations on source configuration lists.

The CONFIGURATION LIST menu contains:

- A scrollable list of configuration points that are stored in the selected configuration list
- A message bar indicating the index associated with the selected configuration point
- Buttons to perform the operations, described below

Button	Use to ...
Select	If you selected **Source** on the main menu, a menu of the source configuration lists presently available on the instrument is displayed. If you selected **Measure** on the main menu, a menu of the measure configuration lists presently available on the instrument is displayed.
New List	Create a new configuration list.
Delete Point	Delete the selected configuration point.
System to List	Store the active settings into a configuration point on the specified configuration list. New points are chronologically numbered with the index starting from 1.
List to System	Recall a configuration point and restore the stored settings to the active state.
View Details	Display the details of the selected configuration point.

Duplicate configuration points:

If you store a second configuration point that has the same settings as a point that is already on the configuration list, the "No change" message is displayed.

Creating a configuration list and giving it a name

For example, use the following information to create `MyMeasList`.

Using the front panel to create the configuration list:

1. Press the **MENU** key.
2. Under Measure, select **Config List**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **New List**. The keypad is displayed.
4. Enter a name for the configuration list you are creating. For example, `MyMeasList`.
5. Select the **Enter** button on the keypad. The MEASURE CONFIGURATION LISTS screen is displayed.
6. Select **HOME** to return to the home screen.

Storing configuration point 1

For example, use the following information to store configuration point 1 to `MyMeasList`.

Using the front panel to configure the instrument:

Configure the instrument with the settings you want to store in the configuration point. For example:

1. Press the **QUICKSET** key and select the **SrcV MeasI** function to source voltage and measure current.
2. Press the **MENU** key.
3. Under Measure, select **Settings**. The MEASURE SETTINGS menu is displayed.
4. Select **Measure Range**, then select **10 nA**.
5. Select **NPLC**, then select **1.00**.

Using the front panel to store active settings to configuration point 1:

Store all active measure settings to `MyMeasList` as configuration point 1 by appending to the end of the initially empty list.

1. Press the **MENU** key.
2. Under Measure, select **Config List**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Choose **Select**. A menu of available configuration lists is displayed.
4. Select **MyMeasList**.
5. Select **System to List**. This saves the active system settings to the configuration point. The configuration point is displayed on the list.

   Select **View Details** to see a subset of the settings stored in configuration point 1. See Instrument settings stored in a measure configuration list (on page 3-36) for information about all settings that are stored.
6. Select **HOME** to return to the home screen.

Storing configuration point 2

Refer to the instructions in Storing configuration point 1 (on page 3-40), and proceed as follows to configure the instrument for configuration point 2.

Using the front panel to configure the instrument:

Change the following instrument settings to configure the instrument with the settings you want to save for configuration point 2:

- Set the measure range to 100 nA
- Set NPLC to 2

Using the front panel to store the active settings to configuration point 2:

Store all active measure settings to `MyMeasList` as configuration point 2 by appending to the end of the list.

1. Return to the configuration list menu and select **MyMeasList**.
2. Select **System to List**. This saves the active system settings to the configuration point.

   Select **View Details** to see a subset of settings stored in configuration point 2. See Instrument settings stored in a measure configuration list (on page 3-36) for information about the settings that are stored.
3. Select **HOME** to return to the home screen.

Recalling a configuration point

You can recall the settings stored in a specific configuration point in a configuration list.

For example, use the following procedure to recall configuration point 2 from `MyMeasList`.

Using the front panel to recall a configuration point:

1. Press the **MENU** key.
2. Under **Measure**, select **Config List**. The MEASURE CONFIGURATION LISTS screen is displayed.

NOTE

If you want to recall a point on a source configuration list, under **Source**, select **Config List**.

3. Choose **Select List**. A menu of available configuration lists is displayed.
4. Select **MyMeasList**. The configuration points in the list display.
5. Select the second configuration point.
6. Select **List to System**.
7. Select **HOME** to return to the operating display.

Viewing configuration list contents

For example, use the following procedure to view configuration point 2 from `MyMeasList`.

Using the front panel to view configuration list contents:

1. Press the **MENU** key.
2. Under **Measure**, select **Config List**. The MEASURE CONFIGURATION LISTS screen is displayed.

NOTE

If you want to view a source configuration list, under **Source**, select **Config List**.

3. Choose **Select List**. A menu of available configuration lists is displayed.
4. Select **MyMeasList**. The configuration points are displayed.
5. Select the second configuration point.
6. Select **View Details**.
7. When you are finished, select **OK**.
8. Select **HOME** to return to the home screen.

Deleting a configuration point

For example, use the following procedure to delete configuration point 4 from `MyMeasList`.

Using the front panel to delete a configuration point:

1. Press the **MENU** key.

   Under **Measure**, select **Config List**. The MEASURE CONFIGURATION LISTS screen is displayed.

NOTE

If you want to delete a point from a source configuration list, under **Source**, select **Config List**.

2. Choose **Select List**. A menu of available configuration lists is displayed.
3. Select **MyMeasList**. The configuration points are displayed.
4. Select the second configuration point.
5. Select **Delete From**.
6. Select **HOME** to return to the operating display.

Using remote commands for configuration list operations

The following topics provide information to:

- Create an example source configuration list named `MySourceList`.
- Store four example configuration points on `MySourceList`.
- View the contents of a specific configuration point.
- Recall a configuration point from `MySourceList`.

The following table provides information about specific settings for each configuration point.

Settings	**Index 1**	**Index 2**	**Index3**	**Index 4**
Function	Voltage	Voltage	Voltage	Voltage
AutoRange	Off	Off	Off	Off
Delay	0.001	0.001	0.001	0.001
AutoDelay	Off	Off	Off	Off
Source Level	2.0 V	**3.0 V**	**4.0 V**	**5.0 V**
ProtectLevel	PROTECT_40	PROTECT_40	PROTECT_40	PROTECT_40
ProtectTripped	Not tripped	Not tripped	Not tripped	Not tripped
OffMode	OFFMODE_ NORMAL	OFFMODE_ NORMAL	OFFMODE_ NORMAL	OFFMODE_ NORMAL
Range	20.0 V	20.0 V	20.0 V	20.0 V
LimitLevel	10.6 nA	10.6 nA	10.6 nA	10.6 nA
LimitTripped	Not tripped	Not tripped	Not tripped	Not tripped
HighC	Off	Off	Off	Off
Readback	On	On	On	On
UserDelay1-5	None	None	None	None

The figures in this section include a graphic representation of `MySourceList` and the four configuration points. For simplicity, the figures will only show two of the source settings listed in the table (source range and source level).

NOTE

This example creates a source configuration list; however, you can use the same process to create a measure configuration list.

Creating a configuration list and giving it a name

Use one of the following methods to create the configuration list used in this example and give it a name.

Using SCPI commands:

```
:SOURce:CONFiguration:LIST:CREate "MySourceList"
```

Using TSP commands:

```
smu.source.configlist.create("MySourceList")
```

Storing configuration point 1

Use one of the following methods to:

- Set the instrument source function to voltage
- Set the instrument source range to 20 V
- Set the instrument source limit to 2.0 V
- Store all active source settings to `MySourceList` as configuration point 1 by appending to the end of the initially empty list

Using SCPI commands:

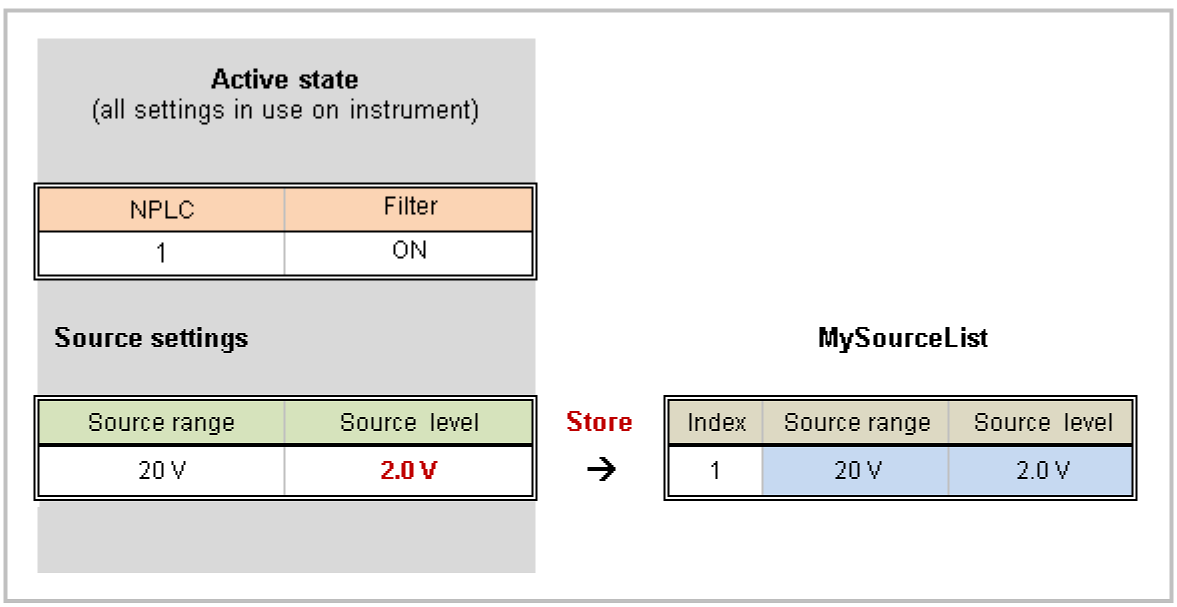
```
:SOURce:FUNC VOLTage
:SOURce:VOLtage:RANGe 20
:SOURce:VOLTage:LEVel 2
:SOURce:CONF:LIST:STORe "MySourceList"
```

Using TSP commands:

```
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.source.level = 2
smu.source.configlist.store("MySourceList")
```

The following figure shows the active state of the instrument after you change the source level to 2.0 V. Refer to Instrument settings stored in a source configuration list (on page 3-38) for a complete list of source settings that the instrument stores in a source configuration list.

Figure 82: Example configuration point 1



Storing configuration point 2

Use one of the following methods to:

- Set the instrument source level to 3.0 V
- Store all active source settings to `MySourceList` as configuration point 2 by appending to the end of the list

Using SCPI commands:

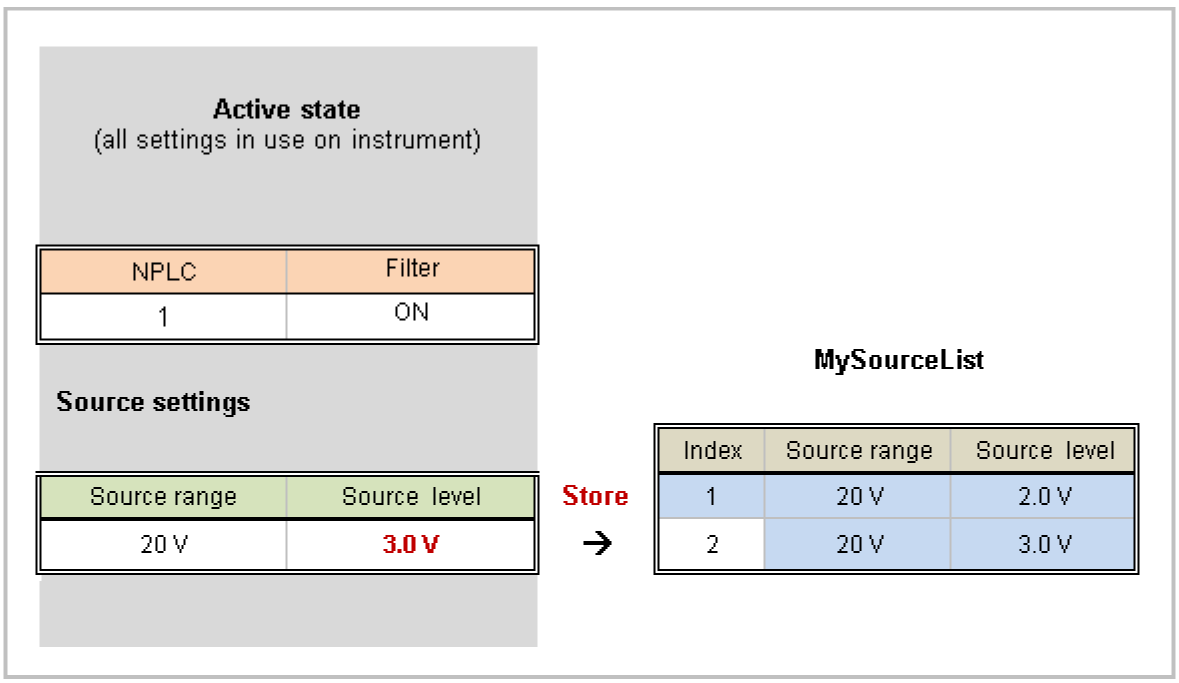
```
:SOURce:VOLTage:LEVel 3
:SOURce:CONF:LIST:STORe "MySourceList"
```

Using TSP commands:

```
smu.source.level = 3
smu.source.configlist.store("MySourceList")
```

The following figure shows the active state of the instrument after you change the source level to 3.0 V. Refer to Instrument settings stored in a source configuration list (on page 3-38) for a complete list of source settings that the instrument stores in a source configuration list.

Figure 83: Example configuration point 2



Storing configuration point 3

Use one of the following methods to:

- Set the time it takes to perform the current measurement (the NPLC setting) to 2 seconds
- Set the instrument source level to 4.0 V
- Store all active source settings to `MySourceList` as configuration point 3 by appending to the end of the list

In this example, the NPLC is set to 2 seconds to demonstrate that a setting not part of the source configuration list settings will not be saved.

Using SCPI commands:

```
:SENSe:CURRent:NPLCycles 2
:SOURce:VOLTage:LEVel 4
:SOURce:CONF:LIST:STORe "MySourceList"
```

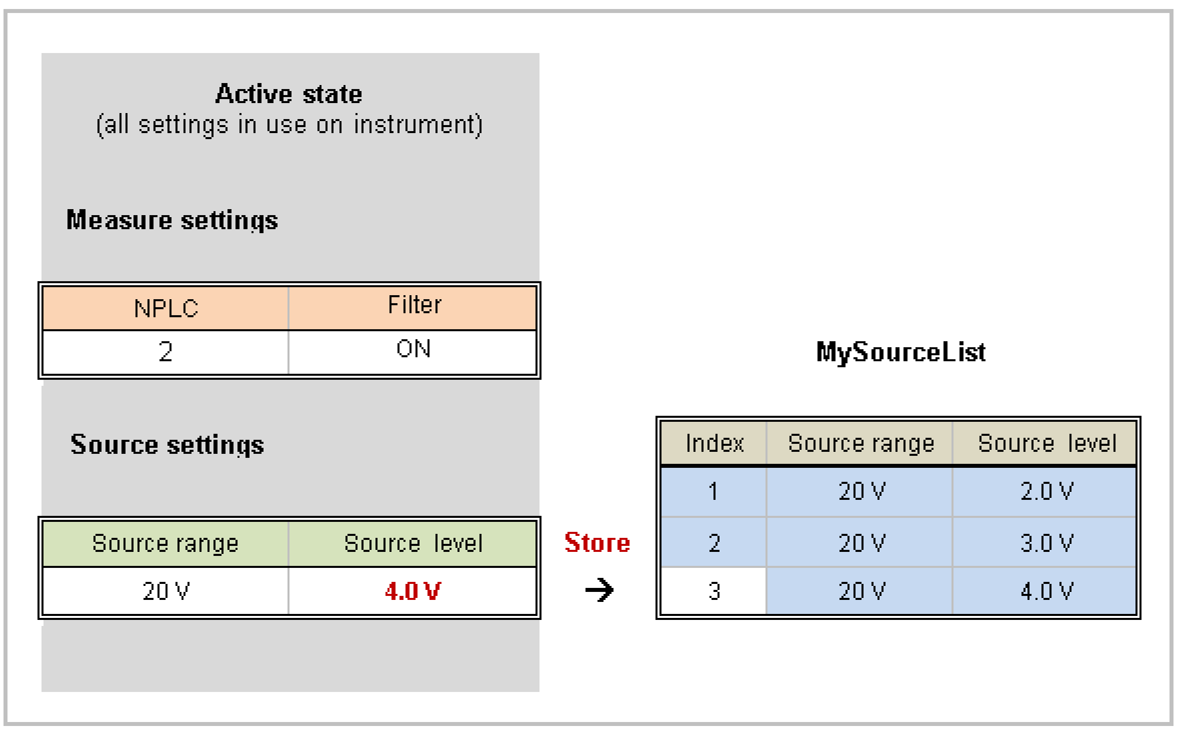
Using TSP commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.nplc = 2
smu.source.level = 4
smu.source.configlist.store("MySourceList")
```

The following figure shows the active state of the instrument after you change the source level to 4.0 V. Notice that the NPLC setting is not stored in `MySourceList` because this setting is not a stored source setting.

Refer to Instrument settings stored in a source configuration list (on page 3-38) for a complete list of source settings that the instrument stores in a source configuration list.

Figure 84: Example configuration point 3



Storing configuration point 4

Use one of the following methods to:

- Set the instrument source level to 5.0 V
- Store all active source settings to `MySourceList` as configuration point 4 by appending to the end of the list

Using SCPI commands:

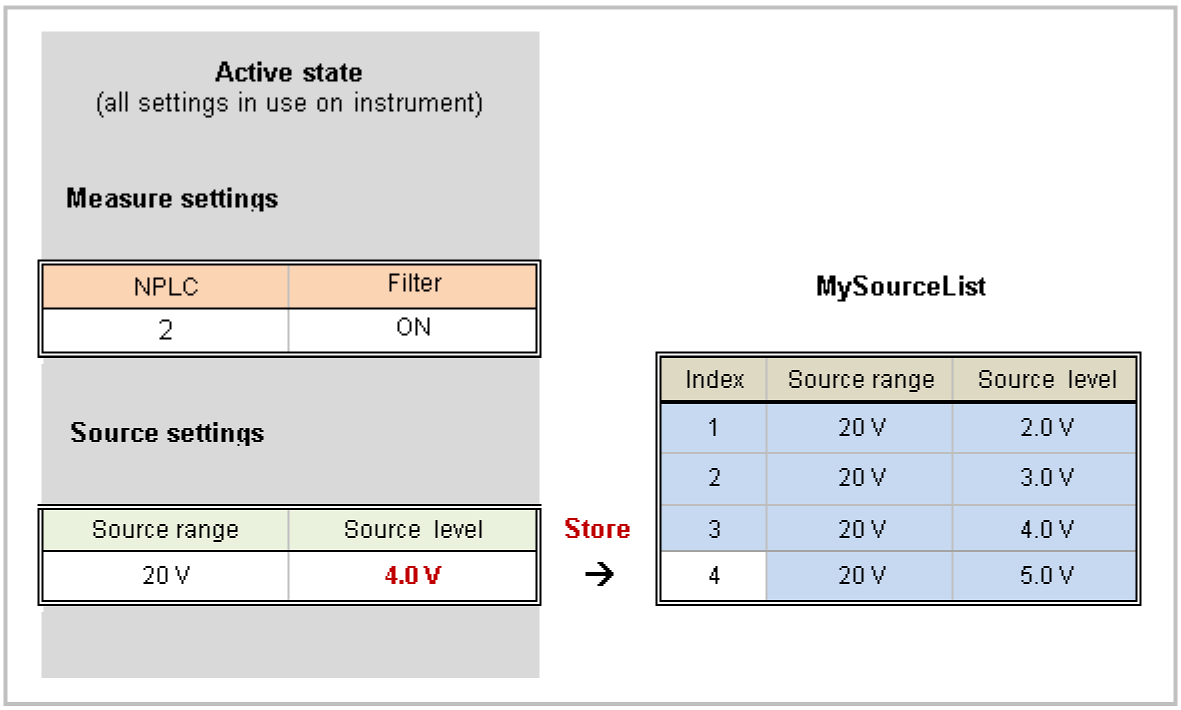
```
:SOURce:VOLTage:LEVel 5
:SOURce:CONF:LIST:STORe "MySourceList"
```

Using TSP commands:

```
smu.source.level = 5
smu.source.configlist.store("MySourceList")
```

The following figure shows the active state of the instrument after you change the source level to 5.0 V. Refer to Instrument settings stored in a source configuration list (on page 3-38) for a complete list of source settings that the instrument stores in a source configuration list.

Figure 85: Example configuration point 4



Recalling a configuration point

Use one of the following methods to recall configuration point 2 on `MySourceList`:

Using SCPI commands:

```
:SOURce:CONFiguration:LIST:RECall "MySourceList", 2
```

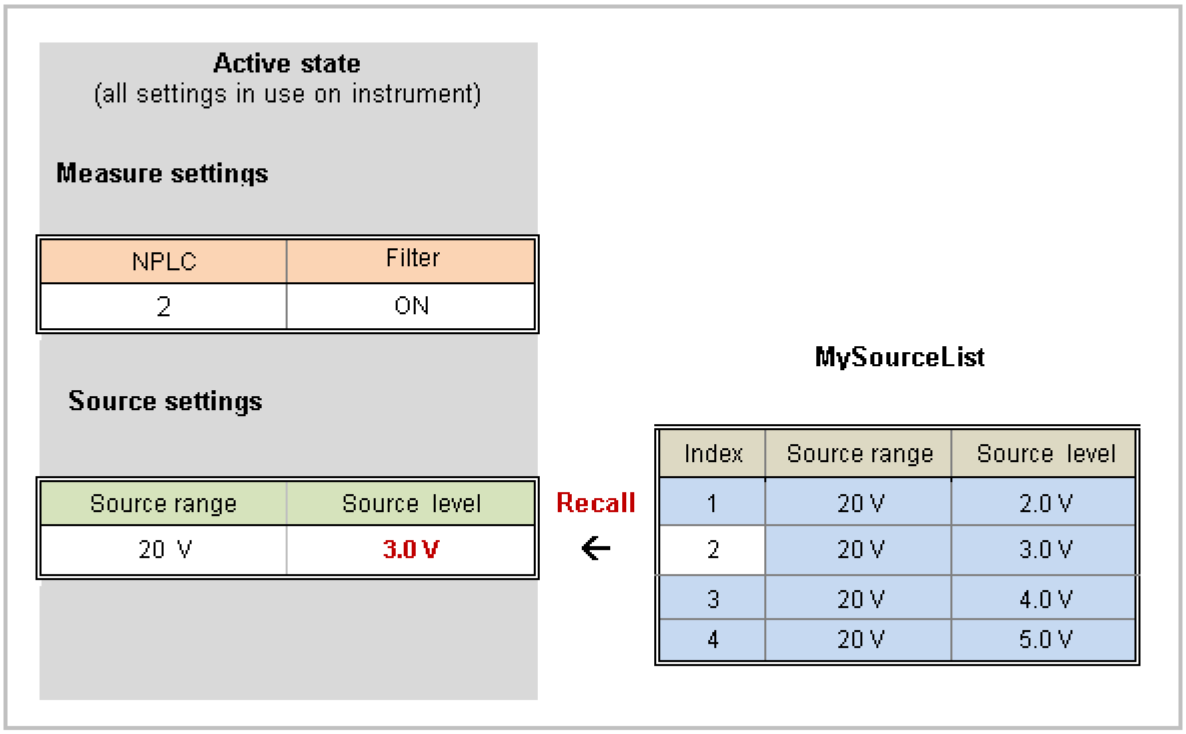
Using TSP commands:

```
smu.source.configlist.recall("MySourceList", 2)
```

The following figure shows the active state of the instrument after you recall configuration point 2.

Notice that the NPLC setting is 2.0. This is because when you recall a source configuration list, the settings that are restored are the source settings that were active at the time the point was stored. Since NPLC is not a source setting, recalling a source configuration list has no effect on the value of the NPLC setting.

Figure 86: Recall configuration point 2



Viewing configuration list contents

You can ask the instrument to display or print the contents of a specific configuration point by sending a query. The contents returned include all of the active settings that the instrument saved when you stored the configuration point.

Using SCPI commands:

The SCPI configuration list query command returns a list of TSP commands that could be used to set the parameters stored in the specified configuration point.

To view a list of commands in configuration point 3 in a source configuration list named `MyConfigList`, send the command:

```
:SOURce:CONFiguration:LIST:QUERY? "MyConfigList"
```

For a measure configuration list, replace `:SOURce` with `:SENSe`.

Using TSP commands:

The TSP configuration list query commands return a list of TSP commands that were used to set the settings stored in the specified configuration point.

To print a list of commands in configuration point 3 in a source configuration list named `MyConfigList`, send the command:

```
print(smu.source.configlist.query("MyConfigList", 3))
```

For a measure configuration list, replace `source` with `measure`.

Deleting a configuration list

This section describes how to delete a specific point on a configuration list, and how to delete an entire list.

Using SCPI commands to delete a specific configuration point or entire configuration list:

To delete configuration point 8 in a source configuration list named `MySourceList`, send the following command specifying the index.

```
:SOURce:CONFiguration:LIST:DELete "MySourceList", 8
```

To delete the entire source configuration list named `MySourceList`, send the following command:

```
:SOURce:CONFiguration:LIST:DELete "MySourceList"
```

For a measure configuration list, replace `:SOURce` with `:SENSe`.

Using TSP commands:

To delete configuration point 8 from a source configuration list named `MyConfigList`, send the command:

```
smu.source.configlist.delete("MyConfigList", 8)
```

For a measure configuration list, replace `source` with `measure`.

To delete an entire source configuration list named `MyConfigList`, send the command:

```
smu.source.configlist.delete("MyConfigList")
```

For a measure configuration list, replace `source` with `measure`.

Viewing the available configuration lists

You can use remote commands to view the names of the configuration lists stored on the instrument.

Using SCPI commands:

To receive the name of one source configuration list stored on the instrument, use the following command.

```
:SOURce:CONFigure:LIST:CATalog?
```

For a measure configuration list, replace `:SOURce` with `:SENSe`.

Each time this command executes, the name of one defined configuration is returned. Keep sending this command until it returns an empty string to get all defined lists. After the command returns an empty string, it wraps around and starts returning names again. If only an empty string is returned, no configuration lists of the specified type exist.

Using TSP commands:

To receive the name of one source configuration list stored on the instrument, use the following command.

```
print(smu.source.configlist.catalog())
```

For a measure configuration list, replace `source` with `measure`.

Each time this command executes, the name of one defined configuration is returned. Keep sending this command until it returns `nil` to get all defined lists. After the command returns `nil`, it wraps around and starts returning names again. If only `nil` is returned, no configuration lists of the specified type exist.

Determining the size of a configuration list

You can view the number of configuration points that are in a specific configuration list.

Using SCPI commands:

To view the number of configuration points in a source configuration list named `MyConfigList`, send the following command:

```
:SOURce:CONFiguration:LIST:SIZE? "MyConfigList"
```

For a measure configuration list, replace `:SOURce` with `:SENSe`.

Using TSP commands:

To view the number of configuration points in a source configuration list named `MyConfigList`, send the following command:

```
smu.source.configlist.size("MyConfigList")
```

For a measure configuration list, replace `source` with `measure`.

Saving a configuration list

Configuration lists are lost when you turn the instrument off and turn it on again. Save a configuration list by creating a configuration script (TSP or front panel) or using the `*SAV` and `*RCL` commands (SCPI). A configuration script saves the settings of the instrument, including all defined source and measure configuration lists. See Saving setups (on page 2-112) for additional information.

Sweep operation

Sweeps allow you to set up the instrument to source specific voltage or current values to a device under test (DUT). A measurement is made for each value.

The Model 2450 can generate linear staircase, logarithmic staircase, linear dual staircase, and logarithmic dual staircase sweeps from the front panel. In addition to these sweeps, you can generate custom sweeps if you use remote commands.

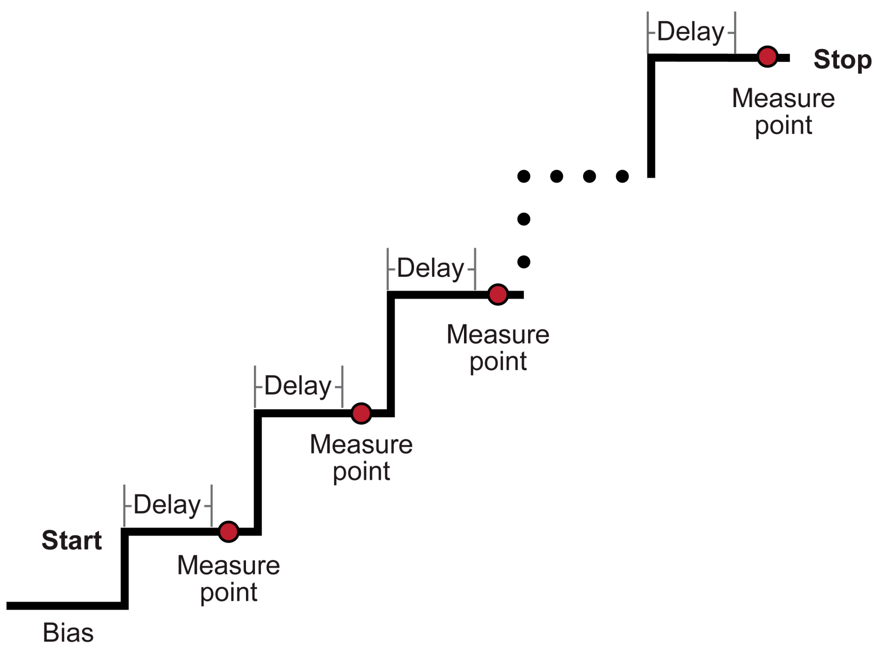
Linear staircase sweep

When you use a linear staircase sweep, the voltage or current source increases or decreases in fixed steps. Each source-measure point is equally spaced between the start and stop.

The sweep begins with a start voltage or current and ends with a stop voltage or current. A measurement is made at each point after the delay. The figure below shows an increasing linear staircase sweep.

When a linear staircase sweep is triggered to start, the output goes from the bias level to the start source level. The output then changes in equal steps until the stop source level is reached. A measurement is performed at each source step (including the start and stop levels). With trigger delay set to zero, the time duration at each step is determined by the source delay and the time it takes to perform the measurement (the NPLC setting). Note that the delay is the same for all steps in the sweep.

Figure 87: Model 2450 sweep linear staircase

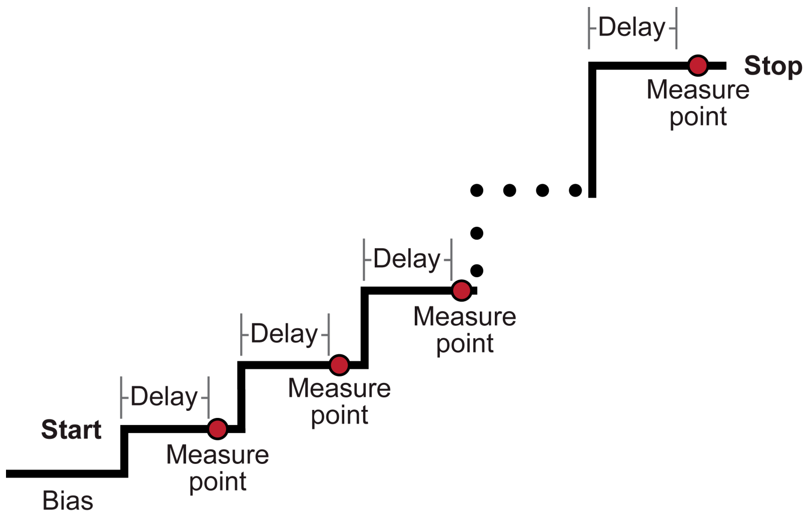


Logarithmic staircase sweep

A logarithmic staircase sweep is similar to a linear staircase sweep. The only difference is that the steps are scaled logarithmically.

The steps in a logarithmic staircase sweep increase or decrease geometrically, beginning with a start voltage or current and ending with a stop voltage or current. The figure below shows an increasing logarithmic staircase sweep.

Figure 88: Logarithmic staircase sweep



Setting up a sweep

NOTE

Defining and generating a sweep creates a new trigger model that will replace an existing trigger model. If you want to preserve the existing trigger model, save a user-saved setup. See Saving setups (on page 2-112) for information on saving an existing trigger model as part of a user-saved setup.

Before setting up the sweep, set up the instrument for the test you will run. Typical settings you can set for a sweep include:

- The source function
- The measure function
- Current or voltage limit
- Source readback
- Voltage protection limits
- 2-wire or 4-wire sense mode
- Front or rear terminal selection

NOTE

If you change settings after you set up a sweep, those changes will also affect the sweep the next time it is initiated.

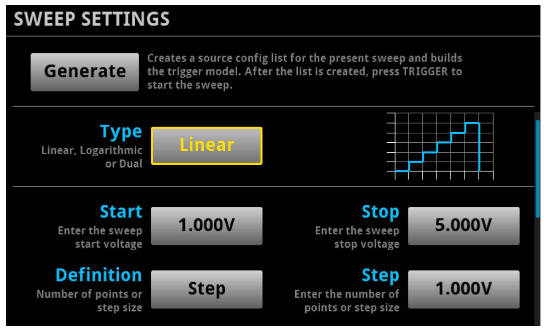
Setting up a sweep from the front panel

To set up a sweep from the front panel, you select options from the Sweep Settings screen.

Set up the sweep from the front panel

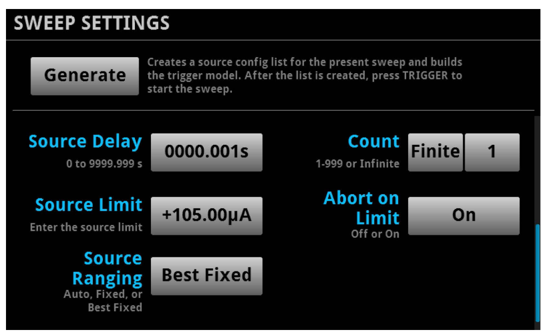
1. Select **FUNCTION** and select the source and measure functions.
2. On the home screen, set the Source value.
3. Press **Menu**.
4. Under Source, select **Sweep**. The Sweep Settings screen is displayed.

Figure 89: Sweep Settings screen - first page



5. Make selections appropriate to your sweep. See the table below for detail on the options.
6. Swipe down to see additional options.

Figure 90: Sweep Settings screen - second page



7. Make selections appropriate to your sweep. See the table below for detail on the options.
8. Select **Generate**.
9. To run the sweep, press the **TRIGGER** key.

Front-panel sweep options

Option	Description
Type	You can select one of the following options: • **Linear**: Sets up a linear staircase sweep. • **Logarithmic**: Sets up a logarithmic staircase sweep. • **Linear Dual**: Sets up a linear staircase sweep that runs from the start source value to the stop source value, then runs from the stop value to the start value. • **Log Dual**: Sets up a logarithmic staircase sweep that runs from the start source value to the stop source value, then runs from the stop value to the start value.
Start	The voltage or current source level at which the sweep starts: • Current: -1.05 to 1.05 • Voltage: -210 to 210 To set up an increasing sweep, set start level to be less than the stop level. To set up a decreasing sweep, set the start level to be more than the stop level.
Stop	The voltage or current at which the sweep stops: • Current: -1.05 to 1.05 • Voltage: -210 to 210
Definition	Determines if the sweeps is set up for a certain number of points or by a specific step size. Select one of the following options: • **Number of Points:** When this option is selected, the instrument calculates the number of source-measure points in the sweep using the following formula: Points = [(Stop - Start) / Step] + 1 • **Step Size:** When this option is selected, the source level changes in equal steps from the start level to the stop level. A measurement is performed at each source step (including the start and stop levels). To calculate the number of source-measure points in a sweep, use one of the following formulas. Linear sweep: $step = \frac{stop - start}{points - 1}$ Logarithmic sweep: $log\ step\ size = \frac{log10(stop) - log10(start)}{points - 1}$
Step	Displayed if the sweep definition is set to Step Size. Set the size that each step should be.
Points	Displayed if the sweep definition is set to Number of Points. Select the number of points that you want to measure in the sweep.
Source Delay	Sets the delay (settling time) for the source function.
Count	How many times the sweep should repeat. You can select one of the following options: • **Finite**: Set a specific number of times to repeat. • **Infinite**: The sweep will repeat until is it aborted.

Front-panel sweep options

Option	Description
Source Limit	Sets the source limit for measurements. The Model 2450 cannot source levels that exceed this limit.
Abort on Limit	Determines if the sweep is stopped immediately if a limit is exceeded. You can select one of the following options: • **ON**: Abort the sweep if a limit is exceeded. • **OFF**: Complete the sweep even if a limit is exceeded.
Source Ranging	The source range that is used for the sweep. You can select one of the following options: • **Best Fixed**: The instrument selects a single fixed source range that will accommodate all the source levels in the sweep. This avoids overshoots during sweeps. • **Auto:** The instrument selects the most sensitive source range for each source level in the sweep. • **Fixed**: The source remains on the range that is set when the sweep is started. If a sweep point exceeds the source range capability, the source will output the maximum level for that range.

Setting up a sweep using SCPI commands

To set up a sweep using SCPI commands, you send one of the following commands:

- :SOURce[1]:SWEep:<function>:LINear: Sets up a linear sweep for a set number of measurement points.
- :SOURce[1]:SWEep:<function>:LINear:STEP: Sets up a linear source sweep configuration list and trigger model with a fixed number of steps.
- :SOURce[1]:SWEep:<function>:LIST: Sets up a sweep based on a configuration list, which allows you to customize the sweep.
- :SOURce[1]:SWEep:<function>:LOG: Sets up a logarithmic sweep for a set number of measurement points.

To create a sweep:

1. Set the source function using :SOURce[1]:FUNCtion[:MODE].
2. Set the source range using :SOURce[1]:<function>:RANGe.
3. Set any other source settings that apply to your sweep. You must set source settings before the sweep function is called.
4. If you are using :SOURce[1]:SWEep:<function>:LIST, set up the source configuration list for your sweep.
5. Set the parameters for the sweep command.
6. Set the measurement function using [:SENSe[1]]:FUNCtion.
7. Set the measurement range using [:SENSe[1]]:<function>:RANGe[:UPPer].
8. Make any other settings appropriate to your sweep.
9. Send :INITiate to start the sweep.

Quick Tip

To save your settings, save them to a user-saved setup using the *SAV command.

For example sweeps, see Sweep programming examples (on page 3-58).

For detail on the commands and options listed above, see the following command descriptions:

- [:SENSe[1]]:FUNCtion[:ON] (on page 6-49)
- [:SENSe[1]]:<function>:RANGe[:UPPer] (on page 6-53)
- :SOURce[1]:FUNCtion[:MODE] (on page 6-72)
- :SOURce[1]:<function>:RANGe (on page 6-74)
- :SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)
- :SOURce[1]:CONFiguration:LIST:STORe (on page 6-65)
- :SOURce[1]:SWEep:<function>:LINear (on page 6-81)
- :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83)
- :SOURce[1]:SWEep:<function>:LIST (on page 6-85)
- :SOURce[1]:SWEep:<function>:LOG (on page 6-87)

Setting up a sweep using TSP commands

To set up a sweep using TSP commands, you send one of the following commands:

- `smu.source.sweeplinear()`: Sets up a linear sweep for a set number of measurement points.
- `smu.source.sweeplinearstep()`: Sets up a linear source sweep configuration list and trigger model with a fixed number of steps.
- `smu.source.sweeplist()`: Sets up a sweep based on a configuration list, which allows you to customize the sweep.
- `smu.source.sweeplog()`: Sets up a logarithmic sweep for a set number of measurement points.

To create a sweep:

1. Set the source function using `smu.source.func`.
2. Set the source range using `smu.source.range`.
3. Set any other source settings that apply to your sweep. You must set source settings before the sweep function is called.
4. If you are using `smu.source.sweeplist()`, set up the source configuration list for your sweep.
5. Set the parameters for the sweep command.
6. Set the measurement function using `smu.measure.func`.
7. Set the measurement range using `smu.measure.range`.
8. Make any other settings appropriate to your sweep.
9. Send `trigger.model.initiate()` to start the sweep.

Quick Tip

To save your settings, save them to a configuration script using the `createconfigscript()` command.

For example sweeps, see Sweep programming examples (on page 3-58).

For detail on the commands and options listed above, see the following command descriptions:

- smu.source.sweeplinear() (on page 8-145)
- smu.source.sweeplinearstep() (on page 8-147)
- smu.source.sweeplist() (on page 8-149)
- smu.source.sweeplog() (on page 8-151)
- smu.source.func (on page 8-134)
- smu.source.range (on page 8-143)
- smu.source.configlist.create() (on page 8-129)
- smu.source.configlist.store() (on page 8-132)
- smu.measure.func (on page 8-103)
- smu.measure.range (on page 8-117)

Aborting a sweep

Sweeps can be stopped for the following reasons:

- The limit set by the abort on limit setting was exceeded
- The trigger model is aborted

You can stop the sweep while it is in progress. When you stop the sweep, all sweep commands in the trigger model are terminated.

Using the front panel:

Press the front-panel **TRIGGER** key for two seconds and select **Trigger Model (Abort)**.

Using SCPI commands:

Send the command:

```
:ABORt
```

Using TSP commands:

Send the command:

```
trigger.model.abort()
```

Sweep programming examples

The following examples show programming examples of typical sweeps.

Linear sweep with a voltage source

The following examples perform a linear sweep that uses a voltage source. They perform the following actions:

- Reset the instrument to its defaults.
- Set the source function to voltage.
- Set the source range to 20 V.
- Set the measure function to current.
- Set the current range to 100 µA.
- Set up a linear sweep that sweeps from 0 to 10 volts in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. In TSP only, name the configuration list that is created for this sweep `VoltLinSweep`.
- Start the sweeps.

No buffer is defined, so the data is stored in `defbuffer1`. See Reading buffers (on page 3-10) for more information on reading buffers.

Using SCPI commands

```
*RST
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SENS:FUNC "CURR"
SENS:CURR:RANG 100e-6
SOUR:SWE:VOLT:LIN 0, 10, 20, 1e-3, 1, FIXED
INIT
```

Using TSP commands

```
reset()
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 100e-6
smu.source.sweeplinear("VoltLinSweep", 0, 10, 20, 1e-3, 1, smu.RANGE_FIXED)
trigger.model.initiate()
```

Logarithmic sweep with a current source

The following examples perform a logarithmic sweep using a current source. They perform the following actions:

- Reset the instrument to its defaults.
- Set the source function to current.
- Set the source range to 100 mA.
- Set up a logarithmic sweep from 100 μA to 100 mA in 10 steps with a source delay of 10 ms, a sweep count of 1, and a fixed source range. In TSP only, name the configuration list that is created for this sweep `CurrLogSweep`.
- Set the measure function to current.
- Set the current range to 100 μA.
- Start the sweep.

No buffer is defined, so the data is stored in `defbuffer1`. See Reading buffers (on page 3-10) for more information on reading buffers.

Using SCPI commands

```
*RST
SOUR:FUNC CURR
SOUR:CURR:RANG 100e-3
SENS:FUNC "VOLT"
SENS:VOLT:RANG 20
SOUR:SWE:VOLT:LOG 100e-6, 100e-3, 10, 10e-3, 1, FIXED
INIT
```

Using TSP commands

```
reset()
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 100e-3
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
smu.source.sweeplog("CurrLogSweep", 100e-6, 100e-3, 10, 10e-3, 1, smu.RANGE_FIXED)
trigger.model.initiate()
```

Voltage sweep based on a configuration list

The following TSP example shows a voltage sweep that is based on a configuration list. It performs the following actions:

- Reset the instrument to its defaults
- Create a source configuration list called `CurrListSweep`.
- Set the source function to current.
- Set the source current range to 100 mA.
- Set the source current level to 10 µA.
- Save the source settings to `CurrListSweep`.
- Set the source current level to 1 mA.
- Save the source settings to `CurrListSweep`.
- Set the source current level to 500 µA.
- Save the source settings to `CurrListSweep`.
- Set the source current level to 7 mA.
- Save the source settings to `CurrListSweep`.
- Set the source current level to 1 mA.
- Save the source settings to `CurrListSweep`.
- Set the source current level to 90 mA.
- Save the source settings to `CurrListSweep`.
- Set up a list sweep that uses the entries from the `CurrListSweep` configuration list and starts at index 1 of the list.
- Set a source delay of 1 ms.
- Start the sweep.

No buffer is defined, so the data is stored in `defbuffer1`. See Reading buffers (on page 3-10) for more information on reading buffers.

Using TSP commands

```
reset()
smu.source.configlist.create("CurrListSweep")
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 100e-3
smu.source.level = 1e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 10e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 5e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 7e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 11e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 9e-3
smu.source.configlist.store("CurrListSweep")
smu.source.sweeplist("CurrListSweep",  1, 0.001)
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
trigger.model.initiate()
```

Increasing the speed of sweeps

To increase the speed of sweeps:

- Reduce the NPLC.
- Turn autozero off. If autozero is on, the instrument takes new reference and zero values for every reading. This can slow down sweep operation. Be aware that if you disable autozero, measurements may drift and become erroneous. To minimize drift when autozero is disabled, use the autozero once feature. For more information on the autozero options, see Automatic reference measurements (on page 2-108).

Digital I/O

You can use the Model 2450 digital input/output with the trigger model or to control an external digital circuit, such as a device handler that is used to perform binning operations. To control or configure any of the six digital input/output lines, send commands to the Model 2450 over a remote interface.

To use the Model 2450 digital I/O in a trigger link system (TLINK), connect it using a Model 2450-TLINK Trigger Link Cable and configure the Model 2450 digital input and output lines.

For more information the trigger model, see Trigger model (on page 3-95).

Digital I/O port

The digital I/O port uses a standard female DB-9 connector, which is located on the rear panel of the Model 2450. You can connect to the Model 2450 digital I/O using a standard male DB-9 connector. The port provides a connection point to each of the six digital I/O lines and other connections as shown in the following table.

Figure 91: Model 2450 digital IO port

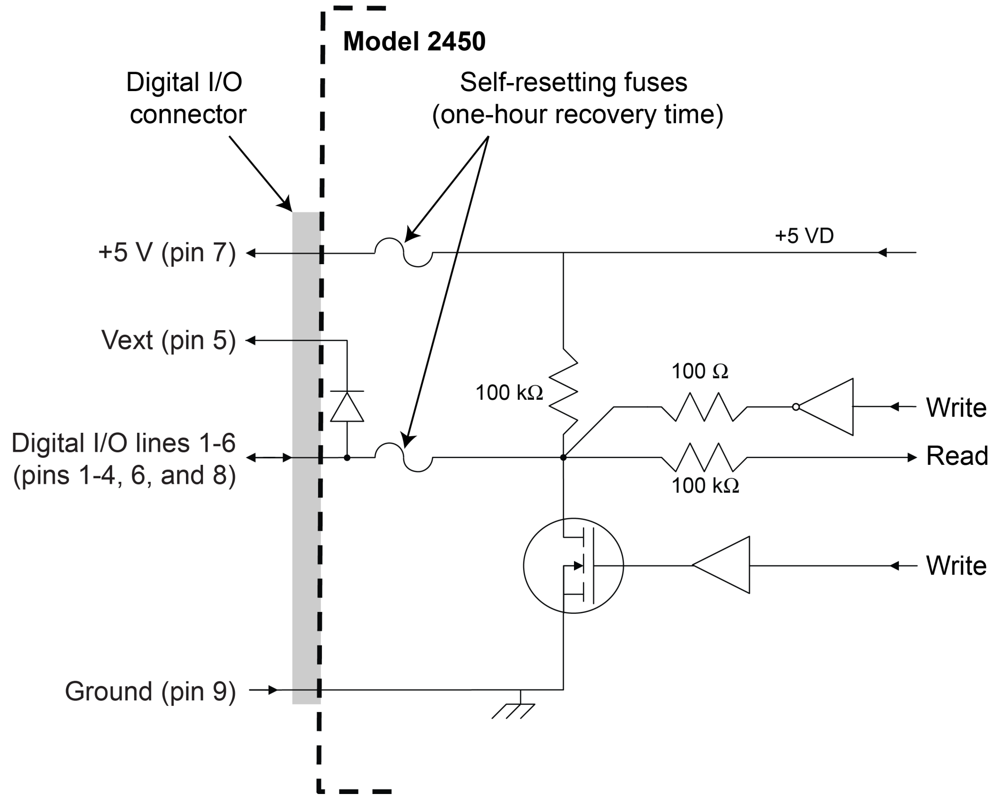
Model 2450 digital I/O port pinouts

Pin	Description
1	I/O line #1
2	I/O line #2
3	I/O line #3
4	I/O line #4
5	Vext line (relay flyback diode protection)
6	I/O line #5
7*	+5 V line
8	I/O line #6
9	Ground

* Use this pin to drive external logic circuitry. Maximum current output is 500 mA. This line is protected by a self-resetting fuse (one-hour recovery time).

The following figure shows the basic configuration of the digital I/O port. To set a line high (approximately +5 V), write a `1` to it; to set a line low (approximately 0 V), write a `0` to it. To allow an external device to pull an I/O line low by shorting it to ground, use the open-drain mode; an attached device must be able to sink at least 50 µA for each I/O line.

Figure 92: Digital I/O port configuration



Digital I/O lines

You can place each digital I/O line one of three modes: input, output, or open-drain.

NOTE

When you configure the digital I/O, configure the output line before the input line. This prevents false input trigger detection.

When you place a line in output mode, you can set the line as logic high (+5 V) or as logic low (0 V). The default level when the line is set to output is logic low (0 V). When you read the line while it is in this mode, it shows the present output status. It will not show external events.

When you place a line in input mode, each line is pulled up to +5 V while it detects external events. You cannot toggle the logic level while the line is in input mode. If you attempt to toggle the logic level, a warning message is generated.

When you place a line in open-drain mode, the line is configured to be an open-drain signal. This makes the line compatible with other instruments that use open-drain digital I/O lines, such as legacy Keithley Instruments products. You must write a `1` to any digital I/O line that is used as an input while in open-drain mode.

Each digital I/O line can be used for digital control or as a trigger line. You can control the line using either instrument commands or the trigger model. You can also configure the line for synchronous triggering using one of two synchronous modes. See Understanding synchronous triggering modes (on page 3-92) for more information.

Vext line

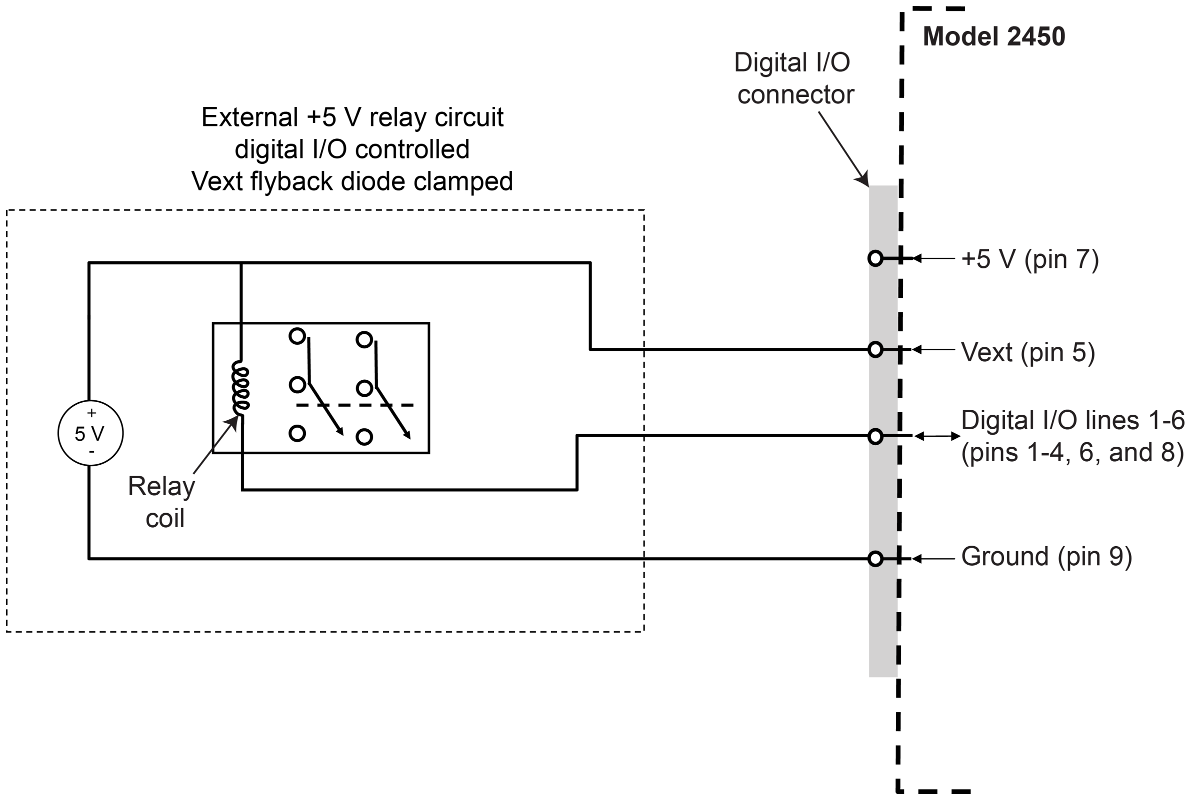
The digital I/O allows connection to an external voltage through a flyback diode clamped line (Vext). You can connect to the Vext line through the Vext pin (pin 5) and the specified digital I/O line. Use this connection to drive relay drive coils, a low power solenoid, or similar external inductive circuitry. The externally supplied voltage can be up to +5 V.

CAUTION

Do not apply more than 50 mA (maximum current) or exceed +5 V (maximum voltage) on the digital I/O port. Applying current or voltage exceeding these limits may damage the instrument.

Refer to the following figure for a simplified schematic of a sample digital I/O Vext relay circuit. You can externally power other devices by replacing the relay coil with the other device. When using the Vext pin to control externally powered devices, make sure to configure the corresponding digital output lines. In the low state (0 V), the output transistor sinks current through the external device. In the high state, the output transistor is off (transistor switch is open). This interrupts current flow through the external device. Most applications use an active-low (ON = 0 V) output sense.

Figure 93: Digital IO Vext (example external circuit)



+5 V line

The digital I/O port provides a +5 V output. You can use this line to drive external logic circuitry. The maximum current output for this line is 500 mA. A self-resetting fuse with a one-hour recovery time protects this line.

Controlling digital I/O lines

You must use a remote interface to set up and control the digital I/O lines. See Remote communication interfaces (on page 2-44) for information on setting up a remote interface and choosing a command set.

You can control each digital I/O line using the trigger model or directly as a digital line that is not controlled by the trigger model. Use any of the Model 2450 digital I/O lines in one of three modes: input, output, or open-drain. When a line is used in an open-drain mode, you must write a `1` to any digital I/O line that is used as an input. By default, each line is configured as a digital line in input mode. To change the configuration, either send the :DIGital:LINE<n>:MODE (on page 6-19) command (SCPI) or digio.line[N].mode (on page 8-43) (TSP).

Remote digital I/O commands

Commands for both SCPI and TSP are summarized in the following table. You can use the digital I/O port to do the following actions:

- Trigger the Model 2450 when external trigger pulses are applied to the digital I/O port
- Provide trigger pulses to external devices
- 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.

SCPI command	TSP command
:DIGital:LINE<n>:MODE (on page 6-19)	digio.line[N].mode (on page 8-43)
A line reset is not available; however, the line is reset when the a global reset (`*RST`) is sent	digio.line[N].reset() (on page 8-44)
:DIGital:LINE<n>:STATe (on page 6-20)	digio.line[N].state (on page 8-45)
:DIGital:READ? (on page 6-21)	digio.readport() (on page 8-45)
:DIGital:WRITe <n> (on page 6-21)	digio.writeport() (on page 8-46)
:TRIGger:DIGital<n>:IN:CLEar (on page 6-149)	trigger.digin[N].clear() (on page 8-170)
:TRIGger:DIGital<n>:IN:EDGE (on page 6-150)	trigger.digin[N].edge (on page 8-171)
:TRIGger:DIGital<n>:IN:OVERrun? (on page 6-151)	trigger.digin[N].overrun (on page 8-172)
Not available	trigger.digin[N].wait() (on page 8-172)
Not available	trigger.digout[N].assert() (on page 8-173)
:TRIGger:DIGital<n>:OUT:LOGic (on page 6-151)	trigger.digout[N].logic (on page 8-174)
:TRIGger:DIGital<n>:OUT:PULSewidth (on page 6-152)	trigger.digout[N].pulsewidth (on page 8-174)
Not available	trigger.digout[N].release() (on page 8-175)
:TRIGger:DIGital<n>:OUT:STIMulus (on page 6-153)	trigger.digout[N].stimulus (on page 8-175)

NOTE

One of the trigger model blocks you can configure is the Notify block. You can configure this block as a stimulus to a digital I/O line. For additional information on the Notify block, see Using the notify block event (on page 3-111).

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 value.

Using SCPI commands:

```
:DIGital:LINE1:MODE DIGital, OUT
:DIGital:LINE1:STATe 1
:DIGital:LINE1:MODE DIGital, IN
:DIGital:LINE1:STATe?
```

Using TSP commands:

```
-- Set bit B1 high
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[1].state = digio.STATE_HIGH

-- Read digital I/O port
digio.line[1].mode = digio.MODE_DIGITAL_IN
print(digio.line[1].state)
```

Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in the following table.

Line #	Bit	Pin	Decimal	Hexadecimal	Binary
1	B1	1	1	0x01	000001
2	B2	2	2	0x02	000010
3	B3	3	4	0x04	000100
4	B4	4	8	0x08	001000
5	B5	6	16	0x10	010000
6	B6	8	32	0x20	100000

Limit testing and binning

Limit testing allows you to set high and low limit values. When the reading falls outside these limits, the instrument beeps and displays the "HI" or "LO" message.

The limit values are stored in volatile memory. The values are reset to zero when the instrument is turned off. Limits can be used with all measurements (except continuity and diode measurements).

The Model 2450 can be set up for limit testing and binning. It can perform simple benchtop limit testing using the front panel or sophisticated limit and binning operations using the trigger model and digital I/O to control external component handling devices.

Some typical forms of limit testing include:

- Simple pass or fail testing
- Resistor grading: Inspect multiple limits until the first failure is received
- Resistor sorting: Inspect multiple limits until the first pass is received
- Complex testing, where multiple tests must be inspected, for applications like diode testing; in this case you might do the following:
  - Test only until you fail one of the tests; immediate binning
  - Test all tests; bin on first failure, end binning

For binning applications, you use limit testing to determine placement of tested parts. To set up the instrument to place the part in the correct bin, you do the following steps:

- Determine and record a bin number for later use
- Output a digital bit pattern to physically place the tested device in a bin
- If multiple tests are performed on the same part, determine when the part should be binned:
  - Bin the part as soon as it fails a test
  - Bin the part after all parameters are measured; bin according to the first failure or a combination of failures

Limit testing using the front-panel interface

You can do pass or fail limit testing through the front panel. When limit testing and a test fails, the limit (1 or 2) that failed is shown on the Home screen.

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Settings**.
3. Next to Limits, select **View**.
4. To enable limit testing, set the state to **On**.
5. The Auto Clear setting automatically clears the limit fail indicator when a new passing measurement is made. To turn this feature off, select **Off**.
6. Set the Low Value. If the measurement is below the Low Value, the limit failure indicator is displayed.
7. Set the High Value. If the measurement is above the High Value, the limit failure indicator is displayed.
8. Select **HOME** to return to the operating display.

An example of using limit testing to check resistors is described in the following topics.

Front-panel limit test

This example is set up to test a box of 100 Ω ±1 % and 100 Ω ±10 % resistors that you want to separate manually. You can change values as needed to adapt the test to your needs.

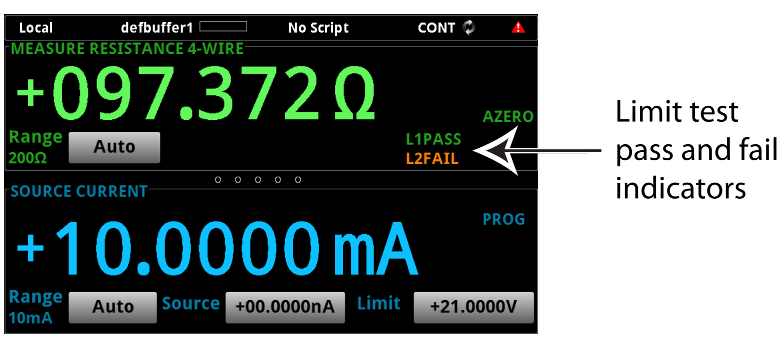
Set up the test:

1. Press the **FUNCTION** key.
2. Under Source Current and Measure, select **Resistance**.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. Next to Limits, select **View**.
6. To enable limit testing, set the state to **On** for both Limit 1 and Limit 2.
7. Leave the Auto Clear setting at the default of **On** for both limits.
8. For Limit 1, set the Low Value to **90 Ω**.
9. For Limit 1, set the High Value to **110 Ω**.
10. For Limit 2, set the Low Value to **99 Ω**.
11. For Limit 2, set the High Value to **101 Ω**.
12. Select **OK**.
13. On the MEASUREMENT SETTINGS screen, set the Sense Mode to **4-Wire**. Leave other settings at the default values.

Run the test:

1. Press the **HOME** key.
2. Use 4-wire connections to connect the first resistor to the instrument.
3. Press **Output ON/OFF** to turn the source on.
4. Verify that the instrument is set to Continuous Measurement. If necessary, hold the **TRIGGER** key for 3 seconds and select **Continuous Measurement**.
5. Observe the measurements. If the resistor is inside the limits set for Limit 1, L1PASS is displayed. If the resistor is not within the limits, L1FAIL is displayed. If the resistor is in the limits set for Limit 2, L2PASS is displayed. If the resistor is not within the limits, L2FAIL is displayed. An example of a test that passed the L1 test but failed the L2 test is shown below.
6. Press **Output ON/OFF** to turn the source off. Note that the limit indicators are displayed until you turn limit testing off.

Figure 94: Limit test front panel indicators



Set up a limit test using the remote interface

You can set up limit testing through a remote interface. There are several methods you can use to set up the limit test:

- Scripting (available with TSP only): Allows the most flexibility; you can set up the limit test as needed.
- Trigger model: Provides the best speed and throughput, using pre-defined trigger blocks to simplify set up.

The following topics show examples using simple and complex limit and binning tests.

Resistor grading using limit testing

This limit test inspects multiple limits until the first failure is received. When a resistor fails, it is sorted into the appropriate bin.

This example grades resistors into tolerance levels (for example, 20%, 10%, 5% and 1%). A single spot measurement is inspected against multiple limits, which tighten progressively around the same nominal value. Since there is no reason to continue limit checking once the appropriate tolerance level for a resistor-under-test is determined, this application will typically immediately bin the tested resistors. The bit patterns assigned to the limits determine into which bin a resistor is placed.

For this example, the same fail bit pattern is assigned to both the lower and upper bounds of the limits so that resistors with resistance values in the range R-P% to R go into the same bin as those with resistance values in the range R to R+P%. If you want to put parts in separate bins that correspond to R-P% to R and R to R+P%, you can do so by assigning different bit patterns for the upper and lower bounds of the limits.

Since the limits are inspected in ascending numeric order, the measured resistance is checked first against Limit 2, which is the 20% limit. If a resistor fails this limit inspection, its resistance value is outside of the 20% tolerance band and it is considered to be a bad part. The trigger model outputs the Limit 2 fail bit pattern, which causes the component handler to place the resistor in the Bad Part bin.

If a resistor passes the 20% limit test, the resistance value is checked against the 10% limit value. If the resistor fails this limit inspection, the resistance is outside of the 10% tolerance band, but in the 20% band. The trigger model outputs the Limit 3 fail bit pattern, which causes the component handler to place the resistor in the 20% tolerance part bin.

If a resistor passes the 10% limit test, the resistance value is checked against the 5% limit value, and so on. If a resistor passes all the limit tests, the trigger model outputs the overall pass bit pattern, which causes the component handler to place the resistor in the 1% tolerance part bin.

Resistor grading example

Figure 95: Resistor grading example blocks 1 to 6

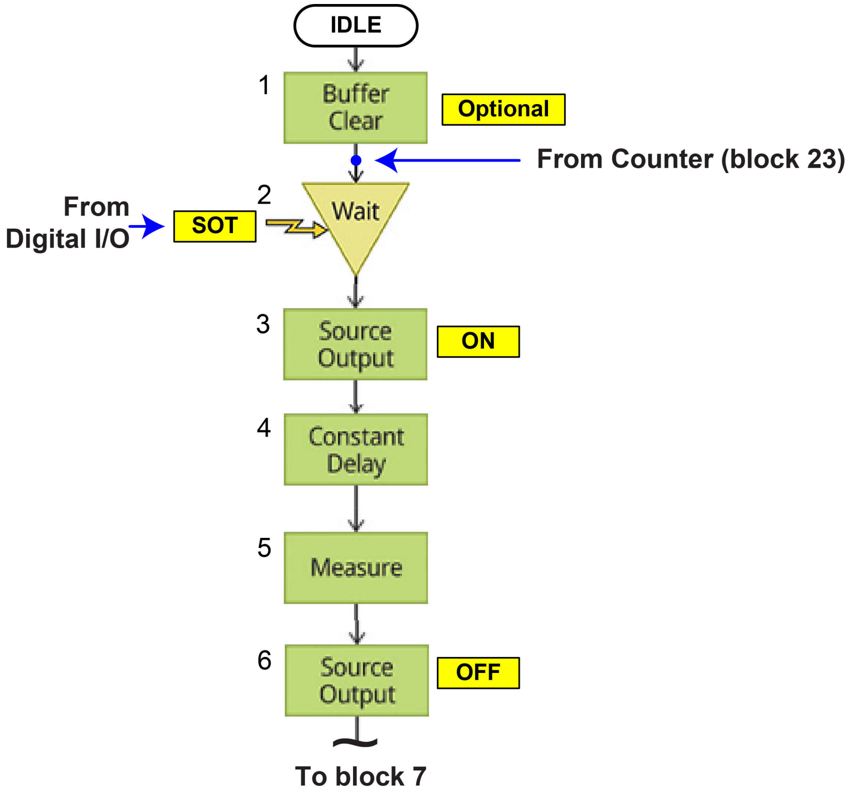


Figure 96: Resistor grading example blocks 7 to 18

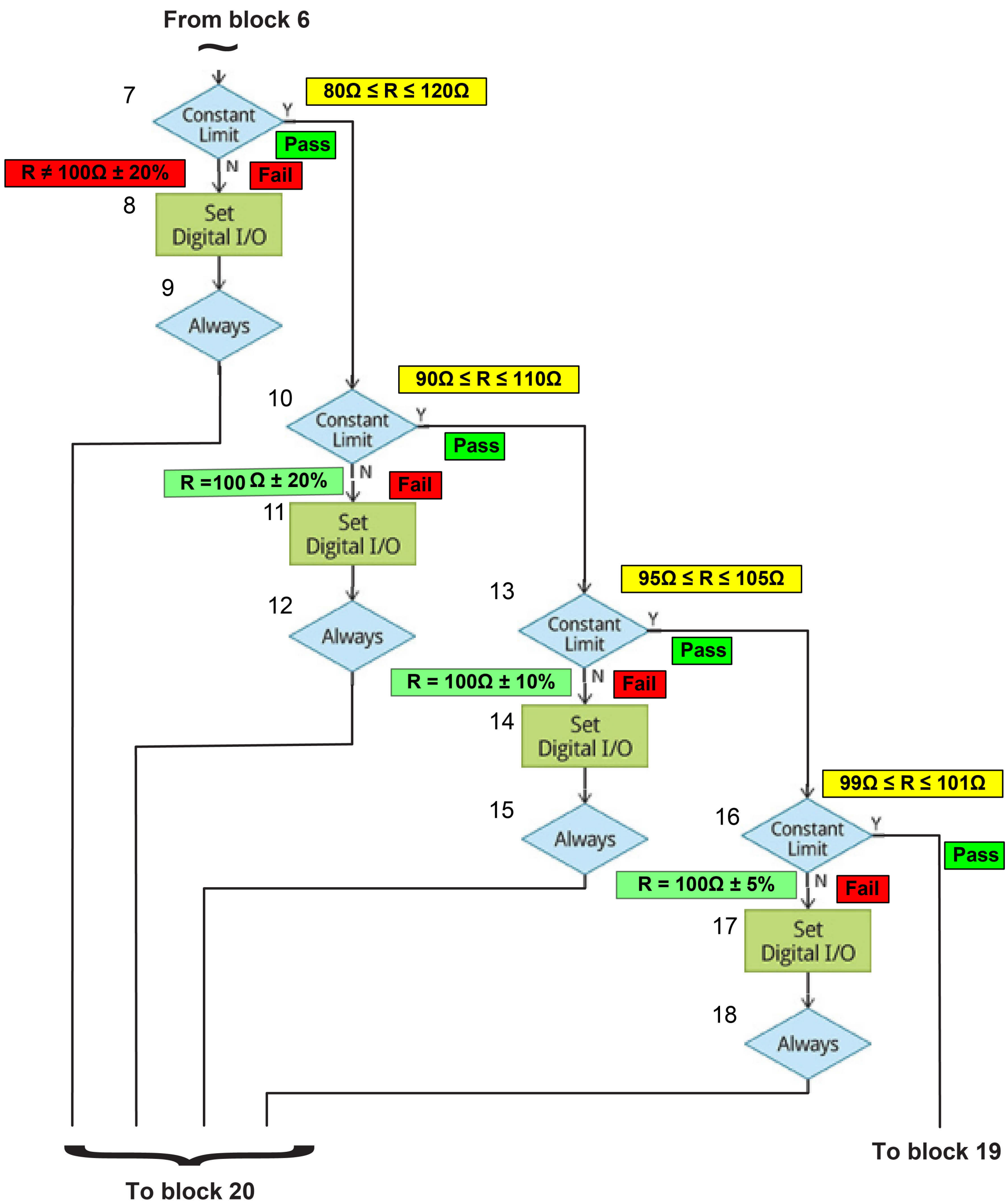
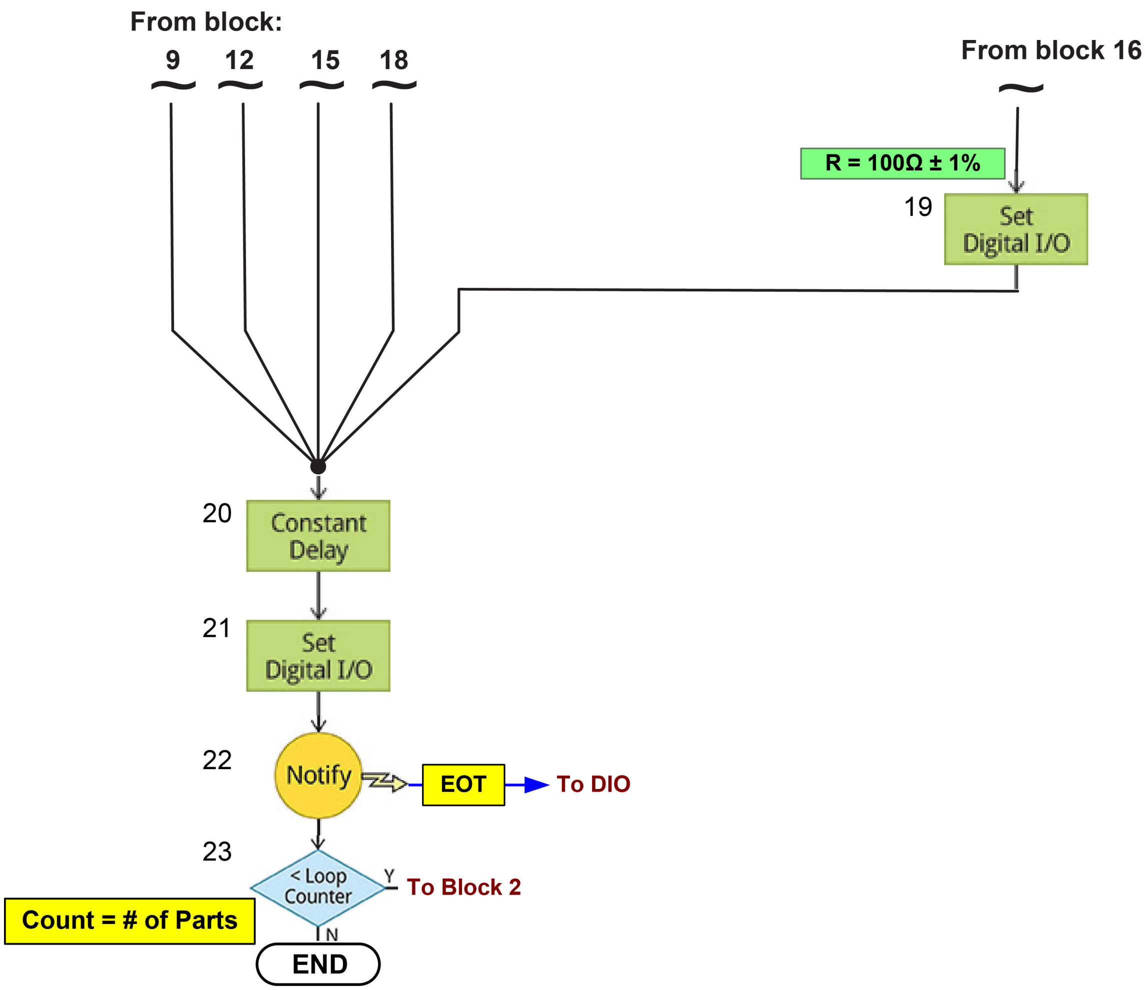


Figure 97: Resistor grading example blocks 19 to 23



Resistor grading SCPI code

Send the following commands for this example application:

Command	Description
`*RST`	Reset the Model 2450.

`SENSe:FUNCtion "RESistance"`
`SENSe:RESistance:MODE MANual`
`SENSe:RESistance:RSENse ON`
	Set the instrument measure resistance in manual ohms mode and set 4-wire remote sensing on.

`SENSe:NPLC 1`	Set the NPLCs to 1.
`SENSe:AZER:ONCE`	Have the instrument autozero once.

`SOURce:CURRent:RANGe 0.01`
`SOUR:CURR 0.01`
`SOUR:CURR:VLIM 2`
`SOUR:CURR:READ:BACK ON`
	Set the instrument to source current with a range of 10 mA and a voltage limit of 2 V. Turn source readback on.

`TRACe:POINts 10`	Set the reading buffer size to 10.

`DIGital:LINE1:MODE DIG, OUT`
`DIG:LINE2:MODE DIG, OUT`
`DIG:LINE3:MODE DIG, OUT`
`DIG:LINE4:MODE DIG, OUT`
	Set the digital I/O lines 1 to 4 to be digital lines that detect rising-edge or falling-edge triggers as input.

`DIG:LINE5:MODE TRIG, IN` `TRIGger:DIG5:IN:EDGE FALL`	Set digital I/O line 5 for trigger model control, detecting falling-edge triggers as input.

`DIG:LINE6:MODE TRIG, OUT`
`TRIGger:DIGital6:OUT:LOGic NEG`
`TRIG:DIG6:OUT:PULSewidth 10e-6`
`TRIG:DIG6:OUT:STIMulus NOT1`
	Set digital I/O line 6 for trigger model control, detecting rising-edge or falling-edge triggers as input. Set the output trigger logic of the trigger event generator to negative. Set the length of time that the trigger line is asserted to 10e-6. Set the stimulus to create a notify event.

`TRIGger:LOAD:EMPTy`	Clear any existing trigger model commands from the instrument.

`TRIGger:BLOCk:BUFFer:CLE 1, "defbuffer1"`
`TRIG:BLOC:WAIT 2, DIG5`
`TRIG:BLOC:SOUR:STAT 3, ON`
`TRIG:BLOC:DEL:CONS 4, 0.001`
`TRIG:BLOC:MEAS 5, "defbuffer1"`
`TRIG:BLOC:SOUR:STAT 6, OFF`
`TRIG:BLOC:BRAN:LIM:CONS 7, IN, 80, 120, 10, 5`
`TRIG:BLOC:DIG:IO 8, 15, 15`
`TRIG:BLOC:BRAN:ALW 9, 20`
`TRIG:BLOC:BRAN:LIM:CONS 10, IN, 90, 110, 13, 5`
`TRIG:BLOC:DIG:IO 11, 1, 15`
`TRIG:BLOC:BRAN:ALW 12, 20`
`TRIG:BLOC:BRAN:LIM:CONS 13, IN, 95, 105, 16, 5`
`TRIG:BLOC:DIG:IO 14, 2, 15`
`TRIG:BLOC:BRAN:ALW 15, 20`
`TRIG:BLOC:BRAN:LIM:CONS 16, IN, 99, 101, 19, 5`
`TRIG:BLOC:DIG:IO 17, 3, 15`
`TRIG:BLOC:BRAN:ALW 18, 20`
`TRIG:BLOC:DIG:IO 19, 4, 15`
`TRIG:BLOC:DEL:CONS 20, 0.001`
`TRIG:BLOC:DIG:IO 21, 0, 15`
`TRIG:BLOC:NOT 22, 1`
`TRIG:BLOC:BRAN:COUN 23, 10, 2`
	Set up the trigger model:
	• Block 1: Clear default buffer 1.
	• Block 2: Set up a wait block to wait for digital line 5.
	• Block 3: Turn the source output on.
	• Bock 4: Set a constant delay of .001 seconds.
	• Block 5: Make a measurement and store it in default buffer 1.
	• Block 6: Turn the output off.
	• Block 7: Set up the constant limits to perform the first test.
	• Block 8: If block 7 fails, drive the digital I/O lines high.
	• Blocks 9, 12, 15, and 18: Branch to block 20.
	• Block 10: Set up the constant limits to perform the second test.
	• Block 11: If block 10 fails, drive line 1 high.
	• Block 13: Set up the constant limits to perform the third test.
	• Block 14: If block 13 fails, drive line 2 high.
	• Block 16: Set up the constant limits to perform the fourth test.
	• Block 17: If block 10 fails, drive lines 1 and 2 high.
	• Block 19: If block 10 fails, drive line 3 high.
	• Block 20: Delay for 1 ms
	• Block 21: Set all digital lines low.
	• Block 22: Send out a notify event to indicate that the test has ended.
	• Block 23: Return to block 2 23 times.

Resistor grading TSP code

```
local number_of_resistors = 100
-- Reset instrument to default settings
reset()
-- Measure function must be first measure setting;
-- most other settings are tied to the function
smu.measure.func = smu.DC_VOLTAGE
smu.measure.unit = smu.UNIT_OHM
-- Use 4-wire or "remote" voltage sensing
smu.measure.sense = smu.SENSE_4WIRE
-- Measure the actual value of the source for higher accuracy
smu.source.readback = smu.ON
-- Set measurement integration time to 1PLC (16.67ms at 60Hz)
-- Decrease to reduce test time; trade off accuracy for speed
smu.measure.nplc = 1
-- Immediately update autozero reference measurements and then disable autozero
   function
smu.measure.autozero.once()
```

```
-- Source settings must come after MANUAL ohms is enabled
-- Source function must be first source setting;
-- most other settings are tied to the function
smu.source.func = smu.FUNC_DC_CURRENT
-- This is actually a voltage range because of manual ohms mode
-- Set this after setting source function to current
smu.measure.range = 2
-- This is a current range
smu.source.range = 0.01
-- Set source level to 10 mA
smu.source.level = 0.01
-- Set voltage limit of current source to 2V; set this after setting measure range
smu.source.vlimit.level = 2
-- This example records the resistance measurements for later statistical analysis.
-- Limit inspection and binning can be performed without recording the
   measurements.
-- Set the buffer capacity equal to the number of resistors to be tested
defbuffer1.capacity = number_of_resistors
-- Configure digital I/O lines 1 through 4 as digital outputs
-- These I/O lines are used to output binning code to component handler
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[2].mode = digio.MODE_DIGITAL_OUT
digio.line[3].mode = digio.MODE_DIGITAL_OUT
digio.line[4].mode = digio.MODE_DIGITAL_OUT

-- Configure digital I/0 line 5 as a trigger input
-- Used to detect start-of-test trigger from component handler
digio.line[5].mode = digio.MODE_TRIGGER_IN
-- Set trigger detector to detect falling edge
trigger.digin[5].edge = trigger.EDGE_FALLING

-- Configure digital I/0 line 6 as a trigger output
-- Used to send end-of-test trigger to component handler
digio.line[6].mode = digio.MODE_TRIGGER_OUT
-- Output a falling edge trigger
trigger.digout[6].logic = trigger.LOGIC_NEGATIVE
-- Set width of output trigger pulse to 10 us
trigger.digout[6].pulsewidth = 10E-6
-- Trigger pulse will be output when Notify Block generates an event
trigger.digout[6].stimulus = trigger.EVENT_NOTIFY1
-- Reset existing trigger model settings.
trigger.model.load("EMPTY")
-- Configure the Trigger Model
-- Block 1: Clear defbuffer1
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
-- Block 2: Wait for start-of-test trigger on digital I/O line 5
trigger.model.setblock(2, trigger.BLOCK_WAIT, trigger.EVENT_DIGIO5)
-- Block 3: Turn SMU output ON
trigger.model.setblock(3, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
-- Block 4: Delay for 1ms to allow source to settle; adjust as appropriate
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 5: Measure resistance and store result in defbuffer1
trigger.model.setblock(5, trigger.BLOCK_MEASURE, defbuffer1)
```

```
-- Block 6: Turn SMU output OFF
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Block 7: Check if 80<=R<=120; if yes, go to Block 10
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 80, 120, 10, 5)
-- Block 8: Set digital I/O lines 1-4; output decimal 15 (binary 1111) to component
   handler
trigger.model.setblock(8, trigger.BLOCK_DIGITAL_IO, 15, 15)
-- Block 9: Go to Block 20
trigger.model.setblock(9, trigger.BLOCK_BRANCH_ALWAYS, 20)
-- Block 10: Check if 90<=R<=110; if yes, go to Block 13
trigger.model.setblock(10, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 90, 110, 13, 5)
-- Block 11: Set digital I/O lines 1-4; output decimal 1 (binary 0001) to component
   handler
trigger.model.setblock(11, trigger.BLOCK_DIGITAL_IO, 1, 15)
-- Block 12: Go to Block 20
trigger.model.setblock(12, trigger.BLOCK_BRANCH_ALWAYS, 20)
-- Block 13: Check if 95<=R<=105; if yes, go to Block 16
trigger.model.setblock(13, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 95, 105, 16, 5)
-- Block 14: Set digital I/O lines 1-4; output decimal 2 (binary 0010) to component
   handler
trigger.model.setblock(14, trigger.BLOCK_DIGITAL_IO, 2, 15)
-- Block 15: Go to Block 20
trigger.model.setblock(15, trigger.BLOCK_BRANCH_ALWAYS, 20)

-- Block 16: Check if 99<=R<=101; if yes, go to Block 19
trigger.model.setblock(16, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 99, 101, 19, 5)
-- Block 17: Set digital I/O lines 1-4; output decimal 3 (binary 0011) to component
   handler
trigger.model.setblock(17, trigger.BLOCK_DIGITAL_IO, 3, 15)
-- Block 18: Go to Block 20
trigger.model.setblock(18, trigger.BLOCK_BRANCH_ALWAYS, 20)
-- Block 19: Set digital I/O lines 1-4; output decimal 4 (binary 0100) to component
   handler
trigger.model.setblock(19, trigger.BLOCK_DIGITAL_IO, 4, 15)
-- Block 20: Delay 1ms; controls duration of digital bit patterns; adjust as
   appropriate
trigger.model.setblock(20, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 21: Set digital I/O lines 1-4; output decimal 0 (binary 0000) clear
   pattern to component handler
trigger.model.setblock(21, trigger.BLOCK_DIGITAL_IO, 0, 15)
-- Block 22: Notify block generates event, which causes output of a trigger pulse
   on digital I/O line 6
trigger.model.setblock(22, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
-- Block 23: Loop back to Block 2; keep looping until all resitors have been tested
trigger.model.setblock(23, trigger.BLOCK_BRANCH_COUNTER, number_of_resistors, 2)

-- After executing all of the above commands the trigger model can be initiated by
   executing "trigger.model.initiate()"
```

Resistor sorting using limit testing with multiple limits

This example inspects multiple resistors until it detects the first pass. This example uses a trigger model using constant limits.

This trigger model provides support for inspecting the output of a single test against multiple limits. The trigger model count block determines the number of devices that will be tested. A test refers to a single source or measure operation. After a measurement for a particular test is performed, it is checked against any enabled limits. The software limits are inspected in ascending numeric order until the first pass is detected. When a pass condition is detected, the pass bit pattern assigned to the limit that passed is output to the digital I/O port, and any subsequent limit inspections are aborted. If all limit inspections for the test fail, the overall fail bit pattern is output.

Figure 98: Resistor sorting example blocks 1 to 6

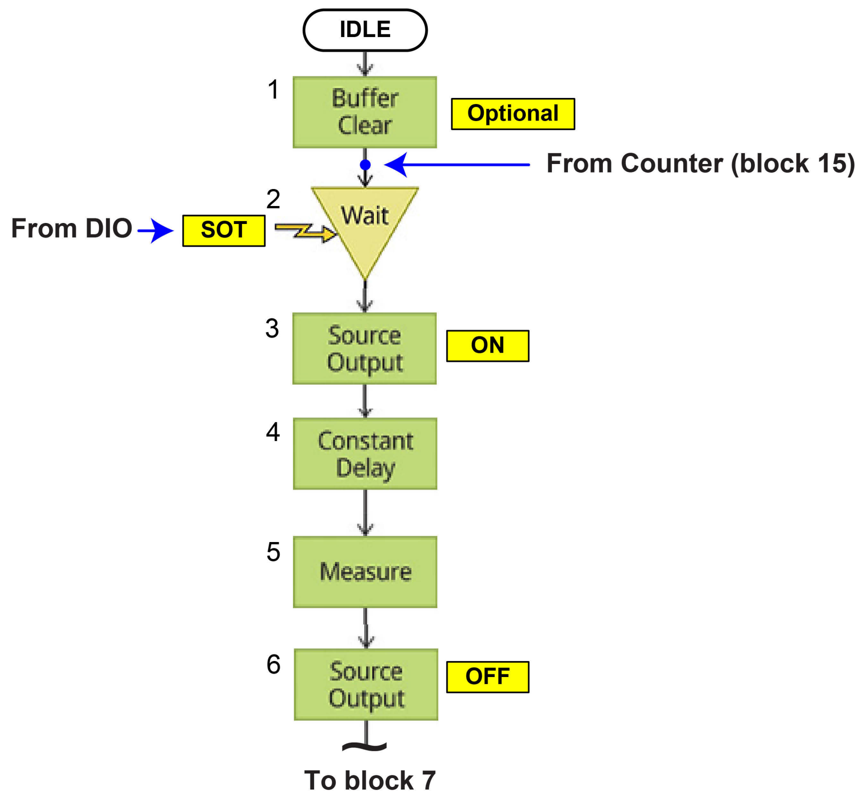
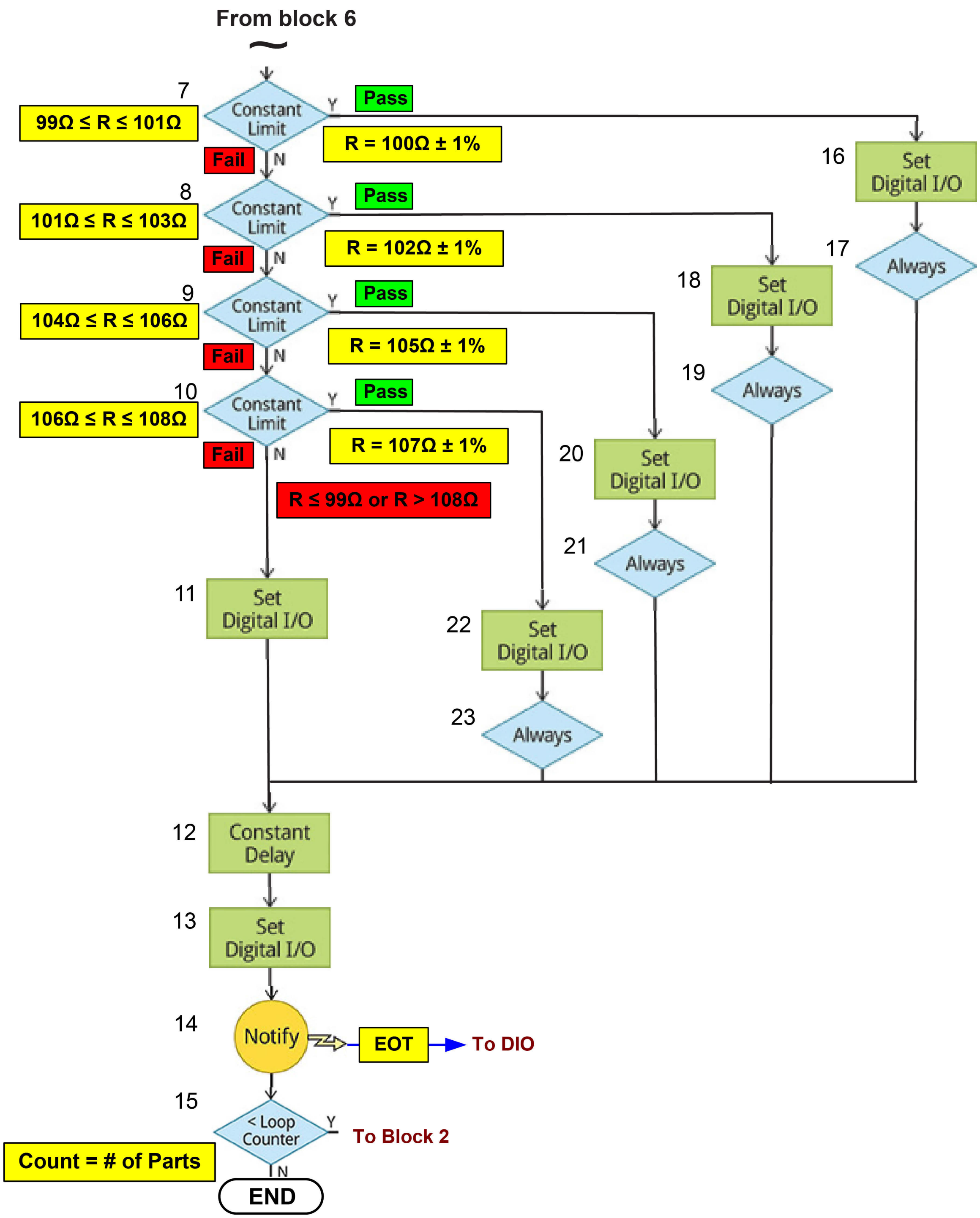


Figure 99: Resistor sorting trigger model blocks 6 to 23



Resistor sorting SCPI code

```
*RST

SYST:RSEN ON
SENS:FUNC "RES"
SENS:RES:MODE MAN
SENS:NPLC 1
SENS:AZER:ONCE
SENS:VOLT:RANG 2

SOUR:FUNC CURR
SOUR:CURR:READ:BACK ON
SOUR:CURR:RANG 0.01
SOUR:CURR 0.01
SOUR:CURR:VLIM 2

TRAC:POIN 10

DIG:LINE1:MODE DIG, OUT
DIG:LINE2:MODE DIG, OUT
DIG:LINE3:MODE DIG, OUT
DIG:LINE4:MODE DIG, OUT

DIG:LINE5:MODE TRIG, IN
TRIG:DIG5:IN:EDGE FALL

DIG:LINE6:MODE TRIG, OUT
TRIG:DIG6:OUT:LOG NEG
TRIG:DIG6:OUT:PULS 10e-6
TRIG:DIG6:OUT:STIM NOT1

TRIG:LOAD:EMPT

TRIG:BLOC:BUFF:CLE 1, "defbuffer1"
TRIG:BLOC:WAIT 2, DIG5
TRIG:BLOC:SOUR:STAT 3, ON
TRIG:BLOC:DEL:CONS 4, 0.001
TRIG:BLOC:MEAS 5, "defbuffer1"
TRIG:BLOC:SOUR:STAT 6, OFF
TRIG:BLOC:BRAN:LIM:CONS 7, IN, 80, 90, 16, 5
TRIG:BLOC:BRAN:LIM:CONS 8, IN, 90, 100, 18, 5
TRIG:BLOC:BRAN:LIM:CONS 9, IN, 100, 110, 20, 5
TRIG:BLOC:BRAN:LIM:CONS 10, IN, 110, 120, 22, 5
TRIG:BLOC:DIG:IO 11, 15, 15
TRIG:BLOC:DEL:CONS 12, 0.001
TRIG:BLOC:DIG:IO 13, 0, 15
TRIG:BLOC:NOT 14, 1
TRIG:BLOC:BRAN:COUN 15, 10, 2
TRIG:BLOC:DIG:IO 16, 1, 15
TRIG:BLOC:BRAN:ALW 17, 12
TRIG:BLOC:DIG:IO 18, 2, 15
TRIG:BLOC:BRAN:ALW 19, 12
TRIG:BLOC:DIG:IO 20, 3, 15
TRIG:BLOC:BRAN:ALW 21, 12
TRIG:BLOC:DIG:IO 22, 4, 15
TRIG:BLOC:BRAN:ALW 23, 12
```

Resistor sorting TSP code

```
local number_of_resistors = 100
-- Reset instrument to default settings
reset()
-- Measure function must be first measure setting;
-- most other settings are tied to the function
smu.measure.func = smu.DC_VOLTAGE
smu.measure.unit = smu.UNIT_OHM
-- Use 4-wire or "remote" voltage sensing
smu.measure.sense = smu.SENSE_4WIRE
-- Measure the actual value of the source for higher accuracy
smu.source.readback = smu.ON
-- Set measurement integration time to 1 PLC (16.67 ms at 60 Hz)
-- Decrease to reduce test time; trade off accuracy for speed
smu.measure.nplc = 1

-- Immediately update autozero reference measurements and then disable autozero
   function
smu.measure.autozero.once()
-- Source settings must come after MANUAL ohms is enabled
-- Source function must be first source setting;
-- most other settings are tied to the function
smu.source.func = smu.FUNC_DC_CURRENT

-- This is actually a voltage range because of manual ohms mode
-- Set this after setting source function to current
smu.measure.range = 2

-- This is a current range
smu.source.range = 0.01
-- Set source level to 10 mA
smu.source.level = 0.01
-- Set voltage limit of current source to 2 V; set this after setting measure range
smu.source.vlimit.level = 2
-- This example records the resistance measurements for later statistical analysis.
-- Limit inspection and binning can be performed without recording the
   measurements.
-- Set default buffer equal to the number of resistors to be tested
defbuffer1.capacity = number_of_resistors

-- Configure digital I/O lines 1 through 4 as digital outputs
-- These I/O lines are used to output binning code to component handler
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[2].mode = digio.MODE_DIGITAL_OUT
digio.line[3].mode = digio.MODE_DIGITAL_OUT
digio.line[4].mode = digio.MODE_DIGITAL_OUT

-- Configure digital I/0 line 5 as a trigger input
-- Used to detect start-of-test trigger from component handler
digio.line[5].mode = digio.MODE_TRIGGER_IN
-- Set trigger detector to detect falling edge
trigger.digin[5].edge = trigger.EDGE_FALLING
```

```
-- Configure digital I/0 line 6 as a trigger output
-- Used to send end-of-test trigger to component handler
digio.line[6].mode = digio.MODE_TRIGGER_OUT
-- Output a falling edge trigger
trigger.digout[6].logic = trigger.LOGIC_NEGATIVE
-- Set width of output trigger pulse to 10 us
trigger.digout[6].pulsewidth = 10E-6
-- Trigger pulse will be output when Notify Block generates an event
trigger.digout[6].stimulus = trigger.EVENT_NOTIFY1
-- Reset existing trigger model settings.
trigger.model.load("EMPTY")

-- Configure the Trigger Model
-- Block 1: Clear defbuffer1
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
-- Block 2: Wait for start-of-test trigger on digital I/O line 5
trigger.model.setblock(2, trigger.BLOCK_WAIT, trigger.EVENT_DIGIO5)
-- Block 3: Turn SMU output ON
trigger.model.setblock(3, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
-- Block 4: Delay for 1 ms to allow source to settle; adjust as appropriate
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 5: Measure resistance and store result in defbuffer1
trigger.model.setblock(5, trigger.BLOCK_MEASURE, defbuffer1)
-- Block 6: Turn SMU output OFF
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Block 7: Check if 99<=R<=101; if yes, go to Block 16
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 99, 101, 16, 5)
-- Block 8: Check if 101<=R<=103; if yes, go to Block 18
trigger.model.setblock(8, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 101, 103, 18, 5)
-- Block 9: Check if 104<=R<=106; if yes, go to Block 20
trigger.model.setblock(9, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 104, 106, 20, 5)
-- Block 10: Check if 106<=R<=108; if yes, go to Block 22
trigger.model.setblock(10, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,
   trigger.LIMIT_INSIDE, 106, 108, 22, 5)
-- Block 11: Set digital I/O lines 1-4; output decimal 15 (binary 1111) to
   component handler
trigger.model.setblock(11, trigger.BLOCK_DIGITAL_IO, 15, 15)
-- Block 12: Delay 1 ms; controls duration of digital bit patterns; adjust as
   appropriate
trigger.model.setblock(12, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 13: Set digital I/O lines 1-4; output decimal 0 (binary 0000) clear
   pattern to component handler
trigger.model.setblock(13, trigger.BLOCK_DIGITAL_IO, 0, 15)
-- Block 14: Notify block generates event, which causes output of a trigger pulse
   on digital I/O line 6
trigger.model.setblock(14, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
-- Block 15: Loop back to Block 2; keep looping until all resistors have been
   tested
trigger.model.setblock(15, trigger.BLOCK_BRANCH_COUNTER, number_of_resistors, 2)
-- Block 16: Set digital I/O lines 1-4; output decimal 1 (binary 0001) to component
   handler
trigger.model.setblock(16, trigger.BLOCK_DIGITAL_IO, 1, 15)
```

```
-- Block 17: Go to Block 12
trigger.model.setblock(17, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- Block 18: Set digital I/O lines 1-4; output decimal 2 (binary 0010) to component
   handler
trigger.model.setblock(18, trigger.BLOCK_DIGITAL_IO, 2, 15)
-- Block 19: Go to Block 12
trigger.model.setblock(19, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- Block 20: Set digital I/O lines 1-4; output decimal 3 (binary 0011) to component
   handler
trigger.model.setblock(20, trigger.BLOCK_DIGITAL_IO, 3, 15)
-- Block 21: Go to Block 12
trigger.model.setblock(21, trigger.BLOCK_BRANCH_ALWAYS, 12)
-- Block 22: Set digital I/O lines 1-4; output decimal 4 (binary 0100) to component
   handler
trigger.model.setblock(22, trigger.BLOCK_DIGITAL_IO, 4, 15)
-- Block 23: Go to Block 12
trigger.model.setblock(23, trigger.BLOCK_BRANCH_ALWAYS, 12)

-- After executing all of the above commands the trigger model can be initiated by
   executing the command trigger.model.initiate()
```

Diode grading with immediate binning example

This example shows a trigger model that can be used for diode test grading. Parts are immediately binned after testing.

In this example, each diode must pass three individual tests to be deemed good. The tests that each diode must pass are:

- Forward Voltage (VF)
- Reverse Breakdown Voltage (VR)
- Reverse Leakage Current (IR)

A sweep is commonly used to perform these measurements.

Each individual measurement is inspected against a single corresponding limit. This example uses end binning so that test data can be gathered for all three tests, even if one of the tests fails. The bit patterns assigned to the limits determine which bin a diode is placed in. It is assumed that the same digital bit pattern is assigned to both the upper and lower bounds of each limit. Using different "fail" bit patterns for each test makes it possible to place a failed diode in a specific bin according to its first detected failure (for example, Bad VF, Bad VR or Bad IR). If the same "fail" bit pattern is used for all of the tests, then any failure will cause the diode-under-test (DUT) to be placed in a bad part bin.

The tests are performed in ascending numeric order: Test 1 (VF), Test 2 (VR) and then Test 3 (IR). Therefore, the value of VF is inspected against its limit first. The requirement for the value of the limit's lower bound is that it not exceed some maximum value. To prevent false failures, you need to set the lower limit value low enough that it will always be less than any value of V that may be measured. The same is true for setting the lower limit value for IR.

If the VF test fails, the failure is recorded and testing continues with Test 2, the VR test. If the VF test passes, then testing continues with the VR test.

If the VR test fails, and this is the first test failure, the failure is logged and testing continues with Test 3, the IR test. If the VR test passes, testing continues with the IR test, and so on until all three tests are complete and all limit inspections have been performed.

If no tests fail, the overall "pass" digital bit pattern is output to the digital I/O, which causes the component handler to place the diode in the Good Part bin. If any of the tests fail during the test sequence, the "fail" bit pattern corresponding to the first failure is output, which causes the diode to be placed in either an overall Bad Part bin, or a more specific Bad VF, Bad VR or Bad IR bin.

Figure 100: Diode grading trigger model with immediate binning blocks 1 to 8

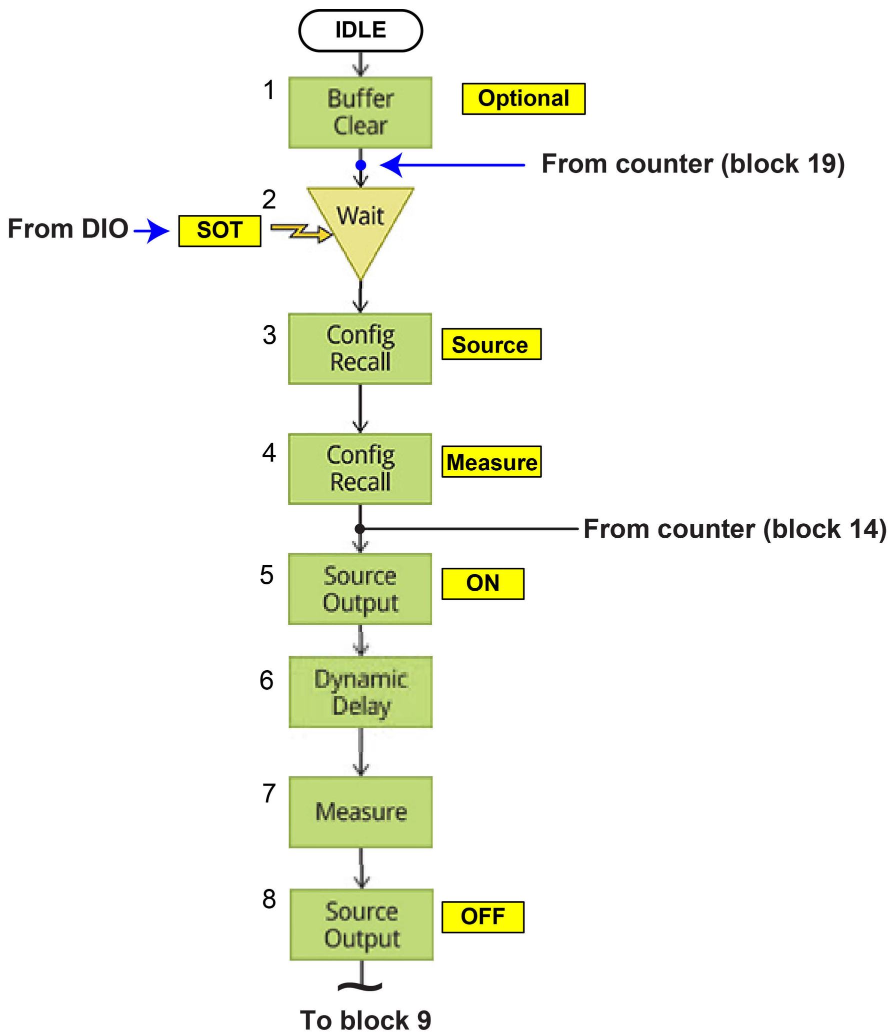
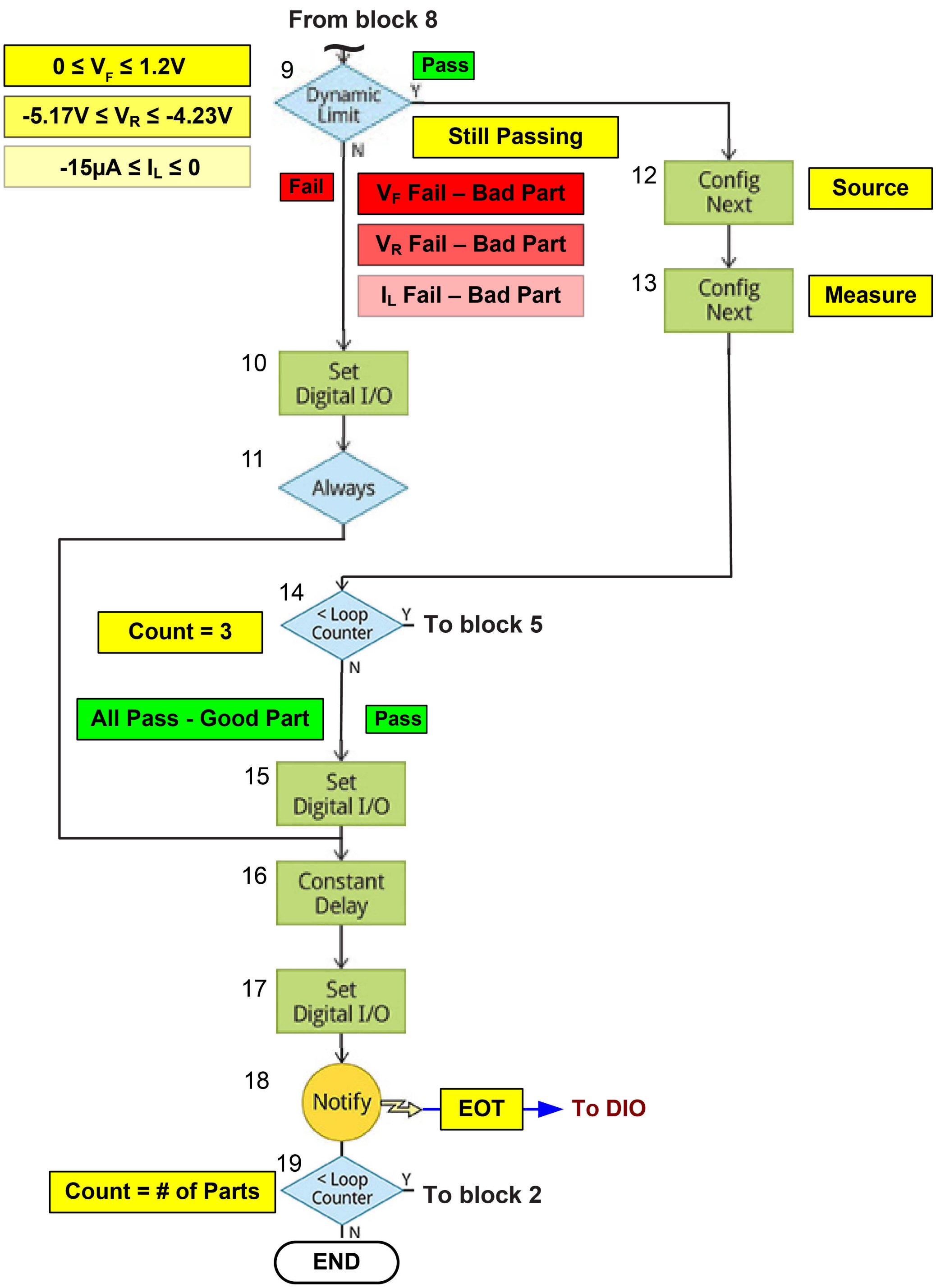


Figure 101: Diode grading trigger model with immediate binning blocks 9 to 19



Diode grading SCPI code

```
*RST
SOUR:CONF:LIST:CRE "SOURCE_LIST"
SENS:CONF:LIST:CRE "MEASURE_LIST"

SENS:FUNC "VOLT"
SOUR:FUNC CURR
SENSe:RESistance:RSENse ON
SENS:NPLC 1
SENS:VOLT:RANG 2.4
SOUR:CURR:RANG 1
SOUR:CURR 1
SOUR:CURR:VLIM 2.4
SOUR:CURR:DEL:USER1 0.001
SOUR:CONF:LIST:STORE "SOURCE_LIST"
CALC2:LIM1:STAT ON
CALC2:LIM1:LOW 0
CALC2:LIM1:UPP 1.2
SENS:CONF:LIST:STORE "MEASURE_LIST"

SENS:FUNC "VOLT"
SOUR:FUNC CURR
SYST:RSEN ON
SENS:NPLC 1
SENS:VOLT:RANG 9.4
SOUR:CURR:RANG -0.26
SOUR:CURR -0.26
SOUR:CURR:VLIM 9.4
SOUR:CURR:DEL:USER1 0.001
SOUR:CONF:LIST:STORE "SOURCE_LIST"
CALC2:LIM1:STAT ON
CALC2:LIM1:LOW -5.17
CALC2:LIM1:UPP -4.23
SENS:CONF:LIST:STORE "MEASURE_LIST"

SENS:FUNC "CURR"
SOUR:FUNC VOLT
SYST:RSEN ON
SENS:NPLC 1
SENS:CURR:RANG 30e-6
SOUR:VOLT:RANG -1.5
SOUR:VOLT -1.5
SOUR:VOLT:ILIM 30e-6
SOUR:VOLT:DEL:USER1 0.01
SOUR:CONF:LIST:STORE "SOURCE_LIST"
CALC2:LIM1:STAT ON
CALC2:LIM1:LOW -15e-6
CALC2:LIM1:UPP 0
SENS:CONF:LIST:STORE "MEASURE_LIST"

SOUR:CONF:LIST:SIZE? "SOURCE_LIST"
SOUR:CONF:LIST:QUERY? "SOURCE_LIST"
SENS:CONF:LIST:SIZE? "MEASURE_LIST"
SENS:CONF:LIST:QUERY? "MEASURE_LIST"

TRAC:POIN 30
```

```
DIG:LINE1:MODE DIG, OUT
DIG:LINE2:MODE DIG, OUT
DIG:LINE3:MODE DIG, OUT
DIG:LINE4:MODE DIG, OUT

DIG:LINE5:MODE TRIG, IN
TRIG:DIG5:IN:EDGE FALL

DIG:LINE6:MODE TRIG, OUT
TRIG:DIG6:OUT:LOG NEG
TRIG:DIG6:OUT:PULS 10e-6
TRIG:DIG6:OUT:STIM NOT1

TRIG:LOAD:EMPT

TRIG:BLOC:BUFF:CLE 1, "defbuffer1"
TRIG:BLOC:WAIT 2, DIG5
TRIG:BLOC:CONF:RECALL 3, "SOURCE_LIST"
TRIG:BLOC:CONF:RECALL 4, "MEASURE_LIST"
TRIG:BLOC:SOUR:STAT 5, ON
TRIG:BLOC:DEL:DYN 6, SOUR1
TRIG:BLOC:MEAS 7, "defbuffer1"
TRIG:BLOC:SOUR:STAT 8, OFF

TRIG:BLOC:DIG:IO 10, 14, 15
TRIG:BLOC:BRAN:LIM:DYN 9, IN, 1, 12, 7
TRIG:BLOC:BRAN:ALW 11, 16
TRIG:BLOC:CONF:NEXT 12, "SOURCE_LIST"
TRIG:BLOC:CONF:NEXT 13, "MEASURE_LIST"
TRIG:BLOC:BRAN:COUN 14, 3, 5
TRIG:BLOC:DIG:IO 15, 1, 15
TRIG:BLOC:DEL:CONS 16, 0.001
TRIG:BLOC:DIG:IO 17, 0, 15
TRIG:BLOC:NOT 18, 1
TRIG:BLOC:BRAN:COUN 19, 10, 2
```

Diode grading TSP code

```
-- Number of diodes to test
local number_of_diodes = 100
-- Reset instrument to default settings
reset()

-- Create source and measure configuration lists
smu.source.configlist.create("SOURCE_LIST")
smu.measure.configlist.create("MEASURE_LIST")

-- Create data tables to simplify setup code
-- Three diode tests performed in the following order: Vf, Vr (limits actually for
   Zener diode voltage, Vz), and Il
local SRC_FUNC = {smu.FUNC_DC_CURRENT, smu.FUNC_DC_CURRENT, smu.FUNC_DC_VOLTAGE}
local SRC_RANGE_LEVEL = {1, -0.260, -1.5}
-- When sourcing current you limit the voltage
local SRC_VLIMIT = {2.4, 9.4, nil}
-- When sourcing voltage you limit the current
local SRC_ILIMIT = {nil, nil, 30e-6}
local SRC_DELAY = {0.001, 0.001, 0.01}

local MEAS_FUNC = {smu.FUNC_DC_VOLTAGE, smu.FUNC_DC_VOLTAGE, smu.FUNC_DC_CURRENT}
local MEAS_RANGE = {2.4, 9.4, 30e-6}
local MEAS_LIMIT_LOW = {0,   -5.17, -15e-6}
local MEAS_LIMIT_HIGH = {1.2, -4.23,  0    }

-- Set up the configuration lists
for i = 1, 3 do

    -- Source function must be first source setting;
    -- most other settings are tied to the function
   smu.source.func = SRC_FUNC[i]

   -- Type of range depends on source function
   smu.source.range = SRC_RANGE_LEVEL[i]
   -- Type of level depends on source function
   smu.source.level = SRC_RANGE_LEVEL[i]

    -- There are 5 user source delays for each point in a configuration list
   -- Only using user delay 1
   smu.source.userdelay[1] = SRC_DELAY[i]

   -- Measure function must be first measure setting;
   -- most other settings are tied to the function
   smu.measure.func = MEAS_FUNC[i]

   -- Use 4-wire or "remote" voltage sensing
   smu.measure.sense = smu.SENSE_4WIRE
   -- Type of range depends on source function
   smu.measure.range = MEAS_RANGE[i]

    -- Set source limit after setting measure range to avoid conflict
    -- between measure range and corresponding limit range
   if SRC_VLIMIT[i] then
      smu.source.vlimit.level = SRC_VLIMIT[i]
   else
```

```
      smu.source.ilimit.level = SRC_ILIMIT[i]
   end

   -- Set measurement integration time to 1PLC (16.67ms at 60Hz)
   -- Decrease to reduce test time; trade off accuracy for speed
   smu.measure.nplc = 1
   -- Immediately update autozero reference measurements and then disable autozero
    function
   smu.measure.autozero.once()
    -- Enable measure Limit 1
    smu.measure.limit[1].enable = smu.ON
    -- Set Limit 1 low limit value
    smu.measure.limit[1].low.value = MEAS_LIMIT_LOW[i]
    -- Set Limit 1 high limit value
    smu.measure.limit[1].high.value = MEAS_LIMIT_HIGH[i]

    smu.source.output = smu.ON
   -- Append source settings to source configuration list
    smu.source.configlist.store("SOURCE_LIST")
   -- Append measure settings to measure configuration list
    smu.measure.configlist.store("MEASURE_LIST")
    -- Turn SMU output OFF after storing source configuration list point
    smu.source.output = smu.OFF

end --for

-- This example records the resistance measurements for later statistical analysis.
-- Limit inspection and binning can be performed without recording the
   measurements.

-- Set buffer capacity equal to 3 tests per diode * number of diodes to be tested
defbuffer1.capacity = 3 * number_of_diodes

-- Configure digital I/O lines 1 through 4 as digital outputs
-- These I/O lines are used to output binning code to component handler
digio.line[1].mode = digio.MODE_DIGITAL_OUT
digio.line[2].mode = digio.MODE_DIGITAL_OUT
digio.line[3].mode = digio.MODE_DIGITAL_OUT
digio.line[4].mode = digio.MODE_DIGITAL_OUT

-- Configure digital I/0 line 5 as a trigger input
-- Used to detect start-of-test trigger from component handler
digio.line[5].mode = digio.MODE_TRIGGER_IN
-- Set trigger detector to detect falling edge
trigger.digin[5].edge = trigger.EDGE_FALLING

-- Configure digital I/0 line 6 as a trigger output
-- Used to send end-of-test trigger to component handler
digio.line[6].mode = digio.MODE_TRIGGER_OUT
 -- Output a falling edge trigger
trigger.digout[6].logic = trigger.LOGIC_NEGATIVE
 -- Set width of output trigger pulse to 10us
-- Trigger pulse will be output when Notify Block generates an event
trigger.digout[6].pulsewidth = 10E-6
trigger.digout[6].stimulus = trigger.EVENT_NOTIFY1

trigger.model.load("EMPTY") -- Reset existing trigger model settings
```

```
-- Configure the Trigger Model
-- Block 1: Clear defbuffer1
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
-- Block 2: Wait for start-of-test trigger on digital I/O line 5
trigger.model.setblock(2, trigger.BLOCK_WAIT, trigger.EVENT_DIGIO5)
-- Block 3: Recall the first point in the source configuration list
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "SOURCE_LIST")
-- Block 4: Recall the first point in the measure configuration list
trigger.model.setblock(4, trigger.BLOCK_CONFIG_RECALL, "MEASURE_LIST")
-- Block 5: Turn SMU output ON
trigger.model.setblock(5, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
-- Block 6: Allow source to settle; delay stored in source configuration list;
   adjust as appropriate
trigger.model.setblock(6, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_S1)
-- Block 7: Measure voltage or current as determined by measure configuration list
   and store result in defbuffer1
trigger.model.setblock(7, trigger.BLOCK_MEASURE, defbuffer1)
-- Block 8: Turn SMU output OFF
trigger.model.setblock(8, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
-- Block 9: Perform Limit 1 test; limit values stored in measure configuration
   list; if pass, go to Block 12
trigger.model.setblock(9, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, trigger.LIMIT_INSIDE,
   1, 12, 7)
-- Block 10: Set digital I/O lines 1-4; output decimal 14 (binary 1110) to
   component handler
trigger.model.setblock(10, trigger.BLOCK_DIGITAL_IO, 14, 15)
-- Block 11: Go to Block 16
trigger.model.setblock(11, trigger.BLOCK_BRANCH_ALWAYS, 16)
-- Block 12: Recall the next point in the source configuration list
trigger.model.setblock(12, trigger.BLOCK_CONFIG_NEXT, "SOURCE_LIST")
-- Block 13: Recall the next point in the measure configuration list
trigger.model.setblock(13, trigger.BLOCK_CONFIG_NEXT, "MEASURE_LIST")
-- Block 14: Loop back to Block 5; will loop 3 times unless diode fails one of its
   tests
trigger.model.setblock(14, trigger.BLOCK_BRANCH_COUNTER, 3, 5)
-- Block 15: Set digital I/O lines 1-4; output decimal 1 (binary 0001) to component
   handler
trigger.model.setblock(15, trigger.BLOCK_DIGITAL_IO, 1, 15)
 -- Block 16: Delay 1ms; controls duration of digital bit patterns; adjust as
   appropriate
trigger.model.setblock(16, trigger.BLOCK_DELAY_CONSTANT, 0.001)
-- Block 17: Set digital I/O lines 1-4; output decimal 0 (binary 0000) clear
   pattern to component handler
trigger.model.setblock(17, trigger.BLOCK_DIGITAL_IO, 0, 15)
-- Block 18: Notify block generates event, which causes output of a trigger pulse
   on digital I/O line 6
trigger.model.setblock(18, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
-- Block 19: Loop back to Block 2; keep looping until all diodes have been tested
trigger.model.setblock(19, trigger.BLOCK_BRANCH_COUNTER, number_of_diodes, 2)

-- After executing all of the above commands the trigger model can be initiated by
   executing "trigger.model.initiate()"
```

Triggering

Triggers are signals that instruct the instrument to make a measurement. The Model 2450 can be set to use the following triggering measurement methods:

- Continuous measurements: The instrument continuously makes measurements.
- Manual trigger key: The instrument makes measurements when you press the **TRIGGER** key on the front panel.
- Trigger model: The instrument makes measurements according to the settings of the trigger model.

Continuous measurement triggering

When you select the continuous measurement method, the instrument makes measurements continuously.

The continuous measurement method is only available when you are controlling the instrument locally (through the front panel).

The instrument stores the readings in a reading buffer. See Reading buffers (on page 3-10) for detail on the buffer options that are available.

If you press the front-panel **TRIGGER** key when the instrument is set to the continuous measurement method, measurements are not made. Instead, a dialog box is displayed that asks if you want to change the measurement method.

Trigger key triggering

When you select the manual trigger key method of triggering, the instrument only makes a measurement when you press the front-panel **TRIGGER** key.

The manual trigger key measurement method is only available when you are controlling the instrument through the front panel.

The instrument stores the readings in a reading buffer. See Reading buffers (on page 3-10) for detail on the buffer options that are available.

Trigger model triggering

When you select the trigger model measurement method, the instrument uses a trigger model to control the sequence in which measurements occur. The Model 2450 trigger model is flexible, allowing you to control as much or as little as needed for your measurement application.

For detail on the trigger model, see Trigger model (on page 3-95).

Switching between measurement methods

The measurement methods that are available to you depend on how you are controlling the instrument.

If you are using the front panel to control the instrument, you can choose any of the measurement methods.

If you are using a remote interface to control the instrument, you can only use the trigger model measurement method. When you switch to a remote interface, the trigger model measurement method is automatically selected. If you switch from remote control to front-panel control, the trigger model measurement method remains selected.

Using the front panel:

1. Press the front-panel **TRIGGER** key for two seconds. A screen displays with the available trigger methods; the presently selected method is in yellow type.
2. Select the method you want to use.
3. If the instrument is in remote control, the instrument displays a confirmation screen. Select **Yes** to change to local control.
4. Select **HOME** to return to the operating display.

Understanding synchronous triggering modes

Use the synchronous triggering modes to implement bidirectional triggering, to wait for one node, or to wait for a collection of nodes to complete all triggered actions.

All non-Keithley instrumentation must have a trigger mode that functions similar to the SynchronousA or SynchronousM trigger modes.

To use synchronous triggering, configure the triggering master to SynchronousM trigger mode or the non-Keithley equivalent. Configure all other nodes in the test system to SynchronousA trigger mode or a non-Keithley equivalent.

Synchronous master trigger mode (SynchronousM)

Use the synchronous master trigger mode (SynchronousM) to generate falling edge output triggers, to detect the rising edge input triggers, and to initiate an action on one or more external nodes with the same trigger line.

In this mode, the output trigger consists of a low pulse. All non-Keithley instruments attached to the synchronization line in a trigger mode equivalent to SynchronousA must latch the line low during the pulse duration.

To use the SynchronousM trigger mode, configure the triggering master as SynchronousM and then configure all other nodes in the test system as Synchronous, SynchronousA, or to the non-Keithley Instruments equivalent.

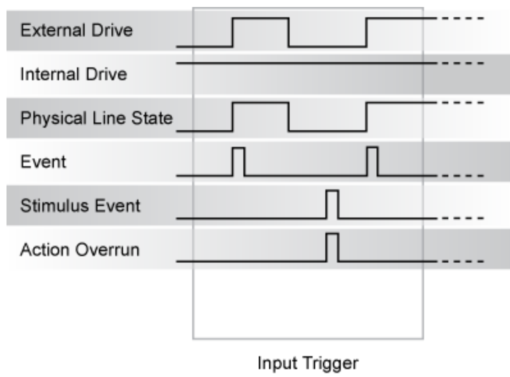
NOTE

Use the SynchronousM trigger mode to receive notification when the triggered action on all nodes is complete.

Input characteristics:

- All rising edges are input triggers.
- When all external drives release the physical line, the rising edge is detected as an input trigger.
- A rising edge is not detected until all external drives release the line and the line floats high.

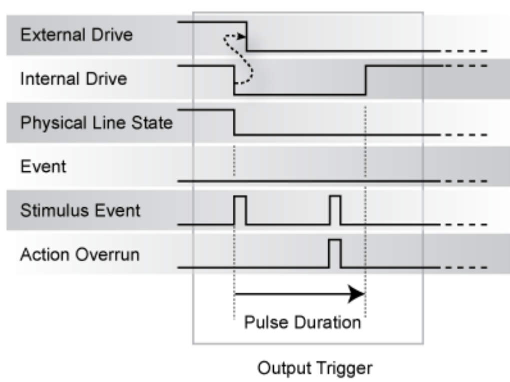
Figure 102: SynchronousM input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the TSP commands `trigger.digout[N].assert()` and `trigger.tsplinkout[N].assert()` generate a low pulse that is similar to the falling edge trigger mode.
- An action overrun occurs if the physical line state is low while a stimulus event occurs.

Figure 103: SynchronousM output trigger



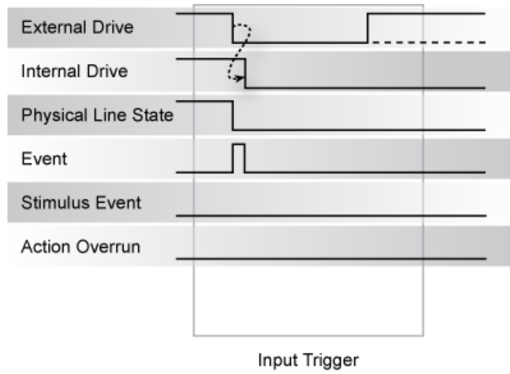
Synchronous acceptor trigger mode (SynchronousA)

Use the synchronous acceptor trigger mode (SynchronousA) with 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.

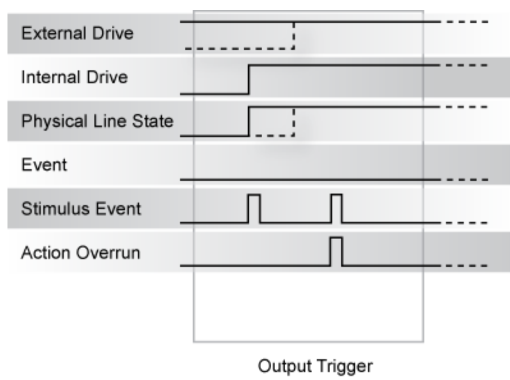
Figure 104: SynchronousA input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the TSP commands `trigger.digout[N].assert()` and `trigger.tsplinkout[N].assert()` 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 105: SynchronousA output trigger



Trigger model

The trigger model controls the sequence in which measurements occur. The Model 2450 trigger model is flexible, allowing you to control as much or as little as needed for your measurement application.

When you are setting up a trigger model, you can choose the following options:

- Wait for an event to occur before taking another measurement
- Notify other equipment that an event has occurred
- Wait for another piece of equipment to signal completion
- Use measure configuration lists to apply different measurement settings dynamically during trigger model operation
- Specify delays between events and measurements
- Use source configuration lists to sweep source settings and values
- Turn the source on and off with programmable delays to create pulses
- Store measurements into a given buffer until an event occurs, then switch to another buffer
- Conditionally take actions based on whether the measurement falls within set limits

Additional options are detailed in the following sections.

The Model 2450 includes predefined trigger models to allow you to quickly implement a trigger model. You can also set up your own trigger models.

Trigger model building blocks

Each trigger model consists of building blocks that can be combined to create the trigger model. The building blocks can be combined from the front panel or by sending remote commands. You can connect the building blocks as needed to control the instrument.

You can combine trigger model building blocks as you would construct a flow chart diagram. Trigger models are created using four fundamental building blocks:

- Wait: Waits for an event to occur before the flow continues
- Branch: Branches when a condition has been satisfied
- Action: Starts an action in the instrument, such as making a measurement or turning on a source
- Notify: Notifies other equipment that an event has occurred

Each type of building block is described in the following topics.

Reading-buffer clear building block

When the trigger model reaches the reading-buffer clear building block, the instrument empties a reading buffer. The buffer can be the default buffer or a buffer that you defined. If you do not define a buffer, the instrument clears the default buffer (`defbuffer1`).

Readings that are made after the buffer is cleared are added to the beginning of the buffer.

If you are defining a specific reading buffer, you must define it before you define this block. For more information about reading buffers, see Reading buffers (on page 3-10).

Measure building block

When the trigger model reaches the measurement block:

1. The instrument makes a reading.
2. The trigger model waits for the measurement to complete.
3. The instrument places the measurement into the specified reading buffer. If no buffer is specified, the reading is placed into the default buffer (`defbuffer1`).

If you are defining a specific reading buffer, you must create it before you define this block.

You can add a delay before or after the measurement by adding a delay building block to the trigger model.

Source building block

The source building block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as `SOURCE_OUTPUT`.

Timing building blocks

Wait building block

The wait building block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to three events for each wait block.

You can use the wait block to synchronize measurements with other instruments and devices.

Events that you can set the instrument to wait for include:

- Digital input/output signals, such as DB-9 and TSP-Link
- LAN events
- Blenders

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. When the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again.

The instrument clears the memory of the recorded event when the trigger model is at the start block and when the trigger model exits the wait block.

You can have up to eight wait blocks in a trigger model.

All items in the list are subject to the same action — you cannot combine `AND` and `OR` logic in a single command.

The events can be one of the events shown in the following table. If you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block. For information, see Branching building blocks (on page 3-98).

Delay building block

When the trigger model reaches a delay building block, it stops the trigger model for the amount of time set by the delay.

The delay time is set by the user delay command. This delay can be different for every point in the configuration list. This makes it possible to have a delay that changes as the configuration list progresses.

NOTE

There are additional delay settings available in the instrument (such as measure delays and source delays). To simplify trigger model development, it is best practice to use only the trigger model delay blocks. Turn off any other delays before running the trigger model.

Notify building block

When the trigger model reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

You can define up to eight notify blocks in a trigger model. You can reference the event that the notify block generates by other commands to assign a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

See Using the notify block event (on page 3-111) for detail on using the notify event.

Configuration list building blocks

You can use configuration list building blocks to recall settings that are stored in a configuration list. When the trigger model reaches a configuration list building block, the commands in the configuration list are executed.

The trigger model building blocks that recall configuration lists are:

- Configuration recall
- Configuration next
- Configuration previous

For detail on configuration lists, see Configuration lists (on page 3-33).

You must define the configuration list before you define the trigger model configuration list building blocks.

Recall index

You can use the configuration list recall index building block to load a specific index from a configuration list.

If you do not specify the index in the configuration list, the entire configuration list is restored. If an index is specified, the commands at that index are restored.

All parameters are changed and will take effect before this execution block completes and the next execution block begins.

In most cases, you should load the first configuration index at the beginning of the trigger model to ensure that the correct initial state is set and that the trigger model will be repeatable.

Recall next

When the trigger model reaches a configuration recall next building block, the settings at the next index point in a configuration list are restored.

Each time this block is encountered, the settings at the next index point in the configuration list are recalled and take effect before the next step executes. When the last index point in the list is reached, it returns to the first point.

Recall previous

The configuration list previous index trigger block type recalls the previous index point in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

Each time the trigger model reaches a configuration list previous block, it goes backward one index point. When the first point in the list is reached, it goes to the last index point in the configuration list.

Digital input/output building block

To set the lines on the digital I/O port high or low, you can send a bit pattern. The pattern can be specified as an integer value, or, if you are using the TSP command set, a six-bit binary or hexadecimal. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The optional bit mask defines the bits in the pattern that are driven high or low. If the bit for a line is set to 1, the line is driven high. If the bit is set to 0, the line is driven low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63, 0x3F, 0b111111) or omit this parameter.

For this command to function as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Branching building blocks

A branch block goes to a trigger block other than the sequential execution block.

For example, if you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block to define when the wait block should be enabled. You can use the Branch Once block to create a bypass and skip the wait block the first time the trigger model runs. This makes it possible to avoid deadlock when multiple instruments are being synchronized and each one is waiting for notification from the other one to start the trigger model.

Loop counter

When the trigger model reaches a loop counter block, it goes to a specified block until the count value is reached. When the counter exceeds the count value, the trigger model ignores the branch and continues to the next building block in the sequence.

The counter is reset to 0 when the trigger model starts. It is incremented each time the trigger model reaches the counter block.

You can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the setting. Therefore, the counter is at 0 until the first comparison. When the counter value has been reached, branching stops and the counter value is greater than the setting.

On event

The branch-on-event building block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

The event can be one of the events shown in the following table.

The trigger event is defined as an event. The events are reset when the trigger model is at the start block, so only events that occur after the trigger model is started are detected by the branch-on-event block. The event is also reset after the trigger model completes the branching block.

You can have up to eight branch-on-event blocks in a trigger model.

For information on trigger events, see Using trigger events to start actions in the trigger model (on page 3-109).

Constant limits

The branch-on-constant-limits building block defines a trigger model block that branches to a block outside the normal trigger model flow if a measurement meets preset criteria.

When you define this building block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The constant limit value or values
- The building block to go to if the measurement meets the criteria
- The building block that makes the measurement that is compared to the limits; the last measurement from that building block is used

You can use this block to create a binning application by having the block branch to a digital I/O block, followed by a branch always block. Multiple tests can be chained together by repeating this.

To use limits that vary programmatically, use the branch-on-dynamic-limits block.

Dynamic limits

The branch-on-dynamic-limits building block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this building block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- Two user-defined limit sets
- The building block to go to if the measurement meets the criteria
- The building block that makes the measurement that is compared to the limits; the last measurement from that building block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values. You set these limit threshold values as separate settings. Limit 1 and limit 2 are stored in the measurement configuration list. You can set them to different values in different indices of the measurement configuration list to allow you to step through different values. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Delta

The branch-on-delta building block defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger model sequence.

If you do not define the measurement block, it will compare measurements of a measure block that precedes the branch delta block. For example, if you have a measure block, a wait block, another measure block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure block.

When you define this building block, you set:

- The target difference; when the difference between the measurements is less than or equal to this value, the trigger model goes to the specified block
- The building block to go to if the difference is less than or equal to the target difference
- The building block that makes the measurements that are compared to the limits; the last two measurements from that building block are used

Always

When the trigger model reaches a branch-always building block, it goes to a specified building block.

Once

When the trigger model reaches a branch-once building block, it goes to a specified building block the first time it is encountered in the trigger model. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

You can use this block to create a bypass. For example, you might place a branch-once block before a wait block to skip the wait block on the first pass of the trigger model.

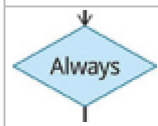
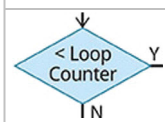
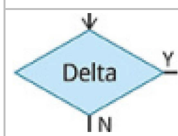
The once block is reset when the trigger model reaches the idle state. Therefore, the branch-once block will always execute the first time the trigger model encounters this block.

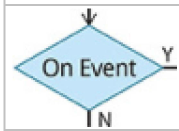
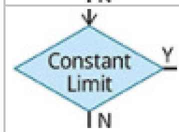
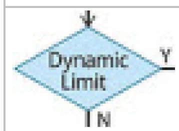
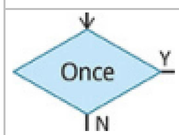
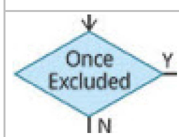
Once excluded

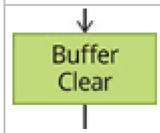
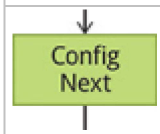
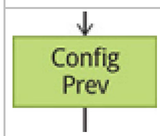
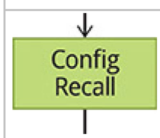
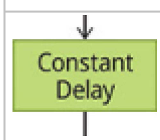
The branch-once-excluded building block is ignored by the trigger model the first time it is encountered. If the trigger model encounters the block again, it goes to a specified building block.

The branch-once-excluded block is reset when the trigger model starts.

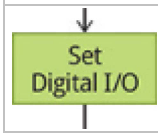
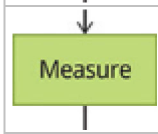
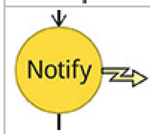
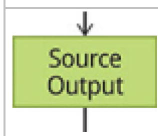
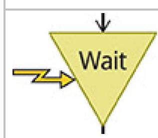
Trigger block summary

Front-panel icon	SCPI command TSP command	Block description
Not applicable	:TRIGger:BLOCk:LIST? (on page 6-142) trigger.model.getblocklist() (on page 8-186)	This returns the settings for all trigger model building blocks
	:TRIGger:BLOCk:BRANch:ALWays (on page 6-128) trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS (on page 8-192)	This defines a trigger model block that always goes to a specific block
Not applicable	:TRIGger:BLOCk:BRANch:COUNter (on page 6-128) (query) trigger.model.getbranchcount() (on page 8-187)	This returns the count value of the trigger model counter block
	:TRIGger:BLOCk:BRANch:COUNter (on page 6-128) trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 8-193)	This branches to a specified block a specified number of times
	:TRIGger:BLOCk:BRANch:DELTa (on page 6-130) trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA (on page 8-194)	This defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCk:BRANch:EVENt (on page 6-131) trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT (on page 8-197)	This branches to a specified block when a specified trigger event occurs
	:TRIGger:BLOCk:BRANch:LIMit:CONStant (on page 6-132) trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT (on page 8-195)	This defines a trigger model block that branches to a block outside the normal trigger model flow if a measurement meets preset criteria
	:TRIGger:BLOCk:BRANch:LIMit:DYNamic (on page 6-133) trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC (on page 8-196)	This defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria
	:TRIGger:BLOCk:BRANch:ONCE (on page 6-134) trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE (on page 8-199)	This causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model
	:TRIGger:BLOCk:BRANch:ONCE:EXCLuded (on page 6-135) trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED (on page 8-200)	This causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCk:BUFFer:CLEar (on page 6-136) trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR (on page 8-201)	This defines a trigger model block that clears the reading buffer
	:TRIGger:BLOCk:CONFig:NEXT (on page 6-136) trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT (on page 8-202)	This recalls the settings at the next index point of a source or measure configuration list
	:TRIGger:BLOCk:CONFig:PREVious (on page 6-137) trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV (on page 8-203)	This defines a trigger model block that recalls the settings stored at the previous index point in a measure or source configuration list
	:TRIGger:BLOCk:CONFig:RECall (on page 6-138) trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL (on page 8-204)	This recalls the system settings that are stored in a measure or source configuration list
	:TRIGger:BLOCk:DELay:DYNamic (on page 6-140) trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 8-206)	This adds a delay to the execution of the trigger model
	:TRIGger:BLOCk:DELay:CONStant (on page 6-139) trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT (on page 8-205)	This adds a constant delay to the trigger model

Trigger block summary

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCk:DIGital:IO (on page 6-141) trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO (on page 8-207)	This trigger model block that sets the lines on the digital I/O port high or low
	:TRIGger:BLOCk:MEASure (on page 6-144) trigger.model.setblock() — trigger.BLOCK_MEASURE (on page 8-209)	This defines a trigger block that makes a measurement
	:TRIGger:BLOCk:NOTify (on page 6-146) trigger.model.setblock() — trigger.BLOCK_NOTIFY (on page 8-211)	This defines a trigger model block that generates a trigger event and immediately continues to the next block
	:TRIGger:BLOCk:SOURce:STATe (on page 6-147) trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT (on page 8-212)	This defines a trigger block that turns the output source on or off
	:TRIGger:BLOCk:WAIT (on page 6-148) trigger.model.setblock() — trigger.BLOCK_WAIT (on page 8-212)	This defines a trigger model block that waits for an event before allowing the trigger model to continue

Predefined trigger models

The Model 2450 includes predefined trigger models for common applications. You can use these predefined trigger models without changing them, or you can modify them to meet the needs of your application.

The predefined trigger models include:

- **Empty:** Selecting this template clears the present trigger model.
- **Config List:** Creates a trigger model that can recall settings that are stored in a configuration list.
- **External Trigger:** Creates a trigger model that allows you to use the digital I/O to send signals to trigger external instruments.
- **Simple Loop:** Creates a trigger model that makes a specified number of readings. The count defines the number of readings.
- **Duration Loop:** Creates a trigger model that makes continuous measurements for a specified amount of time. When you start this trigger model, the output is turned on.

Using a predefined trigger model

Before starting the trigger model, you need to set up your instrument for testing, including the source and measurement settings. The trigger model will use these settings when making measurements.

Loading a predefined trigger model overwrites any existing trigger models.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Templates**.
3. Next to Trigger Model, select the trigger model template to use.
4. If the template you select has additional settings, you can use the default values or make any necessary changes to the settings.
5. Select **Generate**.
6. Press TRIGGER to initiate the trigger model. The trigger mode indicator shows the status of the trigger mode. See Trigger mode indicator (on page 2-16) for descriptions of the indicators.

Using SCPI commands:

See the descriptions of the `TRIGger:LOAD` commands for details on how to format the command:

- :TRIGger:LOAD:CONFiguration:LIST (on page 6-161)
- :TRIGger:LOAD:EMPTy (on page 6-162)
- :TRIGger:LOAD:LOOP:DURation (on page 6-163)
- :TRIGger:LOAD:LOOP:SIMPle (on page 6-164)
- :TRIGger:LOAD:TRIGger:EXTernal (on page 6-165)

Using TSP commands:

See the descriptions of the `trigger.model.load()` command for details on how to format the command:

- trigger.model.load() — Config List (on page 8-188)
- trigger.model.load() — Duration Loop (on page 8-189)
- trigger.model.load() — Empty (on page 8-190)
- trigger.model.load() — External Trigger (on page 8-190)
- trigger.model.load() — Simple Loop (on page 8-191)

Using a predefined trigger model to develop a trigger model

The Model 2450 includes predefined trigger models that you can use as a starting point for developing your trigger model.

After modifying a trigger model, you can save it in a saved setup for future use. See Saving setups (on page 2-112) for information on how to save a configuration.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Templates**. The TRIGGER MODEL TEMPLATES screen is displayed.
3. Next to Trigger Templates, select the trigger model to use.
4. If the template you select has additional settings, you can use the default values or make any necessary changes to the settings.
5. Select **Generate**.
6. Select **EXIT** to return to the MENU screen.
7. Under Trigger, select **Configurable**. The blocks for the predefined trigger model are displayed.
8. Choose or modify the blocks as needed. See Assembling trigger model building blocks (on page 3-106).
9. When the blocks are set up, select **EXIT** to return to the MENU screen.
10. Under Scripts, select **Create Config**.
11. Select **Create**.
12. Enter a configuration script name.
13. Click **OK**.

Assembling trigger model building blocks

This section describes the basic concepts you need to understand to assemble trigger model building blocks.

Sequencing trigger model building blocks

You can set up the trigger model building block from the front panel or by using remote commands.

Building blocks must sequenced in order — you cannot skip numbers. If you skip numbers, when the trigger model reaches the skipped number, it generates an error. When the trigger model completes the last block in the trigger model, the trigger model returns to start. Start is considered execution block 0 and branching to block 0 effectively stops the trigger model.

As the trigger model reaches each block, the action defined by that block is started and completed before the trigger model moves to the next block. Blocks do not overlap.

The trigger model steps through the building blocks in sequential order. You can set up branching blocks to allow nonsequential actions to occur. See Branching building blocks (on page 3-98) for detail on how to use the branching blocks.

Determining the structure of the existing trigger model

You can retrieve the existing trigger model structure from the front panel or by using remote commands.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Configure**.

   The trigger model is displayed.

Using SCPI commands:

To retrieve the settings for all trigger model building blocks, send the command:

```
:TRIGger:BLOCk:LIST?
```

Using TSP commands:

To check the settings for a building block, send the command:

```
print(trigger.model.getblocklist())
```

NOTE

To retrieve the TSP code for trigger model blocks that were entered through the front panel, change the Event Log "Command" setting to On. Refer to Using the event log (on page 2-117) for additional 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 event log. For more information on using the event log, see Using the event log (on page 2-117).

Running the trigger model

You can run the trigger model when the instrument is controlled either locally or remotely. If the instrument is being controlled remotely, trigger model is the only method of triggering that is available. If the instrument is being controlled locally, you can choose any of the measurement methods. Note that if you change from remote to local control, the trigger model method remains selected until you change it.

When you run the trigger model, the existing instrument settings are used for any actions unless you assigned configuration lists to the trigger model.

Other commands can be executed while the trigger model is running if they do not conflict with trigger model actions.

To change the measurement method, see Switching between measurement methods (on page 3-92).

Starting the trigger model

Using the front panel:

1. Press the front-panel **TRIGGER** key for two seconds. A screen displays with the available trigger methods; the presently selected method is in yellow type.
2. Select **Trigger Model (Initiate)**.
3. If the instrument is controlled remotely, a confirmation screen is displayed. Select **Yes** to change to front-panel control and start the trigger model.

Using SCPI commands:

Send the command:

```
:INITiate
```

Using TSP commands:

Send the command:

```
trigger.model.initiate()
```

Aborting the trigger model

You can stop the trigger model while it is in progress. When you stop the trigger model, all trigger model commands on the instrument are terminated, including sweeps.

Using the front panel:

Press the **TRIGGER** key for two seconds and select **Abort Trigger Model**.

Using SCPI commands:

Send the command:

```
:ABORt
```

Using TSP commands:

Send the command:

```
trigger.model.abort()
```

Checking the state of the trigger model

The trigger model can be in one of several states. The state is shown in the indicator bar on the Home page of the instrument. You can also check the status using remote commands.

The trigger model states are described in the following table. This table also shows the indicator that is shown on the front panel and the feedback you get from the remote interface.

Front panel indicator	Remote feedback	Description
CONT	Not available through remote interface	Instrument is making measurements continuously
MAN	Not available through remote interface	Instrument makes measurements when you press the front-panel TRIGGER key.
IDLE	`trigger.STATE_IDLE`	Trigger model stopped.
RUN	`trigger.STATE_RUNNING`	Trigger model is running
WAIT	`trigger.STATE_WAITING`	The trigger model has been in the same wait block for more than 100 ms

Using the front panel

The state of the trigger model is indicated on the status bar with the indicators shown in the previous table.

Using SCPI commands:

Send the command:

```
:TRIGger:STATe?
```

Using TSP commands:

Send the command:

```
print(trigger.model.state())
```

Using trigger events to start actions in the trigger model

You can set up trigger blocks to respond to trigger events. Trigger events are signals that can be generated by the instrument or by other system components.

Sources of the trigger event signals can be:

- Front-panel TRIGGER key
- Notify trigger blocks
- Branch-on-event trigger blocks
- Command interface triggers
- Digital I/O lines
- TSP-Link synchronization lines
- LAN triggers
- Event blenders, which combine other trigger events
- Trigger timers

Trigger events

To use trigger events, you need to specify the event constant. The tables below show the constants for the trigger events in the system.

Trigger events — SCPI command set

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *TRG • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Trigger events — TSP command set

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *TRG • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Using the TRIGGER key to generate an event

You can use the front-panel TRIGGER key to generate a trigger event.

To set a trigger block to respond to the front-panel key press, in SCPI, set the event to `DISPlay`. In TSP, set the event to `trigger.EVENT_DISPLAY`.

For example, if you set a wait block to advance when the TRIGGER key is pressed, the trigger model will reach the wait block. If the TRIGGER key has already been pressed, the trigger model execution will continue. If the TRIGGER key has not been pressed, the trigger model execution is halted until the TRIGGER key is pressed.

There are no action overruns for front-panel TRIGGER key events.

Using the notify block event

When the trigger model reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

You can define up to eight notify blocks in a trigger model. You can reference the event that the notify block generates by other commands to assign a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

When the trigger model executes a notify block, the instrument generates the SCPI event `NOTify<n>` or TSP event `trigger.EVENT_NOTIFYN`. You can assign this event to a command that takes an event. There can be up to eight notify blocks in a trigger model.

For example, if you want a Notify block to trigger a digital I/O line, insert a Notify block into the trigger model, assign it a notify event and then connect it to the stimulus of the digital I/O line to drive.

For example, to set notify block 5 to be a stimulus for digital output line 2, you would send the following commands in SCPI:

```
TRIG:BLOC:NOT 5, 2
:TRIG:DIG3:OUT:STIMulus NOTify2
```

In TSP, you would send the commands:

```
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

Respond to an event with a wait block

The wait building block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to three events for each wait block.

You can use the wait block to synchronize measurements with other instruments and devices.

Events that you can set the instrument to wait for include:

- Digital input/output signals, such as DB-9 and TSP-Link
- LAN events
- Blenders

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. When the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again.

The instrument clears the memory of the recorded event when the trigger model is at the start block and when the trigger model exits the wait block.

You can have up to eight wait blocks in a trigger model.

All items in the list are subject to the same action — you cannot combine `AND` and `OR` logic in a single command.

The events can be one of the events shown in the following table.

Using the branch-on-event trigger blocks

The branch-on-event building block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

The event can be one of the events shown in the following table.

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

To monitor for command interface trigger, use the event `trigger.EVENT_COMMAND` (TSP) or the event `COMMAND` (SCPI). To ensure that commands and triggers that are issued over the command interface are processed in the correct order, the instrument does not generate a trigger event until:

- The trigger command is executed
- TSP only: `trigger.wait()` retrieves the trigger command from the command queue before it would normally be executed

Command interface triggering does not generate action overruns. The triggers are processed in the order that they are received in the Model 2450 command queue. The Model 2450 only processes incoming commands when no commands are running. Unprocessed input triggers can cause an overflow in the command queue. It is important to make sure a script processes triggers while it is running.

NOTE

If you are using a test script using TSP, the command queue can fill up with trigger entries if 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 `trigger.wait()` calls that remove the `*TRG` messages from the command queue. If the command queue fills with too many trigger entries, messages such as `abort` are not processed.

Triggering using hardware lines

You can use the six digital I/O lines and three TSP-Link® synchronization lines to trigger the Model 2450. Both types of lines are configured and controlled the same way.

You can configure the Model 2450 to use the digital I/O to send signals to trigger external instruments. When you link these output triggers to the completion of certain source-measure actions, it enables hardware handshaking.

See Digital I/O (on page 3-62) and TSP-Link System Expansion Interface (on page 3-118) for more information about connections and direct control of the digital I/O and TSP-Link synchronization lines.

Triggering using TSP-Link synchronization lines

The Model 2450 has three synchronization lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

LAN triggering overview

You can send and receive triggers over the LAN interface. The Model 2450 supports LAN extensions for instrumentation (LXI). It has eight LAN triggers that generate and respond to LXI trigger packets.

Understanding hardware value and pseudo line state

LAN triggering and hardware synchronization are similar, except that LAN triggering uses LXI trigger packets instead of hardware signals. A bit in the LXI trigger packet called the hardware value simulates the state of a hardware trigger line. The Model 2450 stores the hardware value as the pseudo-line state. Only the state of the last LXI trigger packet that was sent or received is stored.

The stateless event flag is a bit in the LXI trigger packet that indicates if the hardware value should be ignored. If it is set, the Model 2450 ignores the hardware value of the packet and generates a trigger event. The Model 2450 always sets the stateless flag for outgoing LXI trigger packets. If the stateless event flag is not set, the hardware value indicates the state of the signal.

The instrument interprets changes in the hardware value of consecutive LXI trigger packets as edge transitions. Edge transitions generate trigger events. If the hardware value does not change between successive LXI trigger packets, the Model 2450 assumes an edge transition was missed and generates a trigger event. The following table shows edge detection in LAN triggering.

LXI trigger edge detection

Stateless event flag	Hardware value	Pseudo-line state	Falling edge	Rising edge
0	0	0	Detected	Detected
0	1	0	-	Detected
0	0	1	Detected	-
0	1	1	Detected	Detected
1	-	-	Detected	Detected

You can set the LAN trigger edge detection method in incoming LXI trigger packets. The edge that is selected also determines the hardware value in outgoing LXI trigger packets. The following table lists the LAN trigger edges.

Trigger mode	Input detected	Output generated
Either edge	Either	Negative
Falling edge	Falling	Negative
Rising edge	Rising	Positive

The example below illustrates how to configure the LAN trigger edge.

```
-- Set LAN trigger 2 to detect the falling edge.
trigger.lanin[2].edge = trigger.EDGE_FALLING
```

LAN trigger objects generate LXI trigger events, which are LAN0 to LAN7 (zero based). To specify the LAN trigger event in a command, use `trigger.EVENT_LANN`, where *N* is 1 to 8. `trigger.EVENT_LAN1` corresponds to LXI trigger event LAN0 and `trigger.EVENT_LAN8` corresponds to LXI trigger event LAN7.

Generate LXI trigger packets

You can configure the Model 2450 to output an LXI trigger packet to other LXI instruments.

To generate LXI trigger packets:

1. Call the `trigger.lanout[N].connect()` function.
2. Select the event that triggers the outgoing LXI trigger packet by assigning the specific event to the LAN stimulus input.

Make sure to use the same LXI domain on both the Model 2450 instrument and the other instrument. If the Model 2450 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.

Event blenders

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

There are 1 or 2 event blenders that you can use to monitor and respond to multiple stimulus events. Each event blender can be configured to monitor a maximum of four different trigger events.

Event blender operations

You can use event blenders to perform logical AND or logical OR operations on trigger events. For example, trigger events can be triggered when either a manual trigger or external input trigger is detected.

When AND operation is selected, the event blender generates an event when an event is detected on all of the assigned stimulus inputs.

When OR operation is selected, the event blender generates an event when an event is detected on any one of the four stimulus inputs.

You set the event blender operation using remote commands.

Using SCPI commands:

Send the command `:TRIGger:BLENder<n>:MODE`.

Set the command to `OR` or `AND`.

Using TSP commands:

Send the command `trigger.blender[N].orenable`.

Setting the command to `true` enables OR operation; setting it to `false` enables AND operation.

Assigning blender trigger events

Each event blender has four stimulus inputs. You can assign a different trigger event to each stimulus input.

You set the event trigger events operation using remote commands. See the command descriptions for the list of events that you can assign.

Using SCPI commands:

Send the command `:TRIGger:BLENder<n>:STIMulus<m>`.

Using TSP commands:

Send the command `trigger.blender[N].orenable`.

Trigger blender action overruns

The event blenders can generate action overruns.

When the event blender operation is set to AND, overruns occur when a second event on any of its inputs is detected before an output event is generated.

When the operation is set to OR, overruns occur when two events are detected simultaneously.

Use the status model to monitor for the occurrence of action overruns (for details, see the Status model (on page C-1)).

Trigger timers

You can use trigger timers to add delays, start measurements, and change the source value at timed intervals. The Model 2450 has 1 to 4 independent timers.

Timer attributes

Each timer has attributes that you can configure. These attributes are described in the following sections.

Count

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

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

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

Timer delays

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

A delay is the period of time after the timer is triggered and before the timer generates a trigger event. The programming example below illustrates how to configure timer 3 for a 10-second delay:

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

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 in the list. The timer repeats the delay list after all of the elements in the delay list have been used. The programming example below illustrates how to configure timer 3 for delays of 2, 10, 15, and 7 seconds:

```
-- Configure timer 3 to complete delays of 2 seconds, 10 seconds,
-- 15 seconds, and 7 seconds.
trigger.timer[3].delaylist = {2, 10, 15, 7}
```

NOTE

Assigning a value to the delay attribute is the same as configuring it with a one-element delay list.

Generate

```
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.
```

Timer action overruns

The timer generates an action overrun when it generates a trigger event while a timer delay is still in progress. Use the status model to monitor for the occurrence of action overruns (for details, see the Status model (on page C-1)).

Using trigger timers with timing building blocks

For precise timing or if you need to synchronize timing with other execution blocks or events, you can use the SCPI or TSP trigger timer commands with trigger model wait blocks and notify blocks. You can use the trigger timer commands to add small precise delays or to start measurements or to overcome variable measurement delays. The Model 2450 has 1 to 4 independent timers.

For example, you can use a trigger timer to control the delay between non-sequential blocks. After creating a trigger timer, you can insert a notify block to start the timer at a specific point in the trigger model. You could then add a wait block to wait for the timer to expire.

Another example is a measure block that takes a variable amount of time. To ensure a precise time between measurements, you can create a trigger timer and define it to be a fixed interval that is longer than the longest possible measurement. Then you can set up the trigger model to include:

- A notify block that starts the trigger timer
- A measure block that makes a measurement
- A wait block that waits for the timer to expire
- A branch counter block that iterates some number of times

NOTE

Some attributes of trigger timers should not be used with the trigger model. Attributes you should not set are:

- Count value of 0 (resulting in generation of trigger events indefinitely)
- Delay lists

SCPI trigger timer commands:

- :TRIGger:TIMer<n>:CLEar (on page 6-166)
- :TRIGger:TIMer<n>:COUNt (on page 6-167)
- :TRIGger:TIMer<n>:DELay (on page 6-168)
- :TRIGger:TIMer<n>:STARt:FRACtional (on page 6-169)
- :TRIGger:TIMer<n>:STARt:GENerate (on page 6-170)
- :TRIGger:TIMer<n>:STARt:OVERrun? (on page 6-170)
- :TRIGger:TIMer<n>:STARt:SEConds (on page 6-171)
- :TRIGger:TIMer<n>:STARt:STIMulus (on page 6-171)
- :TRIGger:TIMer<n>:STATe (on page 6-173)

TSP trigger timer commands:

- trigger.timer[N].clear() (on page 8-216)
- trigger.timer[N].count (on page 8-216)
- trigger.timer[N].delay (on page 8-217)
- trigger.timer[N].delaylist (on page 8-217)
- trigger.timer[N].enable (on page 8-219)
- trigger.timer[N].reset() (on page 8-220)
- trigger.timer[N].start.fractionalseconds (on page 8-221)
- trigger.timer[N].start.generate (on page 8-221)
- trigger.timer[N].start.overrun (on page 8-222)
- trigger.timer[N].start.seconds (on page 8-222)
- trigger.timer[N].start.stimulus (on page 8-223)
- trigger.timer[N].wait() (on page 8-224)

TSP-Link System Expansion Interface

Keithley Instruments TSP-Link® is a high-speed trigger synchronization and communication bus that test system builders can use to connect multiple instruments in a master and subordinate configuration. Once connected, all the instruments that are equipped with TSP-Link in a system can be programmed and operated under the control of the master instrument or instruments. The test system can have multiple master and subordinate groups, which can be used to handle multi-device testing in parallel. Combining TSP-Link with a flexible programmable trigger model ensures speed and accuracy.

Using TSP-Link, multiple instruments are connected and can be used as if they are part of the same physical unit for simultaneous multi-channel testing. The test system can be expanded to include up to 32 TSP-Link-enabled instruments.

TSP-Link functionality is only available when using the instrument front panel or the TSP commands to control the instrument. It is not available if you are using SCPI commands.

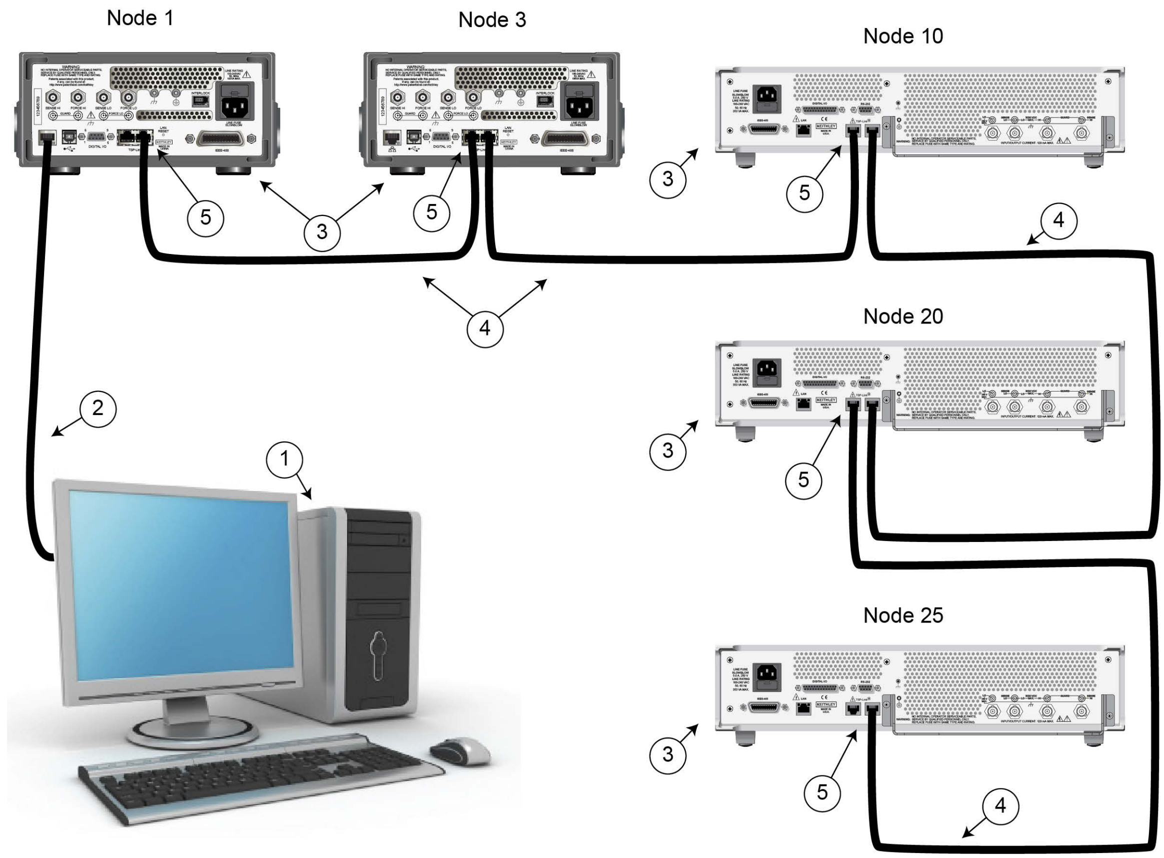
TSP-Link connections

The Model 2450 has three synchronization lines that are built into the TSP-Link connection. If you are using a TSP-Link network, you do not have to modify any connections.

Example connections for a TSP-Link system are shown in the following figure.

The TSP-Link connectors are on the rear panel of the instruments. All the instruments in the system are connected in a sequence (daisy-chained) using LAN crossover cables.

Figure 106: TSP-Link connections



Item	Description	Notes
1	Controller	Optional. A computer is not needed for stand-alone systems.
2	Communication connection	Optional. Connection from controller to the master node through GPIB, LAN, or USB. Details about these computer communication connections are described in Remote communication interfaces (on page 2-44).
3	Nodes	You can have up to 32 nodes on the TSP-Link system. Each node must have a unique node number from 1 to 64.
4	LAN crossover cable	Type 5e category or higher; 3 meters (9.8 feet) maximum between nodes. Available from Keithley Instruments (Model CA-180-3A).
5	TSP-Link connections	Each instrument has two TSP-Link connectors. You can make the connection to either TSP-Link connection.

TSP-Link nodes

Each instrument or enclosure attached to the TSP-Link expansion interface must be identified. This identification is called a TSP-Link node number, and the instruments or enclosures are called nodes. Each node must be assigned a unique node number.

An individual node is accessed as `node[N]`, where *N* is the node number assigned to the node. You can access all TSP-accessible remote commands as elements of the specific node. The following attributes are examples of items you can access:

- `node[N].model`: The product model number of the node.
- `node[N].version`: The product version of the node.
- `node[N].serialno`: The product serial number of the node.

Assigning node numbers

Each Model 2450 instrument is initially assigned as node 2. You can assign node numbers from 1 to 64. However, the system can only include 32 physical nodes.

The node number for each instrument is stored in its nonvolatile memory and remains in storage when the instrument is turned off.

You can assign a node number to an instrument using the front panel or by using a remote command.

To assign a node number using the front panel:

1. Press the **MENU** key.
2. Under System, select **Communication**. The SYSTEM COMMUNICATION window opens.
3. Select the **TSP-Link** tab.
4. Next to Node, set the TSP-Link address for this instrument.

To assign a node number using a remote command:

Set the `tsplink.node` attribute of the instrument:

```
tsplink.node = N
```

Where: `N = 1 to 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.

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.

A TSP-Link system can be stand-alone or computer-based.

In a stand-alone system, scripts are loaded into the instruments. You can run a script from the front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation. When the script is finished running, all the nodes in the system return to local operation, and the master/subordinate relationship between nodes is dissolved.

In a computer-based system, you can use a computer and a remote interface to communicate with a single node in the system. This node becomes the interface to the entire system. When a command is sent through this node, all nodes go into remote operation. The node that receives the command becomes the master and can control all 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. For more information about remote interfaces, see Remote communication interfaces (on page 2-44).

Initializing the TSP-Link system

TSP-Link® system must be initialized after configuration changes. Changes that require you to initialize the system include:

- Turning off power or rebooting any instrument in the system
- Changing node numbers on any instrument in the system
- Rearranging or disconnecting the TSP-Link cable connections between instruments

If initialization is not successful, you can check the event log to check for error messages that indicate the problem. Some typical problems include:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument performing the initialization
- One or more of the instruments in the system is turned off
- The actual number of nodes is less than the expected number

From the front panel:

1. Power on all instruments connected to the TSP-Link network.
2. Press the **MENU** key.
3. Under System, select **Communication**. The SYSTEM COMMUNICATION window opens.
4. Select the **TSP-Link** tab.
5. Select **Initialize**.

Using TSP commands:

To initialize the TSP-Link system, send the command:

```
tsplink.initialize()
```

To check the state of the TSP-Link system, send the command:

```
print(tsplink.state)
```

If initialization was successful, `online` is returned. If initialization was not successful, `offline` is returned.

Sending commands to TSP-Link nodes

You can send remote commands to any instrument on the TSP-Link system by adding `node[N].` to the beginning of the remote command, where *N* is the node number.

For example, to sound the beeper on node 10, you would send the command:

```
node[10].beeper.beep(2, 2400)
```

To send a command to the master, you can use the global variable `localnode` as an alias for the node entry. For example, if a script is running on node 5 (which would make node 5 the master), you can use `localnode` as an alias for `node[5]`. In this example, to access the product model number, you would send:

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

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()` or `tsplink.initialize()` command. The `tsplink.reset()` or `tsplink.initialize()` command reinitializes the TSP-Link network and may change the state of individual nodes in the system.

Use `node[N].reset()` or `localnode.reset()` to reset only one of the nodes. The other nodes are not affected. The following programming example shows this type of reset operation with code that is run on node 1.

```
-- Reset node 1 only.
node[1].reset()
-- Reset the node you are connected to (in this case, node 1).
localnode.reset()
-- Reset node 4 only.
node[4].reset()
```

Terminating scripts on the TSP-Link system

You can terminate a script that is executing on a TSP-Link system.

To terminate an executing script and return all nodes to local control, send the following command:

```
abort
```

This dissolves the master/subordinate relationships between nodes.

You can also abort an executing script and turn off the source outputs on all source-measure units in the TSP-Link system. To do this, press the OUTPUT ON/OFF switch on any instrument in the system.

Triggering using TSP-Link synchronization lines

The Model 2450 has three synchronization lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

Using TSP-Link synchronization lines for digital I/O

Each synchronization line is an open-drain signal. When using the TSP-Link® synchronization lines for digital I/O, any node that sets the programmed line state to zero (0) causes all nodes to read 0 from the line state. This occurs regardless of the programmed line state of any other node. See the table in the Digital I/O bit weighting topic for digital bit weight values.

Running simultaneous test scripts

Running test scripts simultaneously improves functional testing, provides higher throughput, and expands system flexibility. You can use TSP-Link and TSP scripting to run simultaneous test scripts. You can also manage the resources that are allocated to test scripts that are running simultaneously.

In addition, you can use the data queue to do real-time communication between nodes on the TSP-Link system.

To run test scripts simultaneously, you can set up your TSP-Link network in one of the following configurations:

- Multiple TSP-Link networks
- A single TSP-Link network with groups

Using groups to manage nodes on a TSP-Link system

TSP-Link groups allow each group to run a different test script simultaneously. This method requires one TSP-Link network and a single GPIB connection to the computer that is connected to the master node.

A group can consist of one or more nodes. You must assign group numbers to each node using remote commands. If you do not assign a node to a group, it defaults to group 0, which will always be grouped with the master node (regardless of the group to which the master node is assigned).

The following table shows an example of the functions of groups on a single TSP-Link network. Each group in this example runs a different test script than the other groups, which allows the system to run multiple tests simultaneously.

TSP-Link network group functions

Group number	Group members	Present function
0	Master node 1	Initiates and runs a test script on node 2 Initiates and runs a test script on node 6 In addition, the master node can execute scripts and process run commands
1	Group leader node 2	Runs the test script initiated by the master node Initiates remote operations on node 3 through node 5
	Node 3 through node 5	Performs remote operations initiated by node 2
2	Group leader node 6	Runs the test script initiated by the master node Initiates remote operations on node 7 through node *n*
	Node 7 through node *n*	Performs remote operations initiated by node 6

Master node overview

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 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 node's power is turned off, the group number for that node changes to 0.

The following example code dynamically assigns a node to a group:

```
-- Assign node 3 to group 1.
node[3].tsplink.group = 1
```

Running test scripts and programs on remote nodes

You can send the `execute()` command from the master node to initiate a test script and Lua code on a remote node. The `execute()` command places the remote node in the overlapped operation state. As a test script runs on the remote node, the master node continues to process other commands simultaneously.

Use the following code to send the `execute()` command for a remote node. The *N* parameter represents the node number that runs the test script (replace *N* with the node number).

To set the global variable "setpoint" on node N to 2.5:

```
node[N].execute("setpoint = 2.5")
```

The following code demonstrates how to run a test script that is defined on the local node. For this example, *scriptVar* is defined on the local node, which is the node that initiates the code to run on the remote node. The local node must be the master node.

To run scriptVar on node N:

```
node[N].execute(scriptVar.source)
```

The programming example below demonstrates how to run a test script that is defined on a remote node. For this example, *scriptVar* is defined on the remote node.

To run a script defined on the remote node:

```
node[N].execute("scriptVar()")
```

It is recommended that you copy large scripts to a remote node to improve system performance. See Copying test scripts across the TSP-Link network 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 will occur.

You can execute the `waitcomplete()` command on the master node or group leader to wait for overlapped operations. The action of `waitcomplete()` depends on the parameters specified.

If you want to wait for completion of overlapped operations for:

- **All nodes in the local group:** Use `waitcomplete()` without a parameter from the master node or group leader.
- **A specific group:** Use `waitcomplete(N)` with a group number as the parameter from the master node. This option is not available for group leaders.
- **All nodes in the system:** Use `waitcomplete(0)` from the master node. This option is not available for group leaders.

For additional information, see waitcomplete() (on page 8-252).

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 shows 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 as a way 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 completely separate copy of the data in the original table, with no references to the original table or any subtables.

You can access data from the data queue even if a remote group or a node has overlapped operations in process. See the `dataqueue` commands in the TSP command reference (on page 8-1) for more information.

Remote TSP-Link commands

Commands that control and access the TSP-Link® synchronization port are summarized in the following table. See the TSP command reference (on page 8-1) for complete details on these commands.

Use the commands in following table to perform basic steady-state digital I/O operations; for example, you can program the Model 2450 to read and write to a specific TSP-Link synchronization line or to the entire port.

TSP-Link commands

Command	Description
trigger.tsplinkin[N].clear() (on page 8-224)	Clears the event detector for a trigger
trigger.tsplinkin[N].edge (on page 8-225)	Indicates which trigger edge controls the trigger event detector for a trigger line
trigger.tsplinkin[N].overrun (on page 8-226)	Indicates if the event detector ignored an event while in the detected state
trigger.tsplinkin[N].wait() (on page 8-226)	Waits for a trigger
trigger.tsplinkout[N].assert() (on page 8-227)	Simulates the occurrence of the trigger and generates the corresponding trigger event
trigger.tsplinkout[N].logic (on page 8-228)	Defines the trigger output with output logic for a trigger line
trigger.tsplinkout[N].pulsewidth (on page 8-228)	Sets the length of time that the trigger line is asserted for output triggers
trigger.tsplinkout[N].release() (on page 8-229)	Releases a latched trigger on the given TSP-Link trigger line
trigger.tsplinkout[N].stimulus (on page 8-229)	Specifies the event that causes the synchronization line to assert a trigger
tsplink.group (on page 8-230)	The group number of the TSP-Link node
tsplink.initialize() (on page 8-231)	Initializes all instruments and enclosures in the TSP-Link system
tsplink.line[N].mode (on page 8-232)	Defines the trigger operation of a TSP-Link line as digital in or out or trigger in or out
tsplink.line[N].reset() (on page 8-233)	Resets some of the TSP-Link trigger attributes to their defaults
tsplink.line[N].state (on page 8-233)	Reads or writes the digital state of a TSP-Link synchronization line
tsplink.master (on page 8-234)	Reads the node number assigned to the master node
tsplink.node (on page 8-234)	Defines the node number
tsplink.readport() (on page 8-235)	Reads the TSP-Link synchronization lines as a digital I/O port
tsplink.state (on page 8-235)	Describes the TSP-Link online state
tsplink.writeport() (on page 8-236)	Writes to all TSP-Link synchronization lines as a digital I/O port

TSP-Link synchronization programming example

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

```
tsplink.line[1].mode = tsplink.MODE_DIGITAL_OPEN_DRAIN
-- Set bit B1 high.
tsplink.line[1].state = 1
-- Read I/O port.
data = tsplink.readport()
print(data)
```

The output would be similar to:

```
7
```

To read bit B1 only:

```
-- To read bit B1 only
data = tsplink.line[1].state
print(data)
```

The output would be similar to:

```
tsplink.STATE_HIGH
```

Using Model 2450 TSP-Link commands with other TSP-Link products

If you are connecting the Model 2450 in a system with other TSP-Link products, be aware that some of the TSP-Link commands may be different. You can use the earlier versions of the commands, but be aware that they may not be supported in future versions of the product.

Commands that are the same in all TSP-Link products:

- `tsplink.group`
- `tsplink.master`
- `tsplink.node`
- `tsplink.readport()`
- `tsplink.state`
- `tsplink.writeport()`

Model 2450 TSP-Link command	Replaces this command in other TSP-Link products
`trigger.tsplinkin[N].clear()`	`tsplink.trigger[N].clear()`

`trigger.tsplinkin[N].edge`
`trigger.tsplinkout[N].logic`
`tsplink.line[N].mode`
	`tsplink.trigger[N].mode`

`trigger.tsplinkin[N].overrun`	`tsplink.trigger[N].overrun`
`trigger.tsplinkin[N].wait()`	`tsplink.trigger[N].wait()`
`trigger.tsplinkout[N].assert()`	`tsplink.trigger[N].assert()`
`trigger.tsplinkout[N].pulsewidth`	`tsplink.trigger[N].pulsewidth`
`trigger.tsplinkout[N].release()`	`tsplink.trigger[N].release()`
`trigger.tsplinkout[N].stimulus`	`tsplink.trigger[N].stimulus`
`tsplink.initialize()`	`tsplink.reset()`
`tsplink.line[N].reset()`	`tsplink.trigger[N].reset()`
`tsplink.line[N].state`	`tsplink.readbit()` `tsplink.writebit()`
Not applicable	`tsplink.writeprotect`

TSP-Net

TSP-Net provides a simple socket-like programming interface to Test Script Processor (TSP) enabled instruments. Using the TSP-Net library, the Model 2450 can control ethernet-enabled devices directly through its LAN port. This enables the Model 2450 to communicate directly with a device that is that is not TSP-enabled without the use of a controlling computer.

Using TSP-Net library methods, you can transfer string data to and from a remote instrument, transfer and format data into Lua variables, and clear input buffers. The TSP-Net library is only accessible using commands from a remote command interface when you are using the TSP command language.

While you can use TSP-Net commands to communicate with any ethernet-enabled instrument, specific TSP-Net commands exist for TSP-enabled instruments to allow for support of features unique to the TSP scripting engine. These features include script downloads, reading buffer access, wait completion, and handling of TSP scripting engine prompts.

Using TSP-Net commands with TSP-enabled instruments, a Model 2450 can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The Model 2450 can read the data from the remote instrument and either manipulate the data or send the data to a different remote instrument on the LAN.

You can use TSP-Net to connect to a computer; you can use a script on the instrument to transfer data directly to your computer hard drive.

With TSP-Net, you can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the Model 2450.

Using TSP-Net with any ethernet-enabled instrument

NOTE

Refer to TSP command reference (on page 8-1) for details about the commands presented in this section.

The Model 2450 LAN connection 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 2450 to an ethernet device or to a hub.

To set up communication to a remote ethernet-enabled instrument that is TSP® enabled:

1. Send the following command to configure TSP-Net to send an abort command when a connection to a TSP instrument is established:

```
tspnet.tsp.abortonconnect = 1
```

   If the scripts are allowed to run, the connection is made, but the remote instrument may be busy.

2. Send the command:

```
connectionID = tspnet.connect(ipAddress)
```

   Where:

   - `connectionID` is the connection ID that will be used as a handle in all other TSP-Net function calls.
   - `ipAddress` is the IP address of the remote instrument.

See tspnet.connect() (on page 8-237) for additional detail.

To set up communication to a remote ethernet-enabled device that is not TSP enabled:

1. Send the command:

```
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

   Where:

   - `connectionID` is the connection ID that will be used as a handle in all other `tspnet` function calls.
   - `ipAddress` is the IP address 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 8-237) for additional detail.

To communicate to a remote ethernet device from the Model 2450:

1. Connect to the remote device using one of the above procedures. If the Model 2450 cannot make a connection to the remote device, it generates a timeout error. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 seconds.
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 2450 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 2450 suspends operation until the remote device responds or a timeout error 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:

```
-- Disconnect all existing TSP-Net connections.
tspnet.reset()
-- Set tspnet timeout to 5 seconds.
tspnet.timeout = 5
--  Establish connection to another device with IP address 192.168.1.51
-- at port 1394.
id_instr = tspnet.connect("192.168.1.51", 1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set 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, "*idn?" .. "\r\n")
-- Read the data available, then print it.
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

Remote instrument errors

If the Model 2450 is connected to a TSP-enabled instrument through TSP-Net, all errors that occur on the remote instrument are transferred to the event log of the Model 2450. The Model 2450 indicates events from the remote instrument by prefacing these events with “Remote Error.” For example, if the remote instrument generates error code 4909, "Reading buffer not found within device," the Model 2450 generates the error string “Remote Error: (4909) Reading buffer not found within device.”

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

tspnet.clear() (on page 8-236)
tspnet.connect() (on page 8-237)
tspnet.disconnect() (on page 8-238)
tspnet.execute() (on page 8-239)
tspnet.idn() (on page 8-240)
tspnet.read() (on page 8-241)
tspnet.readavailable() (on page 8-242)
tspnet.reset() (on page 8-243)
tspnet.termination() (on page 8-243)
tspnet.timeout (on page 8-244)
tspnet.write() (on page 8-248)

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

tspnet.tsp.abort() (on page 8-245)
tspnet.tsp.abortonconnect (on page 8-245)
tspnet.tsp.rbtablecopy() (on page 8-246)
tspnet.tsp.runscript() (on page 8-247)

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 4

Source-measure considerations

In this section:

Circuit configurations

You can measure current or voltage with either type of source.

The fundamental source-measure configurations for the Model 2450 are described in the following section.

Source current

When you configure the instrument to source current, the instrument functions as a high-impedance current source that can limit voltage and can measure current or voltage.

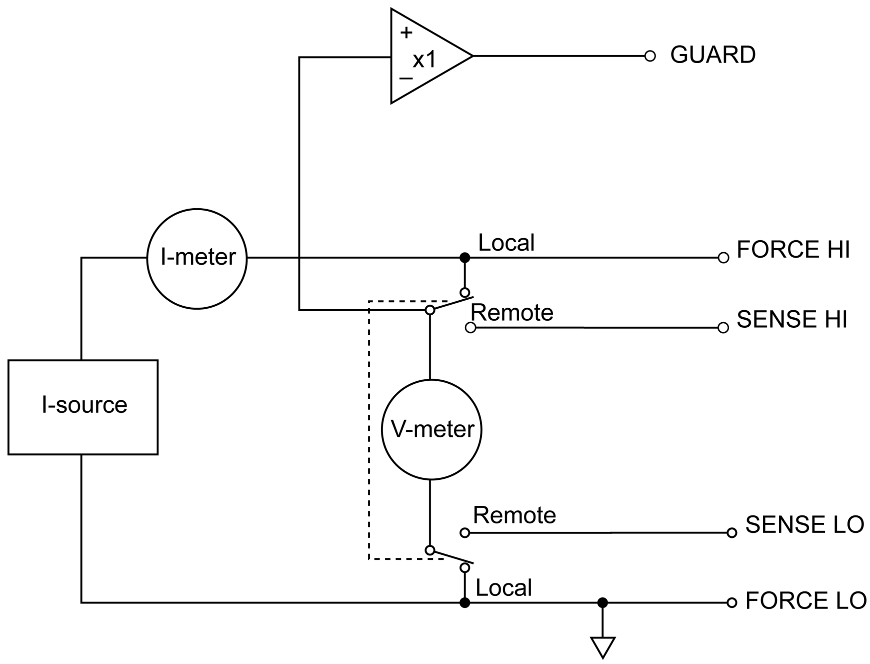
If you set the instrument to 2-wire sense, voltage is measured at the FORCE HI and FORCE LO terminals of the instrument. If you set the instrument to 4-wire sense, voltage is measured directly at the device under test using the sense terminals. Four-wire sense eliminates any voltage drops that may be in the test leads or in the connections between the instrument and the device under test.

The current source does not use the sense leads to enhance current source accuracy. However, if the instrument is in 4-wire sense, the instrument may reach limit levels if you disconnect the sense leads. When 4-wire sense is selected, you must connect the sense leads. If the sense leads are not connected, incorrect operation will result.

If you are sourcing and measuring the same function (for example, sourcing current and measuring current), the measurement range is the same as the source range. This feature is valuable if you are operating when the source limit has been exceeded. When the source limit has been exceeded, the programmed source value is not reached. Thus, measuring the source lets you measure the actual output level. You can also use the source readback function to measure the source. See Source readback (on page 2-110) for information.

You can set overvoltage protection if there is potential for disconnection of the sense leads. For more information on overvoltage protection, see Overvoltage protection (on page 2-98).

Figure 107: Source current configuration



Source voltage

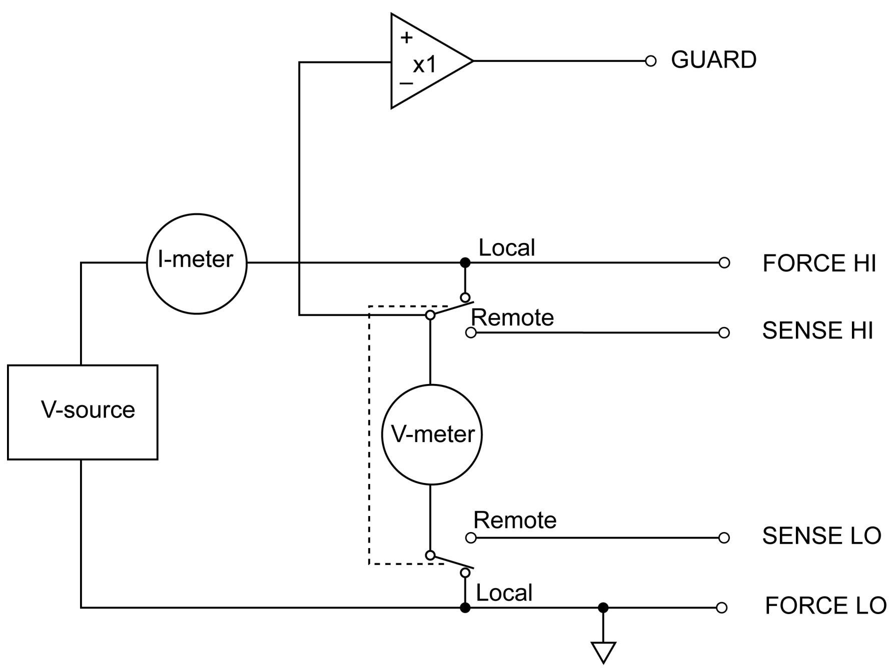
When you configure the instrument to source voltage, it functions like a low-impedance voltage source that can limit current. The instrument can measure current or voltage. This configuration is shown in the figure below.

Sense circuitry continuously monitors the output voltage and makes adjustments to the voltage source as needed. The voltmeter senses the voltage and compares it to the programmed voltage level. If the sensed level and the programmed value are not the same, the source voltage is adjusted accordingly.

If you set the instrument to 2-wire sense, the voltmeter senses the voltage at the FORCE HI and FORCE LO terminals. If it is set to 4-wire sense, the voltmeter sense the voltage at the device under test. Four-wire sense eliminates the effect of voltage drops in the test leads, ensuring that the exact programmed voltage is applied to the device under test.

If you are sourcing and measuring the same function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range. You can use this feature when operating when source limits have been exceeded. When the source limits have been exceeded, the programmed source value is not reached. Thus, measuring the source lets you measure the actual output level. You can also use the source readback function to measure the actual output level. See Source readback (on page 2-110) for information.

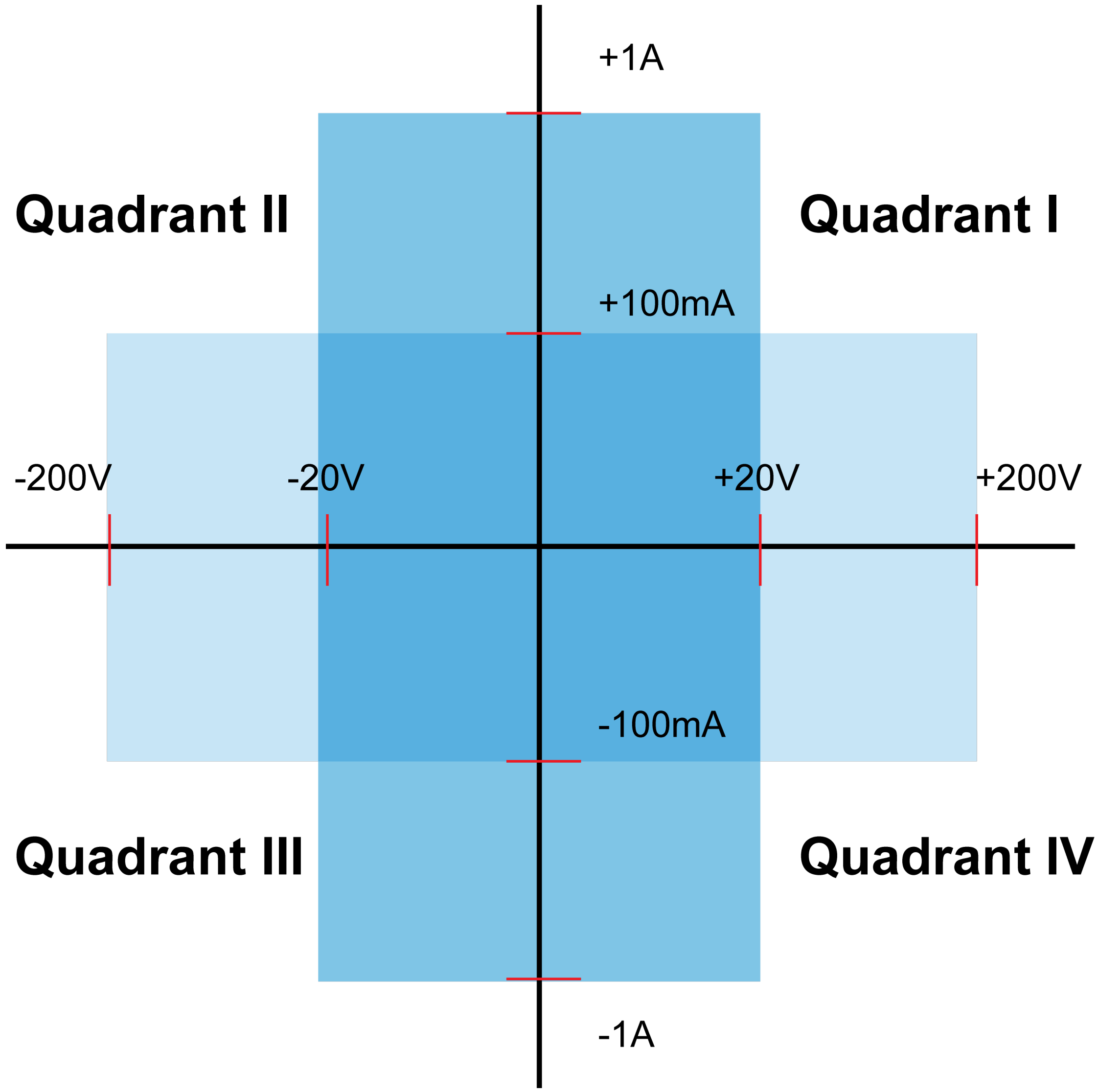
Figure 108: Voltage source configuration



Operating boundaries

Operating boundaries define the current and voltage limits of the instrument. The general operating boundaries of the Model 2450 are shown in the following figure.

Figure 109: Model 2450 operating boundaries



If the voltage or current exceeds the limits, the instrument limits the source voltage to keep operating currents and voltages in these boundaries. You can set operating limits to restrict current or voltage more tightly than the operating boundary limits. In this drawing, the magnitudes are nominal values. The specific maximum output magnitudes of the instrument are defined in the specifications. Also note that the boundaries are not drawn to scale.

These operating boundaries are valid only if the instrument is being operated in an environment where the ambient temperature is 30 °C (86 °F) or less. Above 30 °C, high power operation could overheat the instrument, causing the output to turn off.

The four quadrants of the operating boundaries are defined as I, II, III, and IV. The Model 2450 can operate in any of the four quadrants.

When the instrument is operating in quadrant I or III, the instrument is a source, which means that voltage and current have the same polarity. As a source, the instrument is delivering power to a load.

When the instrument is operating in quadrant II or IV, the instrument is operating as a sink, which means that voltage and current have opposite polarity. As a sink, the instrument dissipates the power internally. An external source or an energy storage device, such as a battery, solar cell, or power supply, can force operation in the sink region. The ability of the instrument to dissipate power is defined by the boundaries shown in the following figure.

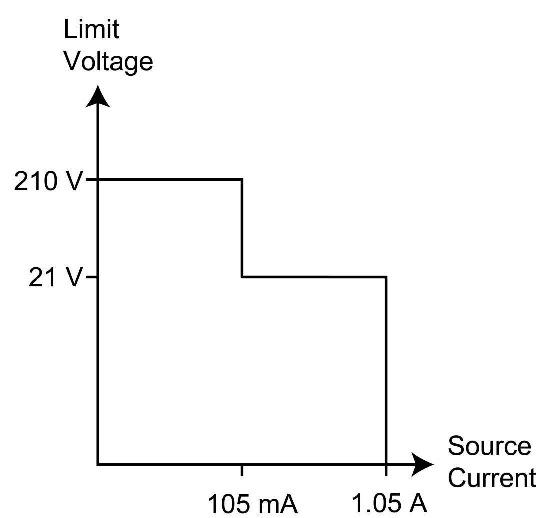
Current source operating boundaries

The operating boundaries for the current source are determined by the source range and limit settings. The operating boundary is the lower of the two settings. For example, if the 100 mA current source range is selected, the current source is limited to 105 mA, even if the source limit is set to 1 A. The voltage limit line represents the actual limit that is in effect. These limit lines are boundaries that represent the operating limits of the instrument for this quadrant of operation.

The operating point can be anywhere in or on these limit lines. The figure below shows operating boundaries for the current source for quadrant I. Operation in the other three quadrants is similar.

The current source line is the maximum source value possible for the presently selected current source range.

Figure 110: Model 2450 current source output characteristics

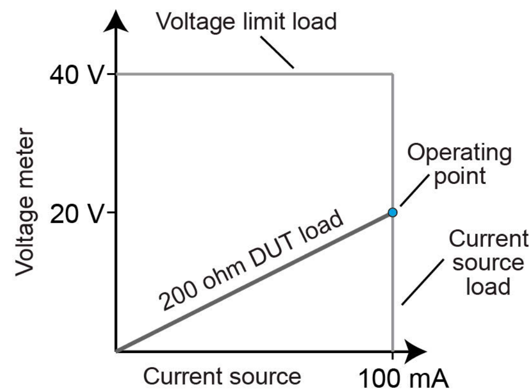


Voltage limit boundary examples

The actual boundaries where the instrument operates depends on the device under test (DUT) that is connected to the output of the instrument.

The following graphs show operation with the instrument set to source 100 mA with a limit of 40 V. In this graph, the resistive load is 200 Ω. The instrument is sourcing 100 mA to the 200 Ω load and subsequently measures 20 V. The load for 200 Ω intersects the 100 mA current source at 20 V.

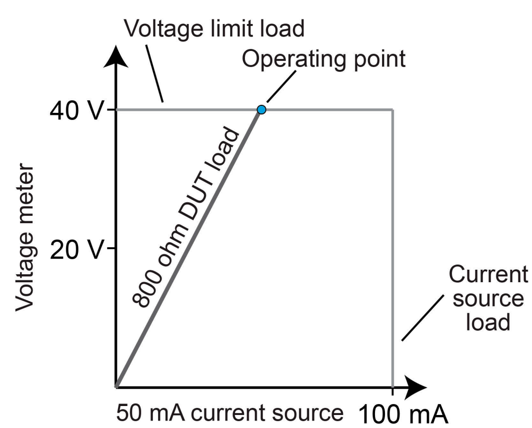
Figure 111: Model 2450 limit boundary example — normal



Voltage meter =	Current source * DUT load
=	(100 mA)(200 Ω)
=	20 V

In the following graph, the resistive load is increased to 800 Ω. The DUT load for 800 Ω intersects the voltage limit, which causes the instrument to limit the current that it is sourcing. For the 800 Ω DUT, the instrument will only output 50 mA at the 40 V limit.

Figure 112: Model 2450 limit boundary example when limited



$$\text{Current source} = \frac{\text{Voltage meter}}{\text{DUT load}}$$

$$= \frac{40\text{ V}}{800\ \Omega}$$

$$= 50\text{ mA}$$

Notice that as resistance increases, the slope of the DUT load increases. As resistance approaches infinity (open output), the instrument sources virtually 0 mA at 40 V. Conversely, as resistance decreases, the slope of the DUT load decreases. At zero resistance (shorted output), the instrument sources 100 mA at virtually 0 V. Regardless of the load, voltage will never exceed the limit of 40 V.

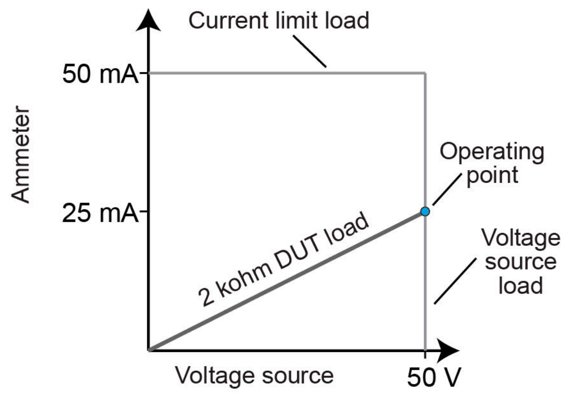
Current limit boundary examples

The actual boundaries where the instrument operates depends on the load (DUT) that is connected to the output of the instrument.

The following graphs show operation with the instrument set to source of 50 V with a limit of 50 mA.

In this graph, the resistive load is 2 kΩ. The instrument is sourcing 50 V to the 2 kΩ load and subsequently measures 25 mA. The load for 2 kΩ intersects the 50 V voltage source at 25 mA.

Figure 113: Model 2450 current limit boundary example normal



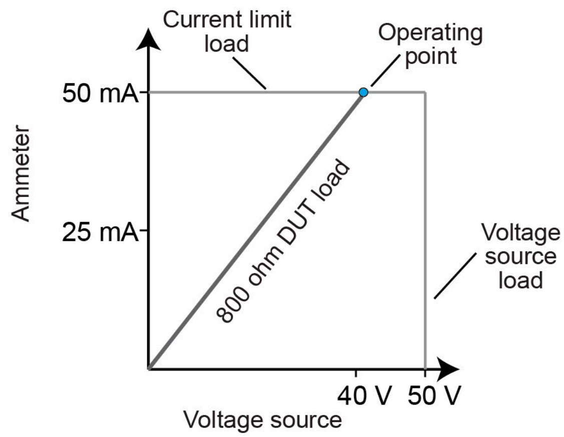
$$\text{Current meter} = \frac{\text{Voltage source}}{\text{DUT load}}$$

$$= \frac{50\ \text{V}}{2\ \text{k}\Omega}$$

$$= 25\ \text{mA}$$

In the following graph, the resistive load is increased to 800 Ω. The DUT load for 800 Ω intersects the current limit, which causes the instrument to limit the voltage that it is sourcing. For the 800 Ω DUT, the instrument will only output 40 V at the 50 mA limit.

Figure 114: Model 2450 current limit boundary example limited



Voltage source =	Current * DUT resistance
=	(50 mA)(800 Ω)
=	40 V

Notice that as resistance decreases, the slope of the DUT load increases. As resistance approaches infinity (open output), the instrument will source virtually 50 V at 0 mA. Conversely, as resistance increases, the slope of the DUT load decreases. At zero resistance (shorted output), the instrument will source virtually 0 V at 50 mA. Regardless of the load, current will never exceed the limit of 50 mA.

Using NPLCs to adjust speed and accuracy

You can adjust the amount of time that the input signal is measured. This time affects the usable measurement resolution, the amount of reading noise, and the reading rate of the instrument.

The amount of time is specified in parameters that are based on the number of power line cycles (PLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50).

The shortest amount of time (0.01 PLC) results in the fastest reading rate, but increases reading noise and decreases the number of usable digits.

The longest amount of time (10 PLC) provides the lowest reading noise and more resolution, but has the slowest reading rate.

Settings between the fastest and slowest number of PLCs are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

NOTE

The speed setting affects the normal mode rejection ratio (NMRR) and common mode rejection ratio (CMRR). Normal mode noise is the noise signal between the HI and LO terminals; common-mode noise is the noise signal between LO and chassis ground. See the Model 2450 specification for NMRR and CMRR values at different PLC settings.

NOTE

After adjusting the speed, you might want to adjust the number of digits that are displayed for measurements. See Setting the number of displayed digits (on page 2-37).

To set NPLC using the front panel:

1. Press the **FUNCTION** key.
2. Select the source and measurement combination.
3. From the Home page, swipe to display the SETTINGS screen.
4. Next to NPLCs, select the number. The number pad dialog box is displayed.
5. Enter the value.
6. Select **OK**.

NOTE

You can also set the speed by pressing the **MENU** key. Under Measure, select **Settings**, and then select the value next to NPLCs.

Using SCPI commands:

To set the number of PLCs for current measurements, send the command:

```
:SENSe:CURRent:NPLCycles <n>
```

To set the number of PLCs for resistance measurements, send the command:

```
:SENSe:RESistance:NPLCycles <n>
```

To set NPLCs for voltage measurements, send the command:

```
:SENSe:VOLTage:NPLCycles <n>
```

Where `<n>` is a value from 0.01 to 10, with 0.01 resulting in the fastest reading rates and 10 resulting in the lowest reading noise.

For example, to set NPLC for resistance measurements to 0.5, send the command

```
RES:NPLC 0.5
```

Using TSP commands:

To set NPLC, send the command `smu.measure.nplc`. For example, to set the NPLC value to 0.5 for voltage measurements, send the commands:

```
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.nplc = 0.5
```

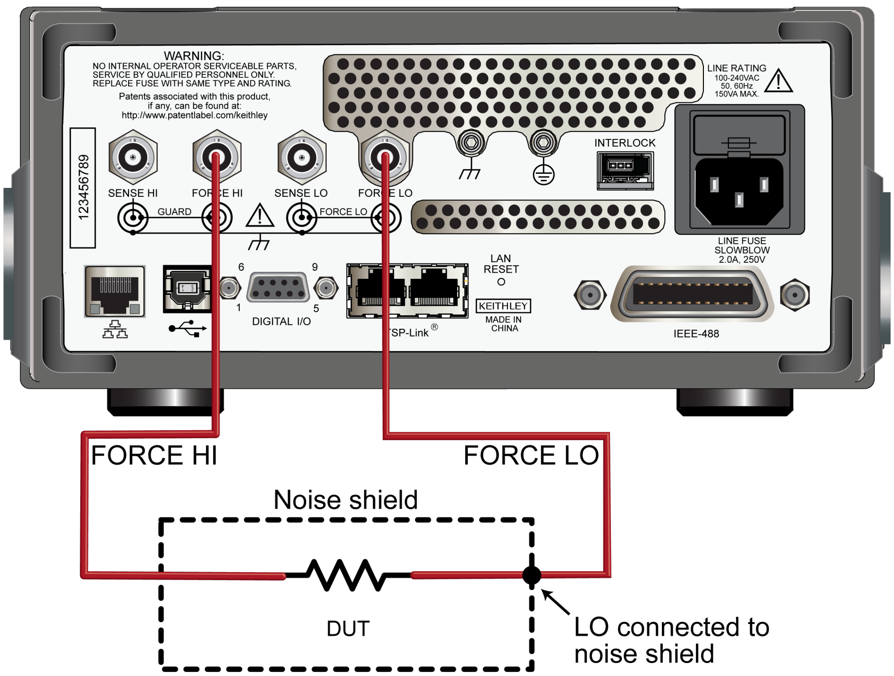
To assign a different measurement function, replace `smu.FUNC_DC_VOLTAGE` with one of the following:

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

Noise shield

Use a noise shield to prevent the introduction of unwanted signals into the test circuit. Low-level signals may benefit from effective shielding. The metal noise shield surrounds the test circuit and should be connected to LO, as shown.

Figure 115: Model 2450 Rear Panel Noise Shield Connections



Safety shield

⚠ WARNING

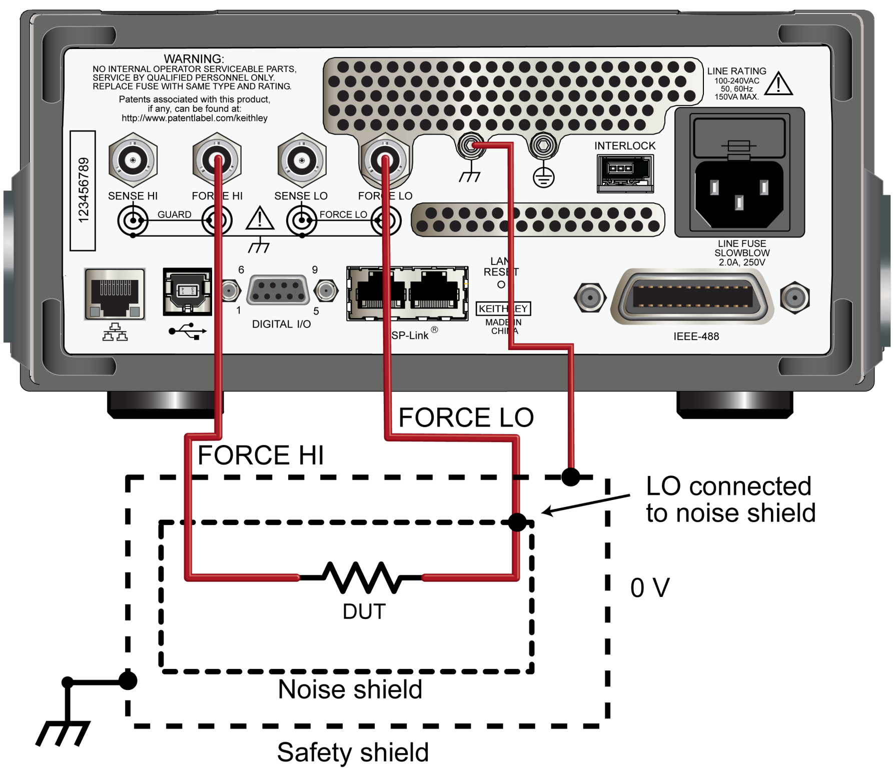
A safety shield must be used whenever hazardous voltages (>30 V RMS, 42 V peak) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the Model 2450 in a test circuit without a properly installed and configured safety shield.

The safety shield can be metallic or nonconductive, and must completely surround the DUT test circuit. A metal safety shield must be connected to a known protective earth (safety ground). See Test fixtures (on page 2-81) for important safety information on the use of a metal or a nonconductive enclosure.

Safety shielding

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

Figure 116: Model 2450 noise and safety shield



Grounding

Noise and chassis ground

Using the chassis as a ground point for signal connections to the Model 2450 chassis may result in different levels of noise, depending on your setup. If the Model 2450 common-mode current is channeled to the chassis instead of the device, the tie point to the chassis can help quiet measurements. However, if other equipment is connected to the chassis, you may have more noise because of other connected equipment.

If you choose to use the chassis as a ground point for signal connections, use the Model 2450 chassis screw as a connection point.

Floating the Model 2450

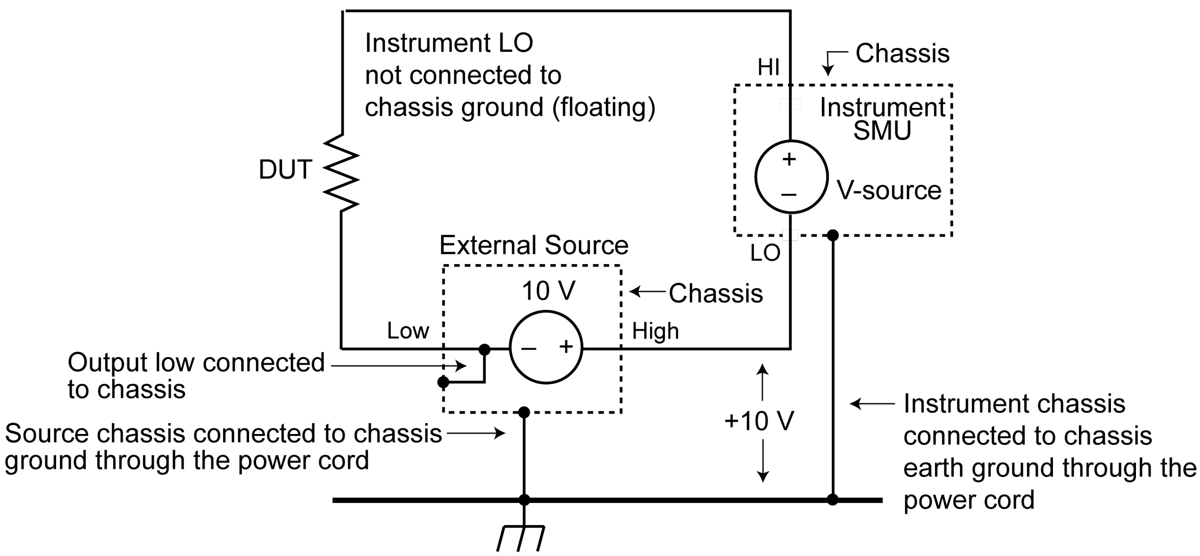
WARNING

INPUT/OUTPUT LO is not internally connected to the chassis and cannot be allowed to float above chassis ground more than the values shown on the front panel.

If you use an external source in the test system, you may need the Model 2450 to float off chassis earth ground. An example is shown below, which includes an external voltage source. Notice that output LO of the external voltage source is connected to chassis ground.

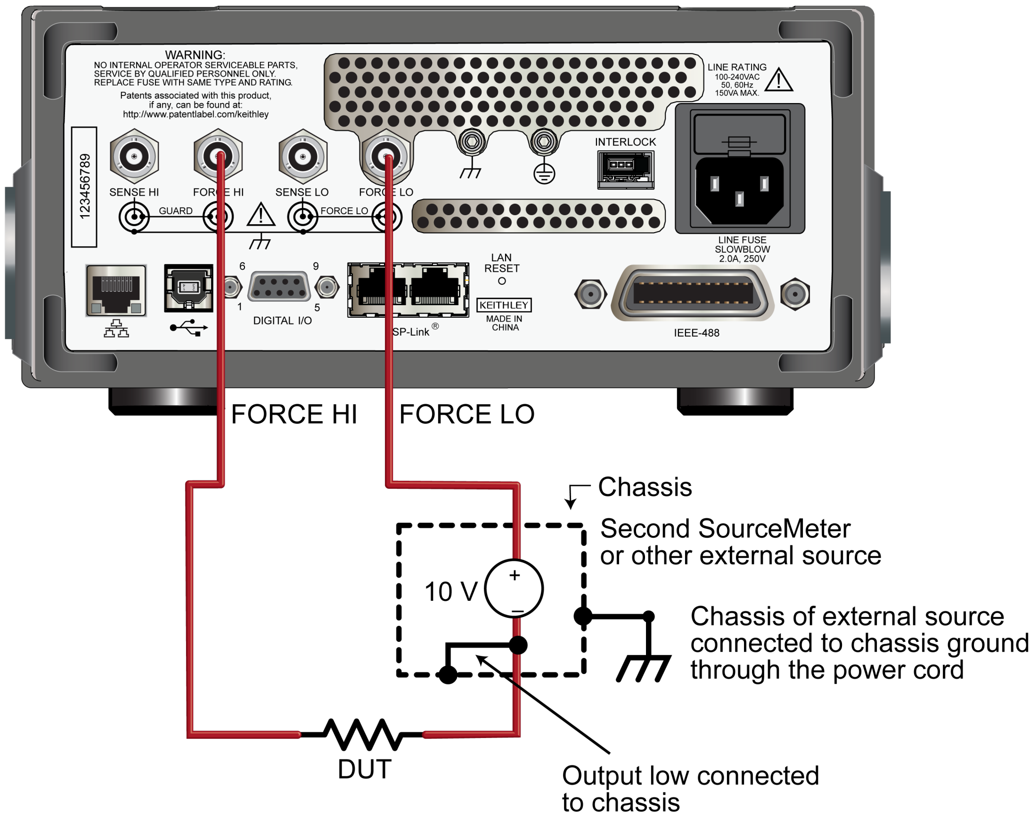
For the test circuit shown below, the Model 2450 must float off chassis ground. As shown, LO of the Model 2450 is floating +10 V above chassis earth ground. If LO of the Model 2450 was instead connected to chassis ground, the external voltage source would be shorted to the chassis ground.

Figure 117: Floating the instrument schematic



The connections for the floating configuration are shown below. To float the SMU, FORCE and SENSE LO must be isolated from chassis ground. To do this, do not connect FORCE and SENSE LO to chassis ground.

Figure 118: Connections for floating the instrument



The external voltage source can be a SMU of a second Model 2450 instrument or other instrument. Keep in mind that if the combined outputs of the sources exceeds 42 V, a safety shield is 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 **4-10**)****).***

Guarding

Guarding is an effective way to reduce the leakage current and capacitance that can exist between HI and LO. A guard is a low impedance point in the circuit that is at nearly the same potential as the high impedance lead that is being guarded. Use guarding when you are sourcing or measuring low current (less than 1 μA) or when test circuit impedance is more than 1 GΩ. Also use guard in noisy environments.

The rear panel of the Model 2450 includes an approximately 10 Ω driven guard at the SENSE HI and FORCE HI connections. This guard is always enabled and provides a buffered voltage. For 2-wire measurements, guard is at the same level as the FORCE HI voltage. For 4-wire measurements, it is at the same level as the SENSE HI voltage.

To use the built-in guards of the Model 2450, you must use the rear-panel triaxial connections. There are no guards available on the front panel.

WARNING

Guard is at the same potential as output HI. Therefore, if hazardous voltages are present at output HI, they are also present at the GUARD terminal. Failure to heed this warning may result in personal injury or death due to electric shock.

Using guard with a test fixture

A test fixture is typically used when testing high-impedance devices. The test fixture reduces noise and protects users from a potentially hazardous voltage on the guard shield.

To extend the guard to a test fixture, use a safety banana plug. Inside the test fixture, the guard can be connected to a guard plate or shield that surrounds the DUT.

Connect the test fixture chassis to LO to reduce noise.

WARNING

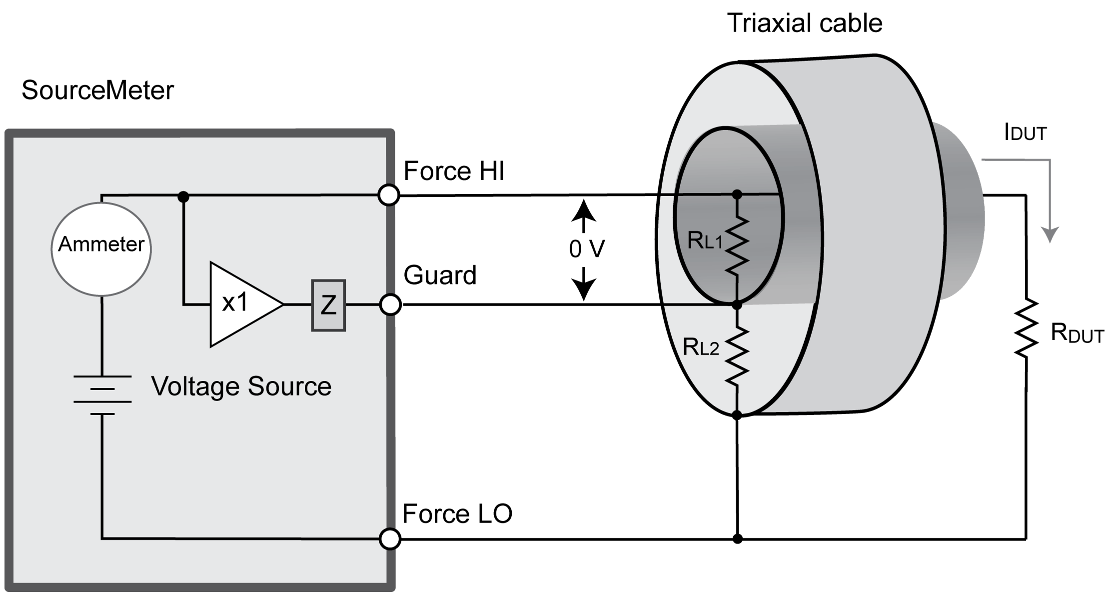
A safety shield must be used whenever hazardous voltages (>30 V RMS, 42 V peak) will be present in the test circuit. To prevent electrical shock that could cause injury or death, never use the Model 2450 in a test circuit that may contain hazardous voltages without a properly installed and configured safety shield.

Inside the test fixture, a triaxial cable can be used to extend guard to the DUT. See the connection diagrams on the rear panel of the instrument for the triaxial guard to conductor connections.

Guard circuit drawing

In the following schematic, note that guarding eliminates resistance leakage current (R_{L1}). The current flowing in resistance leakage 2 (R_{L2}) is supplied by the guard and does not affect the DUT current (I_{DUT}).

Figure 119: Guarded configuration



Using the Model 2450 in a custom test fixture or system

You can use the Model 2450 safely with a properly designed custom test fixture or with a semiconductor device prober or handler.

WARNING

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.

Connecting the Model 2450 output to a custom test fixture

Keithley Instruments offers several accessories that can help you when building a custom test fixture or system.

You can use high voltage coaxial or triaxial connectors on your test fixture. The following sections show you how to connect from the Model 2450 to the custom fixture and also how to make connections to the device inside the custom fixture.

For the Sense LO and Force LO terminals, the center conductor and first shield should be within a few volts of each other to guarantee normal operation.

Figure 120: Model 2450 Sense LO and Force LO terminal

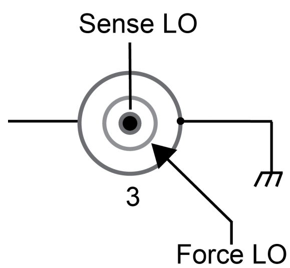
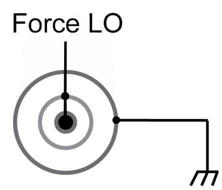


Figure 121: Model 2450 Force LO terminal



WARNING

Connections to LO on the Model 2450 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.

Sink operation

When the Model 2450 is operating as a sink, voltage and current have opposite polarities and the instrument is dissipating power rather than sourcing it. The instrument can be forced into sink operation by an external source, such as a battery, or an energy storage device, such as a capacitor. For detail on the sink region, see Operating boundaries (on page 4-4).

For example, if a 12 V battery is connected to the voltage source (HI to battery high) that is programmed for +10 V, sink operation occurs in the second quadrant (source +V and measure –I).

⚠ CAUTION

Carefully consider and configure the output-off state, source, and limits before connecting the Model 2450 to a device that can deliver energy. Devices that can deliver energy include voltage sources, batteries, capacitors, and solar cells. Configure instrument settings before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

When using the current source as a sink, always set the voltage limit and configure overvoltage protection (OVP) to levels that are higher than the external voltage level. Failure to do so could result in excessive current flow into the Model 2450 (<105 mA) and incorrect measurements.

When the instrument is operating as a sink and you set source or limit values that exceed the operating boundaries, the source limit is reached and the instrument . When the sink limit is reached, the source value turns yellow and the limit annunciator is active.

Battery charge and discharge

⚠ WARNING

To prevent personal injury or damage to the Model 2450, do not attempt to charge nonrechargeable batteries. Some of the batteries that can be charged with a Model 2450 are nickel cadmium (Ni-Cd), nickel metal hydride (Ni-MH), lithium ion (Li-ion), rechargeable alkaline, and lead acid. If you are working with a battery type that is not listed here, please contact your local Keithley representative or call one of our Applications Engineers to get technical assistance.

Always follow the battery manufacturer's requirements for charging or discharging batteries using a Model 2450. Failure to properly charge or discharge batteries may cause them to leak or explode, resulting in personal injury and property damage. Overvoltage and current protection should be provided in the charge circuit, external to the instrument, when charging batteries without built-in protection.

Do not charge or discharge batteries that exceed 21 V at 1.05 A or 210 V at 105 mA.

CAUTION

If you are using the current source to charge or discharge batteries, the following precautions must be observed. Failure to observe these precautions could result in instrument damage that is not covered by the warranty.

Make sure the external voltage never exceeds the voltage limit setting of the current source. This will cause excessive current to be drawn from the external battery or source.

Be sure to set the output-off state of the current source for high impedance. This setting opens the output relay when the output is turned off. With the normal output-off state selected, turning the output off sets the voltage limit to zero. This 0 V source limit condition will cause excessive current to be drawn from the external battery or source.

CAUTION

Carefully consider and configure the output-off state, source, and limits before connecting the Model 2450 to a device that can deliver energy. Devices that can deliver energy include voltage sources, batteries, capacitors, and solar cells. Configure instrument settings before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

When using the current source as a sink, always set the voltage limit and configure overvoltage protection (OVP) to levels that are higher than the external voltage level. Failure to do so could result in excessive current flow into the Model 2450 (<105 mA) and incorrect measurements.

Timing information

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.

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.

Overtemperature protection

To prevent damaging heat build-up and ensure specified performance, make sure there is adequate ventilation and air flow around the instrument to ensure proper cooling. Do not cover the ventilation holes on the top, sides, or bottom of the instrument.

Even with proper ventilation, the instrument can overheat in the following situations:

- If the ambient temperature is too high.
- If you use the instrument as a power sink for long periods.

If the instrument overheats, the output is turned off and an error message is displayed.

CAUTION

If an overtemperature condition occurs, turn off the instrument and allow it to cool for 30 minutes. You cannot turn the output on until the instrument cools down. Verify that there is adequate ventilation. When you return power to the instrument, verify that the cooling fan is running. If not, contact Keithley Instruments. Leaving the instrument turned on with the failure message displayed or with an inoperative cooling fan may result in damage to the instrument.

Calculating accuracy

Instrument accuracy specifications can be expressed in a variety of ways. To illustrate how to calculate measurement errors from instrument specifications, the following topics provide examples of calculations.

Calculating source accuracy

To calculate source accuracy, use the formula:

Accuracy = ± (% *of reading* + *offset)*

For example, assuming:

- Current output = 100 mA on 100 mA range
- Accuracy specification = ± (0.025 % of output + 20 µA)

You calculate the current source accuracy as shown in the following equations.

Error (ΔI) = ± (% *of reading*) + *offset*

= {(100 mA * 0.00025) + 20 µA}
= ± {25 µA + 20 µA}
= ± 45 µA

Calculating measurement accuracy

To calculate measurement accuracy, use the formula:

Accuracy = ± (% *of reading* + *offset)*

For example, to calculate the voltage measurement accuracy, assume:

- Input signal = (20 Ω * 100 mA) = 2 V
- Accuracy specification of 2 V range = ± (0.012% of output + 300 µV)

You calculate the current source accuracy as shown in the following equations.

Error (ΔV) = ±(% *of reading*) + *offset*

= ± {(2 V * 0.00012) + 300 µV}
= ± {240 µV + 300 µV}
= ± 540 µV

Offset-compensated ohm calculations

The presence of thermal EMFs (V_{EMF}) can adversely affect low-resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

NOTE

Instrument operations, including offset-compensated ohms, are performed on the input signal in a sequential manner.

For a normal resistance measurement, the Model 2450 sources a current (I) and measures the voltage (V). The resistance (R) is then calculated as (R=V/I) and the reading is displayed.

For offset-compensated ohms, two measurements are performed: one normal resistance measurement, and one using the lowest current source setting.

The offset-compensated ohms reading is then calculated as follows:

$$\text{Offset-compensated } \Omega = \frac{\Delta V}{\Delta I}$$

where:

$$\Delta V = V2 - V1$$

$$\Delta I = I2 - I1$$

V1 is the voltage measurement with the current source at its normal level.

V2 is the voltage measurement using the lowest current source setting.

I1 is the current measurement with the source set to a specific level.

I2 is the current measurement with the source set to zero.

This 2-point measurement process and reading calculation eliminates the resistance contributed by the presence of V_{EMF}.

When the source is turned on, the output cycles between the programmed value and zero (0 A or 0 V) to derive the offset-compensated ohms measurement.

Power calculations

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

$P = V \times I$

Where:

P is the calculated power

V is the measured voltage

I is the measured current

High capacitance operation

The Model 2450 high capacitance feature can prevent problems when you are measuring low current and driving a capacitive load. In this situation, you may see overshoot, ringing, and instability. This occurs because the pole formed by the load capacitance and the current range resistor can cause a phase shift in the voltage-control loop of the instrument.

The actual operating conditions for a given capacitive load can vary. This is due to the large dynamic range of the current measurement capability and wide range of internal resistors in the instrument.

Some test applications require capacitors larger than 100 nF. In these applications, you can use the high-capacitance feature to minimize overshoot, ringing, and instability.

Enabling the high capacitance feature

Before enabling high-capacitance mode, note the following:

- Test the 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.
- When high-capacitance mode is enabled, a minimum load capacitance of 100 nF is recommended. In the absence of this minimum load capacitance, overshoot and ringing may occur.

Using the front panel:

1. Press the **MENU** key.
2. Under Source, select **Settings**.
3. Next to High Capacitance, select **On**.
4. Select **HOME** to return to the operating display.

Using SCPI commands:

Send the command:

```
:SOURce:CURRent:HIGH:CAPacitance ON
```

To turn on high capacitance for a voltage source, replace `CURRent` with `VOLTage`.

Using TSP commands:

Set the source function, then send the command:

```
smu.source.highc = smu.ON
```

Charging the capacitor and making readings

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

1. Set the current limit to a value that is higher than will be used for the measurement (for example, if measuring at 10 µA, the initial current limit can be set for 1 A).
2. After the capacitor charges, lower the current limit and measure range to obtain the current measurement.

Filtering measurement data

Filters allow you to produce one averaged sample from a number of measurements. In situations where you have noise levels that fluctuate above and below the measured signal, this can help you produce more accurate measurements.

The Model 2450 has two filter options, repeating average and moving average.

The repeating average filter produces slower results, but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

If you create test algorithms and you are using the averaging filters, make sure the algorithms clear the filter memory stacks at appropriate times to avoid averaging an inappropriate set of measurements.

Repeating average filter

When the repeating average filter is selected, a set of measurements are made, which are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack has to be completely filled before an averaged sample can be produced.

Moving average filter

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

Note that when the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

Setting up the averaging filter

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Math**.
3. For the Filter State, select **ON** or **OFF**.
4. For the Filter Type, select **Moving** or **Repeat**.
5. For the Filter Count, enter the number of measurements to be made for each averaged measurement sample.
6. Select **HOME** to return to the operating display.

Quick Tip

Once the filter type and count is set up, you can enable and disable the averaging filter from the Measure Settings screen. When filtering is enabled, the FILT indicator on the Home screen is lit.

Using SCPI commands:

To set number of measurements to be averaged for current measurements, send the command:

```
:SENSe:CURRent:AVERage:COUNt <n>
```

where `<n>` is the number of measurements to be averaged from 1 to 100.

To set number of measurements to be averaged, send the command:

```
:SENSe:CURRent:AVERage:TCONtrol <type>
```

where `<type>` is the filter type, `REP`eat or `MOV`ing.

To enable the selected averaging filter, send the command:

```
:SENSe:CURRent:AVERage:STATe ON
```

To set the above commands for resistance measurements, replace `CURRent` with `RESistance`. To set the above commands for voltage measurements, replace `CURRent` with `VOLTage`.

Using TSP commands:

Before sending the filter commands, set the measurement function. The filter settings apply to the selected measurement function.

To set number of measurements to be averaged for current measurements, send the command:

```
filterCount = smu.measure.filter.count
```

where *`filterCount`* is the number of measurements to be averaged, from 1 to 100.

To set number of measurements to be averaged, send the command:

```
filterType = smu.measure.filter.type
```

where `<type>` is the filter type, `smu.FILTER_MOVING_AVG` or `smu.FILTER_REPEAT_AVG`.

To enable the selected averaging filter, send the command:

```
smu.measure.filter.enable = smu.FILTER_ON
```

Reset default values

When you turn instrument power on and off or send a reset command, many of the settings in the instrument are reset to their default values.

The settings that are affected are listed in the following tables. The tables show SCPI, TSP, and front panel settings for each setting. They are sorted alphabetically by the name of the SCPI command.

Default values

Math and limit reset values

Setting	Default value on reset
MENU > Math > Math Function :CALCulate[1]:<function>:MATH:FORMat (on page 6-7) smu.measure.math.format (on page 8-112)	Percent
MENU > Math > b Value :CALCulate[1]:<function>:MATH:MBFactor (on page 6-8) smu.measure.math.mxb.bfactor (on page 8-113)	0
MENU > Math > m Value :CALCulate[1]:<function>:MATH:MMFactor (on page 6-10) smu.measure.math.mxb.mfactor (on page 8-114)	1
MENU > Math > Percent :CALCulate[1]:<function>:MATH:PERCent (on page 6-11) smu.measure.math.percent (on page 8-115)	1.0
MENU > Math > Math State :CALCulate[1]:<function>:MATH:STATe (on page 6-12) smu.measure.math.enable (on page 8-111)	Off
MENU > Measure > Settings > Limits > View > Auto Clear :CALCulate2:<function>:LIMit<Y>:CLEar:AUTO (on page 6-13) smu.measure.limit[Y].autoclear (on page 8-104)	On
MENU > Measure > Settings > Limits > Low Value :CALCulate2:<function>:LIMit<y>:LOWer[:DATA] (on page 6-16) smu.measure.limit[Y].low.value (on page 8-110)	1
MENU > Measure > Settings > Limits > State :CALCulate2:<function>:LIMit<y>:STATe (on page 6-17) smu.measure.limit[Y].enable (on page 8-106)	Off
MENU > Measure > Settings > Limits > High Value :CALCulate2:<function>:LIMit<y>:UPPer[:DATA] (on page 6-18) smu.measure.limit[Y].high.value (on page 8-109)	1

Digital I/O reset values

Setting	Default value on reset
Not available from front panel :DIGital:LINE<n>:MODE (on page 6-19) digio.line[N].mode (on page 8-43)	Digital line, input

Display reset values

Setting	Default value on reset
SETTINGS swipe screen > Display Digits :DISPlay:<function>:DIGits (on page 6-23) smu.measure.displaydigits (on page 8-100)	5½

Format reset values

Setting	Default value on reset
Not available from front panel :FORMat:BORDer (on page 6-29) format.byteorder (on page 8-63)	 Swapped Little endian
Not available from front panel :FORMat[:DATA] (on page 6-30) format.data (on page 8-64)	ASCII
Not available from front panel :FORMat:ASCii:PRECision (on page 6-28) format.asciiprecision (on page 8-62)	Automatic

Localnode reset values

Setting	Default value on reset
Not available from front panel Not applicable for SCPI localnode.prompts (on page 8-71)	Disabled
Not available from front panel Not applicable for SCPI localnode.showevents (on page 8-73)	0

Output reset values

Setting	Default value on reset
MENU > Source > Settings > Output Off Mode :OUTPut[1]:<function>:SMODe (on page 6-31) smu.source.offmode (on page 8-139)	Normal
OUTPUT ON/OFF switch :OUTPut[1][:STATe] (on page 6-34) smu.source.output (on page 8-140)	Off

Terminal reset values

Setting	Default value on reset
TERMINALS button :ROUTe:TERMinals (on page 6-35) smu.measure.terminals (on page 8-124)	Front

Measurement reset values

Setting	Default value on reset
Not available from front panel [:SENSe[1]]:COUNt (on page 6-42) smu.measure.count (on page 8-97)	1
MENU > Measure > Filter/Math > Filter Count [:SENSe[1]]:<function>:AVERage:COUNt (on page 6-43) smu.measure.filter.count (on page 8-100)	10
MENU > Measure > Filter/Math > Filter State [:SENSe[1]]:<function>:AVERage[:STATe] (on page 6-44) smu.measure.filter.enable (on page 8-101)	Off
MENU > Measure > Filter/Math > Filter Type [:SENSe[1]]:<function>:AVERage:TCONtrol (on page 6-45) smu.measure.filter.type (on page 8-102)	Repeat
SETTINGS swipe > Auto Zero [:SENSe[1]]:<function>:AZERo[:STATe] (on page 6-46) smu.measure.autozero.enable (on page 8-89)	On
Not available from front panel [:SENSe[1]]:<function>:DELay:USER<n> (on page 6-47) smu.measure.userdelay[N] (on page 8-126)	0
MENU > Measure > Settings (when instrument is set to source current and measure resistance) [:SENSe[1]]:<function>:OCOMpensated (on page 6-49) smu.measure.offsetcompensation (on page 8-117)	Off
SETTINGS swipe > NPLCs [:SENSe[1]]:<function>:NPLCycles (on page 6-48) smu.measure.nplc (on page 8-116)	1
FUNCTION key [:SENSe[1]]:FUNCtion[:ON] (on page 6-49) smu.measure.func (on page 8-103)	Current
HOME > Range [:SENSe[1]]:<function>:RANGe:AUTO (on page 6-50) smu.measure.autorange (on page 8-86)	On
MENU > Measure > Settings > Auto Range Low Limit [:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 6-51) smu.measure.autorangelow (on page 8-88)	Current: 1e-8 amps (10 nA) Voltage: 20 V Resistance: 20 ohms
Not available from front panel [:SENSe[1]]:<function>:RANGe:AUTO:ULIMit (on page 6-52) smu.measure.autorangehigh (on page 8-87)	Voltage: 0.20 Resistance: 2e5
HOME > Measure Range [:SENSe[1]]:<function>:RANGe[:UPPer] (on page 6-53) smu.measure.range (on page 8-117)	AUTO

Setting	Default value on reset
Not available from front panel [:SENSe[1]]:<function>:RELative (on page 6-55) smu.measure.rel.level (on page 8-122)	0
MENU > Measure > Filter/Math > Rel State [:SENSe[1]]:<function>:RELative:STATe (on page 6-57) smu.measure.rel.enable (on page 8-121)	Off
MENU > Measure > Settings > Sense Mode [:SENSe[1]]:<function>:RSENse (on page 6-58) smu.measure.sense (on page 8-123)	2-wire
Not available from front panel [:SENSe[1]]:<function>:UNIT (on page 6-59) smu.measure.unit (on page 8-125)	Current: Amp Voltage: Voltage Resistance: Ohms

Source reset values

Setting	Default value on reset
Not available from front panel :SOURce[1]:<function>:DELay:AUTO (on page 6-67) smu.source.autodelay (on page 8-128)	On
MENU > Source > Settings > Hi Capacitance :SOURce[1]:<function>:HIGH:CAPacitance (on page 6-69) smu.source.highc (on page 8-135)	Off
HOME > Source :SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] (on page 6-70) smu.source.level (on page 8-135)	0
HOME > Limit :SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 6-71) smu.source.xlimit.level (on page 8-137)	Current: 105 µA Voltage: 21 V
FUNCTION button :SOURce[1]:FUNCtion[:MODE] (on page 6-72) Not applicable for TSP	Current
MENU > Source > Settings > Overvoltage Protection :SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73) smu.source.protect.level (on page 8-141)	None
MENU > Source > Settings > Source Range :SOURce[1]:<function>:RANGe (on page 6-74) smu.source.range (on page 8-143)	Current: 1e-08 Voltage: 20 mV
HOME > Source Range :SOURce[1]:<function>:RANGe:AUTO (on page 6-75) smu.source.autorange (on page 8-127)	On
MENU > Source > Settings > Source Readback :SOURce[1]:<function>:READ:BACK (on page 6-77) smu.source.readback (on page 8-144)	On
FUNCTION key :SOURce[1]:FUNCtion[:MODE] (on page 6-72) smu.source.func (on page 8-134)	Voltage

Buffer reset values

Setting	**Default value on reset**
Not available from front panel :TRACe:FILL:MODE (on page 6-112) bufferVar.fillmode (on page 8-19)	Default buffers: Continuous User-defined buffers: Once
Not available from front panel :TRACe:LOG:STATe (on page 6-113) bufferVar.logstate (on page 8-22)	Default buffers: ON User-created buffers: OFF

Trigger reset values

Setting	**Default value on reset**
Not available from front panel :TRIGger:BLENder<n>:MODE (on page 6-125) trigger.blender[N].orenable (on page 8-166)	AND
Not available from front panel :TRIGger:DIGital<n>:IN:EDGE (on page 6-150) trigger.digin[N].edge	Falling edge
Not available from front panel :TRIGger:DIGital<n>:OUT:LOGic (on page 6-151) trigger.digout[N].logic (on page 8-174)	Positive
Not available from front panel :TRIGger:DIGital<n>:OUT:PULSewidth (on page 6-152) trigger.digout[N].pulsewidth (on page 8-174)	10e-6
Not available from front panel :TRIGger:DIGital<n>:OUT:STIMulus (on page 6-153) trigger.digout[N].stimulus (on page 8-175)	None
Not available from front panel :TRIGger:LAN<n>:IN:EDGE (on page 6-155) trigger.lanin[N].edge (on page 8-177)	Either edge
Not available from front panel :TRIGger:LAN<n>:OUT:IP:ADDRess (on page 6-158) trigger.lanout[N].ipaddress (on page 8-182)	0.0.0.0
Not available from front panel :TRIGger:LAN<n>:OUT:PROTocol (on page 6-159) trigger.lanout[N].protocol (on page 8-183)	TCP
Not available from front panel :TRIGger:LAN<n>:OUT:STIMulus (on page 6-159) trigger.lanout[N].stimulus (on page 8-184)	None
Not available from front panel :TRIGger:TIMer<n>:COUNt (on page 6-167) trigger.timer[N].count (on page 8-216)	1
Not available from front panel :TRIGger:TIMer<n>:DELay (on page 6-168) trigger.timer[N].delay (on page 8-217)	10e-6
Not available from front panel :TRIGger:TIMer<n>:STARt:FRACtional (on page 6-169) trigger.timer[N].start.fractionalseconds (on page 8-221)	0

Setting	Default value on reset
Not available from front panel :TRIGger:TIMer<n>:STARt:GENerate (on page 6-170) trigger.timer[N].start.generate (on page 8-221)	Off
Not available from front panel :TRIGger:TIMer<n>:STARt:SEConds (on page 6-171) trigger.timer[N].start.seconds (on page 8-222)	0
Not available from front panel :TRIGger:TIMer<n>:STARt:STIMulus (on page 6-171) trigger.timer[N].start.stimulus (on page 8-223)	No event
Not available from front panel :TRIGger:TIMer<n>:STATe (on page 6-173) trigger.timer[N].enable (on page 8-219)	Off
Not available from front panel Not applicable for SCPI trigger.tsplinkin[N].edge (on page 8-225)	Falling
Not available from front panel Not applicable for SCPI trigger.tsplinkout[N].logic (on page 8-228)	Positive
Not available from front panel Not applicable for SCPI trigger.tsplinkout[N].pulsewidth (on page 8-228)	10e-6

TSP-Link and TSP-Net reset values

Setting	Default value on reset
Not available from front panel Not applicable for SCPI tsplink.line[N].mode (on page 8-232)	Digital open drain
Not available from front panel Not applicable for SCPI tspnet.timeout (on page 8-244)	20
Not available from front panel Not applicable for SCPI tspnet.tsp.abortonconnect (on page 8-245)	1

Section 5

Introduction to SCPI commands

In this section:

Introduction to SCPI

The Standard Commands for Programmable Instruments (SCPI) standard is a syntax and set of commands that is used to control test and measurement devices.

The following information describes some basic SCPI command information and how SCPI is used with the Model 2450 and presented in the Model 2450 documentation.

Command execution rules

Command execution rules are as follows:

- Commands execute in the order that they are presented in the command message.
- An invalid command generates an error and is not executed.
- Valid commands that precede an invalid command in a command message are executed.
- Valid commands that follow an invalid command in a command message are ignored.

Command messages

A command message is made up of one or more command words sent by the controller to the instrument.

SCPI commands contain several command words that are structured to create command messages. The command words are separated by colons (:). For example, to configure an ethernet connection, the command words are:

```
:SYSTem:COMMunication:LAN:CONFigure
```

Many commands have query options. If there is a query option, it is created by adding a question mark (?) to the command. For example, to query the present ethernet settings, send:

```
:SYSTem:COMMunication:LAN:CONFigure?
```

Commands often take parameters. Parameters follow the command words and a space. For example, to set the instrument to automatically detect the ethernet settings, send:

```
:SYSTem:COMMunication:LAN:CONFigure AUTO
```

SCPI can also use common commands, which consist of an asterisk (*) followed by three letters. For example, you can reset the instrument by sending the following command:

```
*RST
```

The examples above show commands that are sent individually. You can also group command messages when you send them to the instrument. To group a set of commands, separate them with semicolons. For example, to reset the instrument, enable relative offset, and set a relative offset of 0.5 for the current function in the same message, send the command:

```
*RST; SENSe:CURRent:REL:STAT ON; :SENSe:CURRent:RELative .5
```

The colon (:) at the beginning of a command is optional. For example, the following commands are equivalent:

```
:SENSe:CURRent:REL:STAT ON
SENSe:CURRent:REL:STAT ON
```

SCPI command programming notes

This section contains general information about using Standard Commands for Programmable Instruments (SCPI).

SCPI command formatting

This section describes the formatting that this manual uses when discussing SCPI commands.

SCPI command short and long forms

This documentation shows SCPI commands with both uppercase and lowercase letters. The uppercase letters are the required elements of a command. The lowercase letters are optional. However, if you choose to include the letters that are shown in lowercase letters, you must include all of them.

When you send a command to the instrument, case is not important — you can mix uppercase and lowercase letters in program messages.

For example, you can send the command `SENSe:COUNt` in any of the following formats:

```
SENSe:COUNt
sense:count
SENS:COUN
Sens:Coun
```

Optional command words

If a command word is enclosed in brackets ([ ]), the command word is optional. Do not include the brackets if you send the optional command word to the instrument.

For example, you can send the command `:SYSTem:BEEPer[:IMMediate] <n1>`, <`n2`> in any of the following formats:

```
:SYSTem:BEEPer:IMMediate 500, 1
:SYSTem:BEEPer 500, 1
:SYST:BEEP:IMMediate 500, 1
:SYST:BEEP 500, 1
```

MINimum, MAXimum, and DEFault

You can use `MINimum`, `MAXimum`, or `DEFault` instead of a parameter for some commands.

For example, you can set <`defaultParameter`> for the command `[:SENSe[1]]:RESistance:NPLCycles <defaultParameter>` to the minimum, maximum, or default value. To set NPLC to the minimum value, you can send either of these commands:

```
:SENSe1:RESistance:NPLCycles MINimum
:SENS:RES:NPLC MIN
```

Queries

Some commands are queries and others have a query option. These commands have a question mark (`?`) after the command. You can use the query to determine the present value of the parameters of the command or to get information from the instrument.

For example, to determine what the present setting for NPLC is, you can send:

```
:SENSe1:RESistance:NPLCycles?
```

This query returns the present setting.

If the command has `MINimum`, `MAXimum`, and `DEFault` options, you can use the query command to determine what the minimum, maximum, and default values are. In these queries, the ? is placed before the MINimum, MAXimum, or DEFault parameter. For example, to determine the default value for NPLC, you can send:

```
:SENSe1:RESistance:NPLCycles? DEFault
```

If you send two query commands without reading the response from the first, and then attempt to read the second response, you may receive some data from the first response followed by the complete second response. To avoid this, do not send a query command without reading the response. When you cannot avoid this situation, send a device clear before sending the second query command.

When you query a Boolean option, the instrument returns a 0 or 1, even if you sent OFF or ON when you originally sent the command.

SCPI parameters

The parameters of the SCPI commands are shown in angle brackets (< >). For example:

```
:SYSTem:BEEPer[:IMMediate] <frequency>, <time>
```

The type of information that you can use to replace <`frequency`> and `<time>` is defined in the Usage section of the command description. For this example, the Usage is:

`<frequency>`	The frequency of the beep (20 to 8000)
`<time>`	The amount of time to play the tone in seconds (0.001 to 100)

For this example, you can generate an audible sound by sending:

```
:SYSTem:BEEPer 500, 1
```

Note that you do not include the angle brackets when sending the command.

Sending strings

If you are sending a string, it must begin and end with matching quotes (either single quotes or double quotes). If you want to include a quote character as part of the string, type it twice with no characters in between.

A command string sent to the instrument must terminate with a <new line> character. The IEEE-488.2 EOI (end-or-identify) message is interpreted as a <new line> character and can be used to terminate a command string in place of a <new line> character. A <carriage return> followed by a <new line> is also accepted. Command string termination will always reset the current SCPI command path to the root level.

Using the SCPI command reference

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

Figure 122: SCPI command description example

:SYSTem:BEEPer[:IMMediate]

This command generates an audible tone.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:BEEPer[:IMMediate] <frequency>, <time>
```

<frequency>	The frequency of the beep (0.001 to 8000)
<time>	The amount of time to play the tone in seconds (0.1 to 100)

Details

You can use the beeper of the Model 2450 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 only sounds if it is enabled.

Example

`:SYSTem:BEEPer 500, 1`	Beep at 500 Hz for one second

Also see

:SYSTem:BEEPer:STATe (on page 6-100)

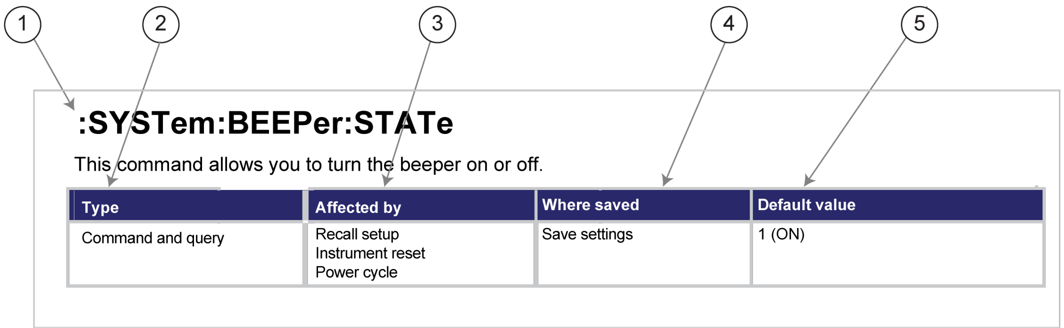
Each command listing is divided into five major subsections that contain information about the command:

- Command name and summary table
- Usage
- Details
- Example
- Also see

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

Command name and summary table

Each instrument command description starts with the command name, followed by a table with relevant information for each command. Definitions for the numbered items below are listed following the figure.



1 **Instrument command name**. Signals the beginning of the command description and is followed by a brief description of what the command does.

2 **Type of command**. Options are:

- **Command only**. There is a command but no query option for this command.
- **Command and query**. The command has both a command and query form.
- **Query only**. This command is a query.

3 **Affected by**. Commands or actions that have a direct effect on the instrument command.

- **Recall settings.** If you send `*RCL` to recall the system settings, this setting is changed to the saved value.
- **Instrument reset.** When you reset the instrument, this command is reset to its default values. Reset can be done from the front panel or when you send `*RST`.
- **Power cycle.** The settings for this command are not saved through a power cycle.
- **Source configuration list**. If you recall a source configuration list, this setting changes to the stored setting.
- **Measure configuration list.** If you recall a measure configuration list, this setting changes to the stored setting.

4 **Where saved**. Indicates where the command settings reside once they are used on an instrument. Options include:

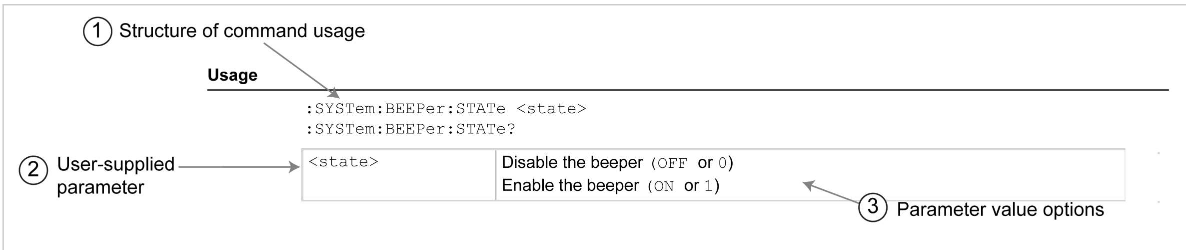
- **Not saved.** Command is not saved and must be sent each time you use it.
- **Nonvolatile memory.** The command is stored in a storage area in the instrument where information is saved even when the instrument is turned off.
- **Save settings.** This command is saved when you send the `*SAV` command.
- **Source configuration list.** This command is stored in source configuration lists.
- **Measure configuration list.** This command is stored in measure configuration lists.

5 **Default value:** Lists the default value for the command. The parameter values are defined in the Usage or Details sections of the command description.

Command usage

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

Figure 123: SCPI command description usage identification



1 **Structure of command usage:** Shows how the parts of the command should be organized.

2 **User-supplied parameters:** Indicated by angle brackets (< >).

NOTE

Some commands have optional parameters. Optional parameters are presented on separate lines in the Usage section, presented in the required order with each valid permutation of optional parameters. For example:

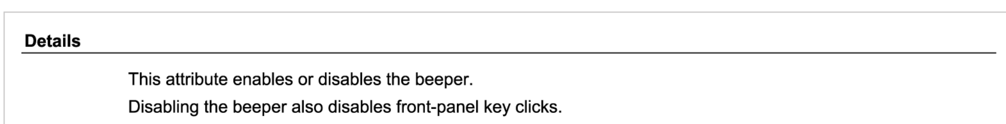
```
:SYSTem:COMMunication:LAN:CONFigure AUTO
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress, NETmask
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress, NETmask, GATeway
:SYSTem:COMMunication:LAN:CONFigure?
```

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

Command details

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

Figure 124: Details section of command listing



Example section

The Example section of the command description shows some simple examples of how the command can be used.

Figure 125: SCPI command description code examples

1. Example code that you can copy from this table and paste into your own application. Examples are generally shown using the short forms of the commands.
2. Description of the code and what it does. This may also contain the output of the code.

Related commands list

The Also see section of the remote command description lists commands that are related to the command being described.

Also see
:SYSTem:BEEPer[:IMMediate] (on page 6-99)

Section 6

SCPI command reference

In this section:

*RCL

This command returns the instrument to the setup that was saved with the `*SAV` command.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*RCL <n>
```

`<n>`	An integer between 0 and 4 that represents the saved setup

Details

Restores the state of the instrument from a copy of user-saved settings that are stored in the setup memory. The settings are saved using the `*SAV` command.

If you view the user-saved settings from the front panel of the instrument, these are stored as scripts named `Setup0<n>`.

Example

`*RCL 3`	Restores the settings stored in memory location 3.

Also see

Saving setups (on page 2-112)
*SAV (on page 6-2)

*SAV

This command saves the present instrument settings as a user-saved setup.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Nonvolatile memory	Not applicable

Usage

`*SAV <n>`

`<n>`	An integer from 0 to 4

Details

Save the present instrument settings as a user-saved setup. You can restore the settings with the `*RCL` command.

Any command that is affected by `*RST` can be saved with the `*SAV` command.

Any settings that had been stored previously as `<n>` are overwritten.

If you view the user-saved setups from the front panel of the instrument, they are stored as scripts named `Setup0<n>`.

Example

`*SAV 2`	Saves the instrument settings in memory location 2.

Also see

*RCL (on page 6-1)

:ABORt

This command stops all trigger model commands on the instrument.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

`:ABORt`

Details

When this command is received, the Model 2450 stops the trigger model.

Also see

Aborting the trigger model (on page 3-108)
Trigger model (on page 3-95)

:FETCh?

This query command requests the latest reading that was stored in a reading buffer.

Type	Affected by	Where saved	Default value
Query	Not applicable	Not applicable	Not applicable

Query

```
:FETCh?
:FETCh? <bufferName>
:FETCh? <bufferName>, <bufferElements>
```

`<bufferName>`	The name of the buffer where the reading is stored; if nothing is specified, the reading that is stored in `defbuffer1`
`<bufferElements>`	See **Details**; if nothing is defined, the measurement is returned

Details

This command requests the last available reading from a reading buffer. If you send this command more than once and there are no new readings, it will return the values that were returned for the previous query.

If you send `:FETCh?` while a trigger model is running, no data is returned until the trigger model is in idle.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
`DATE`	The date when the data point was measured
`FORMatted`	The measured value as it appears on the front panel
`FRACtional`	The fractional seconds for the data point when the data point was measured
`READing`	The measurement reading based on the SENS:FUNC setting; if no buffer elements are defined, this option is used
`RELative`	The relative time when the data point was measured
`SEConds`	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
`SOURce`	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 6-77))
`SOURFORMatted`	The source value as it appears on the display
`SOURSTATus`	The status information associated with sourcing
`SOURUNIT`	The unit of value associated with the source value
`STATus`	The status information associated with the measurement
`TIME`	The time for the data point
`TSTamp`	The timestamp for the data point
`UNIT`	The unit of measure associated with the measurement

NOTE

If you have `FORMat[:DATA]` set to `REAL` or `SREAL`, you will have fewer options for buffer elements. If you request one of the buffer elements, you will see the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

Example

`FETCh? "defbuffer1", DATE, READ, SOUR`	Retrieve the date, measurement value, and source setting for the most recent data captured in `defbuffer1`. Typical output is: `03/21/2013,-1.375422E-11,0.000000E+00`

Also see

:MEASure:<function>? (on page 6-4)
:READ? (on page 6-6)
:TRACe:DATA? (on page 6-109)

:MEASure:<function>?

This command makes a measurement and returns the reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:MEASure:<function>?
:MEASure:<function>? <bufferName>
:MEASure:<function>? <bufferName>, <bufferElements>
```

`<function>`	Current: `CURRent[:DC]` Resistance: `RESistance` Voltage: `VOLTage[:DC]`
`<bufferName>`	The name of the buffer where the reading is stored; if nothing is specified, the reading that is stored in `defbuffer1`
`<bufferElements>`	See **Details**

Details

This command makes a measurement using the specified function and stores the reading in a reading buffer. When a reading buffer is used with a command or action that involves taking multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

If a different function is selected, sending this command will change the measurement function to the one specified by `<function>`. This function remains selected after the measurement is complete.

This command performs the same function as sending `:SENse:FUNCtion`, then `READ?`.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for <bufferElements> are described in the following table.

Option	Description
DATE	The date when the data point was measured
FORMatted	The measured value as it appears on the front panel
FRACtional	The fractional seconds for the data point when the data point was measured
READing	The measurement reading based on the SENS:FUNC setting; if no buffer elements are defined, this option is used
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
SOURce	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 6-77))
SOURFORMatted	The source value as it appears on the display
SOURSTATus	The status information associated with sourcing
SOURUNIT	The unit of value associated with the source value
STATus	The status information associated with the measurement
TIME	The time for the data point
TSTamp	The timestamp for the data point
UNIT	The unit of measure associated with the measurement

NOTE

If you have FORMat[:DATA] set to REAL or SREAL, you will have fewer options for buffer elements. If you request one of the buffer elements, you will see the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

Example

```
TRACe:MAKE "voltMeasBuffer", 10000
MEAS:VOLT? "voltMeasBuffer", FORM, DATE, READ
```

Create a buffer named voltMeasBuffer. Make a voltage measurement and store it in the buffer voltMeasBuffer and return the formatted reading, the date, and the reading elements from the buffer.
Sample output is:

```
-00.0024 mV,05/16/2014,-2.384862E-06
```

Also see

:READ? (on page 6-6)
[:SENSe[1]]:FUNCtion[:ON] (on page 6-49)

:READ?

This query makes measurements, places them in a reading buffer, and returns the latest reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:READ?
:READ? <bufferName>
:READ? <bufferName>, <bufferElements>
```

`<bufferName>`	The name of the buffer where the reading is stored; if nothing is specified, the reading that is stored in `defbuffer1`
`<bufferElements>`	See **Details**

Details

When a reading buffer is used with a command or action that involves taking multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
`DATE`	The date when the data point was measured
`FORMatted`	The measured value as it appears on the front panel
`FRACtional`	The fractional seconds for the data point when the data point was measured
`READing`	The measurement reading based on the SENS:FUNC setting; if no buffer elements are defined, this option is used
`RELative`	The relative time when the data point was measured
`SEConds`	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
`SOURce`	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 6-77))
`SOURFORMatted`	The source value as it appears on the display
`SOURSTATus`	The status information associated with sourcing
`SOURUNIT`	The unit of value associated with the source value
`STATus`	The status information associated with the measurement
`TIME`	The time for the data point
`TSTamp`	The timestamp for the data point
`UNIT`	The unit of measure associated with the measurement

NOTE

If you have `FORMat[:DATA]` set to `REAL` or `SREAL`, you will have fewer options for buffer elements. If you request one of the buffer elements, you will see the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

Example

```
:TRACe:MAKE "voltMeasBuffer", 10000
:SENSe:FUNCtion "VOLTage"
:READ? "voltMeasBuffer", FORM, DATE, READ
```

Create a buffer named `voltMeasBuffer`. Make a measurement, store it in the buffer `voltMeasBuffer`, and return the formatted readings, data, and reading buffer elements for the last reading stored in `voltMeasBuffer`.

Sample output is:

```
-00.0020 mV,05/16/2014,-2.031637E-06
```

Also see

:FETCh? (on page 6-3)
:INITiate[:IMMediate] (on page 6-124)

CALCulate subsystem

The commands in this subsystem configure and control the math and limit operations.

:CALCulate[1]:<function>:MATH:FORMat

This command specifies which math operation is performed on measurements.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PERC

Usage

```
:CALCulate[1]:<function>:MATH:FORMat <name>
:CALCulate[1]:<function>:MATH:FORMat?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<name>`	The name of the math operation: • y = mx+b: `MXB` • Percent: `PERCent` • Reciprocal: `RECiprocal`

Details

This specifies which math operation is performed on measurements for the selected measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Specify a constant that is applied to the measurement and display measurements as percentages.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is 1/X, where `X` is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

> **NOTE**
>
> If you send this command without the `<function>` parameter, it will set the state of the math format for all functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STATE ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 3-6)
:CALCulate[1]:<function>:MATH:MBFactor (on page 6-8)
:CALCulate[1]:<function>:MATH:MMFactor (on page 6-10)
:CALCulate[1]:<function>:MATH:PERCent (on page 6-11)
:CALCulate[1]:<function>:MATH:STATe (on page 6-12)

:CALCulate[1]:<function>:MATH:MBFactor

This command specifies the offset for the y = mx + b operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
:CALCulate[1]:<function>:MATH:MBFactor <n>
:CALCulate[1]:<function>:MATH:MBFactor?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The offset for the y = mx + b operation; the valid range is −1e12 to +1e12

Details

This attribute specifies the offset (b) for an mx + b operation.

The mx+b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$y = mx + b$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

NOTE

If you send this command without the `<function>` parameter, it will set the scale factor for all functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STATE ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 3-6)
:CALCulate[1]:<function>:MATH:FORMat (on page 6-7)
:CALCulate[1]:<function>:MATH:MMFactor (on page 6-10)
:CALCulate[1]:<function>:MATH:STATe (on page 6-12)

:CALCulate[1]:<function>:MATH:MMFactor

This command specifies the scale factor for the y = mx + b math operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1.000000

Usage

```
:CALCulate[1]:<function>:MATH:MMFactor <n>
:CALCulate[1]:<function>:MATH:MMFactor?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The scale factor; the valid range is –4294967295 to +4294967295

Details

This command sets the scale factor (m) for an mx + b operation for the selected measurement function.

The mx+b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

y = mx + b

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

NOTE

If you send this command without the `<function>` parameter, it will set the scale factor for all functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STATE ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

:CALCulate[1]:<function>:MATH:FORMat (on page 6-7)
:CALCulate[1]:<function>:MATH:MBFactor (on page 6-8)
:CALCulate[1]:<function>:MATH:STATe (on page 6-12)

:CALCulate[1]:<function>:MATH:PERCent

This command specifies the constant that is used when math operations are set to percent.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1.000000

Usage

```
:CALCulate[1]:<function>:MATH:PERCent <n>
:CALCulate[1]:<function>:MATH:PERCent?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The constant when the math operation is set to percent; the range is −1e12 to +1e12

Details

This is the constant that is used when the math operation is set to percent.

The percent math function displays measurements as percent deviation from a specified constant. The percent calculation is:

$$Percent = \left(\frac{input - reference}{reference}\right) \times 100\%$$

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

NOTE

If you send this command without the `<function>` parameter, it will set the constant for all functions.

Example

`CALC:VOLT:MATH:FORM PERC`
`CALC:VOLT:MATH:PERC 50`
`CALC:VOLT:MATH:STAT ON`
	Set the math operations for voltage to percent.
	Set the percentage value to 50.
	Enable math operations.

Also see

Calculations that you can apply to measurements (on page 3-6)
:CALCulate[1]:<function>:MATH:FORMat (on page 6-7)
:CALCulate[1]:<function>:MATH:MBFactor (on page 6-8)
:CALCulate[1]:<function>:MATH:MMFactor (on page 6-10)
:CALCulate[1]:<function>:MATH:STATe (on page 6-12)

:CALCulate[1]:<function>:MATH:STATe

This command enables or disables the math operations.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
:CALCulate[1]:<function>:MATH:STATe <n>
:CALCulate[1]:<function>:MATH:STATe?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	Enable math operations: `ON` or `1` Disable math operations: `OFF` or `0`

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

NOTE

If you send this command as `CALC:STAT`, it will set the state of math operations for all functions.

Example

`:CALC:VOLT:MATH:FORM MXB`
`:CALC:VOLT:MATH:MMF 0.80`
`:CALC:VOLT:MATH:MBF 50`
`:CALC:VOLT:MATH:STATE ON`
	Set the math function for voltage measurements to mx+b.
	Set the scale factor for voltage measurements to 0.80.
	Set the offset factor to 50.
	Enable the math function.

Also see

:CALCulate[1]:<function>:MATH:FORMat (on page 6-7)
Calculations that you can apply to measurements (on page 3-6)

:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO

This command indicates if limit *Y* should be cleared automatically or not.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1 (ON)

Usage

```
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO <state>
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<Y>`	The limit: `1` or `2`
`<state>`	The auto clear setting: • Enable: `ON` or `1` • Disable: `OFF` or `0`

Details

When this command sets autoclear to on for a measurement function, if a measurement fails limit, but the next measurement passes limit, the failed limit condition is cleared. Therefore, if you are making a series of measurements, the instrument uses last measurement limit for the fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the auto clear setting to off. When this set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The auto clear setting affects both the high and low limits of *Y*.

NOTE

If you send this command without the `<function>` parameter, it will set autoclear for all functions.

Example

`:CALC2:CURR:LIMit1:CLEar:AUTO ON`	Turns on autoclear for limit 1 when measuring current.

Also see

:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate] (on page 6-14)

:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate]

This command clears the results of the limit test.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

`:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate]`

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<y>`	The limit that the setting applies to, 1 or 2

Details

Use this command to clear the test results of limit *Y* when the limit auto clear command is disabled. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, enable the auto clear command.

NOTE

If you send this command without the `<function>` parameter, it will set the limit for all functions.

Example

`calc2:curr:lim1:clear`	Clear the results for limit test 1.

Also see

:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 6-16)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 6-18)

:CALCulate2:<function>:LIMit<Y>:FAIL?

This command queries the results of a limit test.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:CALCulate2:<function>:LIMit<Y>:FAIL?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<Y>`	The test that the result applies to, 1 or 2

Details

The results of the limit test for limit *Y*:

- `NONE`: Test passed; the measurement is between the upper and lower limits
- `HIGH`: Test failed; the measurement exceeded the upper limit
- `LOW`: Test failed; the measurement exceeded the lower limit
- `BOTH`: Test failed; the measurement exceeded both limits

These commands query the result of a limit test for the selected measurement function.

The response message indicates if the limit test has passed or how it failed.

Reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command.

Note that if you are making a series of measurements and auto clear enabled for a limit, the last measurement limit dictates the fail indication for the limit. If autoclear is disabled, you can take a series of readings and read fails to see if any of one of the readings failed.

To use this attribute, you must set the limit state to enable.

Example

`CALC2:VOLT:LIM1:LOW 0.25`
`CALC2:VOLT:LIM1:UPP 2.5`
`CALC2:VOLT:LIMIT1:STAT ON`
`CALC2:VOLT:LIMIT1:FAIL?`
	Set lower limit 1 for voltage to 0.25 volts.
	Set upper limit 1 for voltage to 2.5 volts.
	Enable limit 1 testing for voltage.
	Check the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:STATe (on page 6-17)
Limit testing and binning (on page 3-68)

:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]

This command specifies the lower limit for limit tests.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	-1.000000E+00

Usage

```
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <n>
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]?
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]? DEFault
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]? MINimum
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]? MAXimum
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<Y>`	The limit test that the lower limit applies to, 1 or 2
`<n>`	The value of the lower limit (–9.99999e+11 to +9.99999e+11)

Details

This command sets the lower limits for the limit tests for the selected measurement function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

NOTE

If you send this command without the `<function>` parameter, it will set the limit for all functions.

Example

`CALC2:VOLT:LIM1:LOW 0.25`
`CALC2:VOLT:LIM1:UPP 2.5`
`CALC2:VOLT:LIMIT1:STAT ON`
`CALC2:VOLT:LIMIT1:FAIL?`
	Set lower limit 1 for voltage to 0.25 volts.
	Set upper limit 1 for voltage to 2.5 volts.
	Enable limit 1 testing for voltage.
	Check the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] (on page 6-18)

:CALCulate2:<function>:LIMit<Y>:STATe

This command enables or disables a limit test.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
:CALCulate2:<function>:LIMit<Y>:STATe <b>
:CALCulate2:<function>:LIMit<Y>:STATe?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<Y>`	The limit that the setting applies to, 1 or 2
`<b>`	Enable the limit test: `1` or `ON` Disable the limit test: `0` or `OFF`

Details

This command enables or disables a limit test for the selected measurement function.

NOTE

If you send this command without the `<function>` parameter, it will set the state of math operations for all functions.

Example

`CALC2:VOLT:LIM1:LOW 0.25`
`CALC2:VOLT:LIM1:UPP 2.5`
`CALC2:VOLT:LIMIT1:STAT ON`
`CALC2:VOLT:LIMIT1:FAIL?`
	Set lower limit 1 for voltage to 0.25 volts.
	Set upper limit 1 for voltage to 2.5 volts.
	Enable limit 1 testing for voltage.
	Check the test results.

Also see

None

:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]

This command specifies the upper limit for a limit test.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1.000000E+00

Usage

```
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <n>
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]?
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? DEFault
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? MINimum
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? MAXimum
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<Y>`	The limit test that the upper limit applies to, 1 or 2
`<n>`	The value of the upper limit (–9.99999e+11 to +9.99999e+11)

Details

This command sets the high limits for the limit tests for the selected measurement function. When limit testing is enabled for this limit, the instrument generates a fail indication when the measurement value is more than this value.

NOTE

If you send this command without the `<function>` parameter, it will set the limit for all functions.

Example

`CALC2:VOLT:LIM1:LOW 0.25`
`CALC2:VOLT:LIM1:UPP 2.5`
`CALC2:VOLT:LIMIT1:STAT ON`
`CALC2:VOLT:LIMIT1:FAIL?`
	Set lower limit 1 for voltage to 0.25 volts.
	Set upper limit 1 for voltage to 2.5 volts.
	Enable limit 1 testing for voltage.
	Check the test results.

Also see

:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] (on page 6-16)

DIGital subsystem

The commands in the DIGital subsystem control the digital I/O lines.

:DIGital:LINE<n>:MODE

This command sets the digital I/O line to be a digital line or trigger model line and sets the line as an input, output, or open-drain. You can also use this attribute to configure synchronous triggering modes.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	DIG, IN

Usage

```
:DIGital:LINE<n>:MODE <triggerType>, <lineState>
:DIGital:LINE<n>:MODE?
```

`<n>`	The digital I/O line (1 to 6)
`<triggerType>`	Sets the digital line configuration type; the options are: • Allow direct digital control of the line (not used by trigger model): `DIGital` • Configure for trigger control: `TRIGger` • Configure as a synchronous master or acceptor: `SYNChronous`
`<lineState>`	The line state; see **Details** for values

Details

Set `<lineState>` to one of the values shown in the following table.

Value	Description
`IN`	Detects falling-edge, rising-edge, or either-edge triggers as input.
`OUT`	Generates a positive or negative edge for output when requested.
`OPENdrain`	Configures the line to be an open-drain signal (makes the line a digital open-drain line with 100 kΩ pull-up resistor). When used with `<triggerType>` set to `TRIGger`, this values uses the edge setting as `IN`.
`ACCeptor`	Use this value only with the `SYNChronous` trigger type. This value detects a falling-edge as an input trigger and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line. Detects a falling-edge input pulse.
`MASTer`	Use this value only with the `SYNChronous` trigger type. This value detects a rising-edge triggers as an input. Asserts a TTL-low pulse for output. Detects a rising-edge input pulse.

Example

`:DIG:LINE1:MODE DIG, OUT`	Set digital I/O line 1 as a digital output line.

Also see

Digital I/O port (on page 3-62)

:DIGital:LINE<n>:STATe

This command sets a digital I/O line high or low.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	See **Details**

Usage

```
:DIGital:LINE<n>:STATe <state>
:DIGital:LINE<n>:STATe?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<state>`	Bit low: `0` Bit high: `1`

Details

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

Set the state to zero (`0`) to clear the bit; any non-zero value sets the bit.

Example

`:DIG:LINE1:STAT 1`	Sets line 1 (bit B1) of the digital I/O port high.

Also see

Digital I/O port (on page 3-62)
:DIGital:LINE<n>:MODE (on page 6-19)
:DIGital:READ? (on page 6-21)
:DIGital:WRITe <n> (on page 6-21)
:TRIGger:DIGital<n>:IN:EDGE (on page 6-150)

:DIGital:READ?

This command reads the digital I/O port.

Type	Affected by	Where saved	Default value
Query	Not applicable	Not applicable	Not applicable

Usage

```
:DIGital:READ?
```

Details

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

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

Example

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

Also see

Digital I/O bit weighting (on page 3-67)
Digital I/O port (on page 3-62)

:DIGital:WRITe <n>

This command writes to all digital I/O lines.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

```
:DIGital:WRITe <n>
```

`<n>`	The value to write to the port (0 to 63)

Details

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a `data` value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other 3 lines are set low).

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

Example

`:DIG:WRIT 63`	Sets digital I/O lines 1 through 6 high (binary 111111).

Also see

Digital I/O bit weighting (on page 3-67)
Digital I/O port (on page 3-62)

DISPlay subsystem

This subsystem contains commands that control the front-panel display.

:DISPlay:CLEar

This command clears the front-panel User Display swipe screen.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

```
:DISPlay:CLEar
```

Details

This command clears the User Display screen.

If there are active scripts running, there might be a delay before the screen clears. The screen is cleared as soon as processing time becomes available.

Example

`DISP:CLE` `DISP:SCR USER`	Clear the User Display screen and switch to display the User Display screen.
`DISP:USER1:TEXT "Batch A122"` `DISP:USER2:TEXT "Test running"`	Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

:DISPlay:USER<n>:TEXT[:DATA] (on page 6-27)

:DISPlay:<function>:DIGits

This command determines the number of digits that are displayed for measurements on the front panel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	5

Usage

```
:DISPlay:<function>:DIGits <n>
:DISPlay:<function>:DIGits?
:DISPlay:<function>:DIGits? DEFault
:DISPlay:<function>:DIGits? MINimum
:DISPlay:<function>:DIGits? MAXimum
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	3.5 digit resolution: `3` 4.5 digit resolution: `4` 5.5 digit resolution: `5` 6.5 digit resolution: `6`

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

NOTE

If you send this but do not define the function (`:DISPlay:DIGits`), the digits values for all functions are changed.

Example

`:DISP:CURR:DIG 5`	Set the front panel to display current measurements with 5½ digits.

Also see

None

:DISPlay:LIGHt:STATe

This command sets the brightness of the front-panel display.

Type	Affected by	Where saved	Default value
Command and query			ON75

Usage

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

`<brightness>`	The brightness of the display: • 100%: `ON100` • 75%: `ON75` • 50%: `ON50` • 25%: `ON25` • Display off: `OFF` • Display and all indicators off: `BLACkout`

Details

This command determines the brightness of the front-panel display.

NOTE

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

Example

`DISP:LIGH:STAT ON50`	Set the display brightness to 50%

Also see

None

:DISPlay:READing:FORMat

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

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	PREF

Usage

```
:DISPlay:READing:FORMat <format>
:DISPlay:READing:FORMat?
```

`<format>`	Use exponent format: `EXPonent` Use leading zeros: `PREFix`

Details

This setting only affects the front-panel display. It does not affect the readings in buffers.

When the prefix option is selected, the display automatically shows in exponent format when the prefix format does not fit.

Example

`DISP:READ:FORM EXP`	Change front-panel display to show readings in exponent format.

Also see

None

:DISPlay:SCReen

This command changes which front-panel screen is displayed.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

`:DISPlay:SCReen <screenName>`

`<screenName>`	The screen to display: • Home screen: `HOME` • Source swipe screen: `SOURce` • Data Trend swipe screen: `PLOT` • User Display swipe screen: `USER` • Buffer Statistics swipe screen: `STATistics` • Settings swipe screen: `SETTings` • Graph screen: `GRAPh` • Data Sheet screen: `DATasheet`

Example

`DISP:CLE`
`DISP:SCR USER`
`DISP:USER1:TEXT "Batch A122"`
`DISP:USER2:TEXT "Test running"`
	Clear and display the User Display swipe screen.
	Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

None

:DISPlay:USER<n>:TEXT[:DATA]

This command defines the text that is displayed on the front-panel User Display screen.

Type	Affected by	Where saved	Default value
Command only	Power cycle	Not applicable	Not applicable

Usage

```
:DISPlay:USER<n>:TEXT[:DATA] <textMessage>
```

`<n>`	The line of the User Display on which to display text: • Top line: `1` • Bottom line: `2`
`<textMessage>`	String that contains the message; up to 20 characters for `USER1` and 32 characters for `USER2`

Details

`USER1` sets the text for the top line of the User Display. `USER2` sets the text for the bottom line.

These commands define text messages for the User Display swipe screen.

If you enter too many characters, the instrument displays an error message and shortens the message to fit.

Example

`DISP:CLE`
`DISP:SCR USER`
`DISP:USER1:TEXT "Batch A122"`
`DISP:USER2:TEXT "Test running"`
	Clear the User Display screen and switch to display the User Display screen.
	Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

:DISPlay:SCReen (on page 6-26)

FORMat

The commands for this subsystem select the data format that is used to transfer instrument readings over the remote interface.

:FORMat:ASCii:PRECision

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

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:FORMat:ASCii:PRECision <n>
:FORMat:ASCii:PRECision?
:FORMat:ASCii:PRECision? MINimum
:FORMat:ASCii:PRECision? MAXimum
:FORMat:ASCii:PRECision? DEFault
```

`<n>`	The precision: • Automatic: `0` • Specific value: `1` to `16` • `MINimum` (0) • `MAXimum` (16) • `DEFault` (0)

Details

This attribute specifies the precision (number of digits) for queries.

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

Example

`:FORM:ASC:PREC 10`	Set a precision of 10 digits. An example of the output is: `-6.999999881E-01`

Also see

:FORMat[:DATA] (on page 6-30)

:FORMat:BORDer

This command sets the byte order for the IEEE-754 binary formats.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	SWAP

Usage

```
:FORMat:BORDer <name>
:FORMat:BORDer?
```

<name>	The binary byte order: • Normal byte order: NORMal • Reverse byte order for binary formats: SWAPped

Details

This attribute selected the byte order in which data is written.

The SWAPped byte order must be used when transmitting binary data to a computer with a Microsoft Windows operating system.

The ASCII data format can only be sent in the normal byte order. If the ASCII format is selected, the SWAPped selection is ignored.

When you select NORMal byte order, the data format for each element is sent as follows:

```
Byte 1 Byte 2 Byte 3 Byte 4 (Single precision)
```

When you select SWAPped, the data format for each element is sent as follows:

```
Byte 4 Byte 3 Byte 2 Byte 1 (Single precision)
```

The “#0” header is not affected by this command. The header is always sent at the beginning of the data string for each measurement conversion.

Example

FORM:BORD NORM	Use the normal byte order.

Also see

:FORMat[:DATA] (on page 6-30)

:FORMat[:DATA]

This command selects the data format that is used when transferring readings over the remote interface.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	ASC

Usage

```
:FORMat[:DATA] <type>
:FORMat[:DATA] REAL
:FORMat[:DATA]?
```

`<type>`	The data format, which can be one of the following: • ASCII format: `ASCii` • IEEE Std. 754 double-precision format: `REAL` • IEEE Std. 754 single-precision format: `SREal`

Details

This command affects the output of `READ?`, `FETCh?`, `MEASure:<function>?`, and `TRACe:DATA` queries over a remote interface. All other queries are returned in the ASCII format.

NOTE

The Model 2450 only responds to input commands using the ASCII format, regardless of the data format that is selected for output strings.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with “#0” and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

If you set this to `REAL` or `SREAL`, you will have fewer options for buffer elements with the `TRACe:DATA?`, `READ?`, `MEASURE:<function>?`, and `FETCh?` commands. If you request one of the buffer elements, you will see the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

Example

`FORM REAL`	Set the format to double-precision format.

Also see

:TRACe:DATA? (on page 6-109)

OUTPut subsystem

The output subsystem provides information and settings that control the output of the selected source.

:OUTPut[1]:<function>:SMODe

This command defines the state of the source when the output is turned off.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	NORM

Usage

```
:OUTPut[1]:<function>:SMODe <state>
:OUTPut[1]:<function>:SMODe?
```

`<function>`	The function to which this setting applies: • Current: `CURRent[:DC]` • Voltage: `VOLTage[:DC]`
`<state>`	See the **Details** below for specifics regarding each of the following options: • `NORMal` • `HIMPedance` • `ZERO` • `GUARd`

Details

This command sets the state of the output when the source is off for the selected function.

When the Model 2450 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10 percent of the full scale of the present current range
- OUTPUT OFF is displayed in the Home page Source area
- The Source button on the Home page shows the output that will be sourced when the output is turned on again

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

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

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

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

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

- The measurement sense is changed to 2-wire sense
- The source function is set to voltage
- The source voltage is set to 0
- Set the range to the presently selected range (turn off autorange)
- Program the voltage DAC to zero, and the current DAC to full scale of the present current range.

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

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

- The measurement sense is changed to 2-wire sense
- The current source is selected and set to 0 A
- The voltage limit is set to 10% full scale of the present voltage range

NOTE

If you send this command without the `<function>` parameter, it will set the output-off state for all functions.

Example

`:OUTP:CURR:SMOD HIMP`	Sets the output-off state for the current function so that the instrument opens the output relay when the output is turned off.

Also see

Output-off state (on page 2-82)
:OUTPut[1][:STATe] (on page 6-34)

:OUTPut[1]:INTerlock:TRIPped?

This command indicates that the interlock has been tripped.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
OUTPut[1]:INTerlock:TRIPped?
```

Details

This command gives you the status of the interlock. When the safety interlock signal is asserted, all voltage ranges of the instrument are available. However, when the safety interlock signal is not asserted, the 200 V range is disabled, limiting the nominal output to ±37 V.

When the interlock is not asserted:

- The front-panel INTERLOCK indicator is on.
- High voltage ranges are disabled.
- An event message is generated when you attempt to turn on the source with a voltage higher than ±20 V.

If `1` is returned, the interlock signal is asserted and all voltage ranges are available.

If `0` is returned, the interlock is not asserted and the 200 V range is disabled. Lower voltage ranges are available.

Example

`OUTP:INT:TRIP?`	If the interlock is not asserted, returns `0`. If the interlock is asserted, returns `1`.

Also see

None

:OUTPut[1][:STATe]

This command enables or disables the source output.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0 (OFF)

Usage

```
:OUTPut[1][:STATe] <b>
:OUTPut[1][:STATe]?
```

`<b>`	Turn source off: `0` or `OFF` Turn source on: `1` or `ON`

Details

When the output is switched on, the instrument sources either voltage or current, as set by `[:SOURce[1]]:FUNCtion[:MODE]`.

Example

`:OUTP ON`	Switch the source output of the instrument to on.

Also see

:SOURce[1]:FUNCtion[:MODE] (on page 6-72)

ROUTe subsystem

The ROUTe subsystem selects which set of input and output terminals to enable (front panel or rear panel).

:ROUTe:TERMinals

This command determines which set of input and output terminals the instrument is using.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	FRON

Usage

```
:ROUTe:TERMinals <location>
:ROUTe:TERMinals?
```

<location>	Measure using the front-panel input and output terminals: `FRONt` Measure using the rear-panel input and output terminals: `REAR`

Details

This command selects which set of input and output terminals the instrument uses. You can select front panel or rear panel terminals.

If the output is turned on when you change from one set of terminals to the other, the output is turned off.

Example

`:ROUT:TERM REAR` `:ROUT:TERM?`	Set the instrument to use the rear-panel terminals and query to verify. Output: `REAR`

Also see

None

SENSe1 subsystem

The Sense1 subsystem commands configure and control the measurement functions of the Model 2450.

Many of these commands are set for a specific function (current, voltage, or resistance). For example, you can program a range setting for each function. The settings are saved with that function.

[:SENSe[1]]:AZERo:ONCE

This command causes the instrument to autozero once.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:AZERo:ONCE
```

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

`AZER:ONCE`	Do a once time refresh of the reference and zero measurements.

Also see

[:SENSe[1]]:<function>:AZERo[:STATe] (on page 6-46)

[:SENSe[1]]:CONFiguration:LIST:CATalog?

This command returns the name of one measure configuration list.

Type	Affected by	Where saved	Default value
Query	Not applicable	Not applicable	Not applicable

Query

```
[:SENSe[1]]:CONFiguration:LIST:CATalog?
```

Details

You can use this command to see the names of measure configuration lists stored on the instrument.

This command returns one name each time you send it. This command returns an empty string when there are no more names to return. If the command returns an empty string the first time you send it, no measure configuration lists have been created for the instrument.

Examples

`:SENSe:CONFiguration:LIST:CATalog?`	Send this command to receive the name of one measure configuration list stored on the instrument. Keep sending this command until it returns `NULL` to get all stored lists.

Also see

Configuration lists (on page 3-33)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 6-37)

[:SENSe[1]]:CONFiguration:LIST:CREate

This command creates an empty measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:CREate <name>
```

`<name>`	A string that represents the name of a measure configuration list

Details

This command creates an empty configuration list. To add configuration points to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list, use a saved setup to store the instrument settings, which include defined configuration lists.

Example

`:SENS:CONF:LIST:CRE "MyMeasList"`
Creates a measure configuration list named `MyMeasList`.

Also see

*SAV (on page 6-2)
Configuration lists (on page 3-33)
[:SENSe[1]]:CONFiguration:LIST:STORe (on page 6-41)

[:SENSe[1]]:CONFiguration:LIST:DELete

This command deletes a measure configuration list.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:DELete <name>
[:SENSe[1]]:CONFiguration:LIST:DELete <name>, <point>
```

`<name>`	A string that represents the name of a measure configuration list
`<point>`	A specific configuration point in the configuration list

Details

Deletes a configuration list. If the point parameter is not specified, the entire configuration list is deleted. If the point parameter is specified, only the specified configuration point in the list is deleted.

Examples

`:SENSe:CONF:LIST:DELete "myMeasList"`	Deletes a configuration list named `myMeasList`.
`:SENSe:CONF:LIST:DELete "myMeasList", 2`	Deletes configuration point 2, in a configuration list named `myMeasList`.

Also see

Configuration lists (on page 3-33)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 6-37)

[:SENSe[1]]:CONFiguration:LIST:QUERy?

This command returns a list of TSP commands that represent the parameters that are stored in the specified configuration point.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:QUERy? <name>, <point>
```

`<name>`	A string that represents the name of a measure configuration list
`<point>`	A specific configuration point in the configuration list

Details

This command can only return data for one configuration point. To get data for additional configuration points, send the command specifying different configuration points each time.

For additional information about the attributes this command returns, see Instrument settings stored in a measure configuration list (on page 3-36).

Example

`:SENS:CONF:LIST:QUER? "MyMeasList", 2`	
Returns the TSP commands that represent the settings in configuration point 2.	

Also see

Configuration lists (on page 3-33)

[:SENSe[1]]:CONFiguration:LIST:CREate (on page 6-37)

[:SENSe[1]]:CONFiguration:LIST:RECall

This command recalls a specific configuration point in a specific measure configuration list.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:RECall <name>
[:SENSe[1]]:CONFiguration:LIST:RECall <name>, <point>
```

`<name>`	A string that represents the name of a measure configuration list
`<point>`	A specific configuration point in the configuration list

Details

Use this command to recall the settings stored in a specific configuration point on a specific configuration list. If you do not specify a point when you send the command, it recalls the settings stored in the first configuration point on the specified configuration list.

Examples

`:SENSe:CONF:LIST:RECall "MyMeasList", 5`	Recalls configuration point 5 in a configuration list named MyMeasList.
`:SENSe:CONF:LIST:RECall "MyMeasList"`	Since a point was not specified, this command recalls configuration point 1 from a configuration list named `MyMeasList`.

Also see

Configuration lists (on page 3-33)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 6-37)
[:SENSe[1]]:CONFiguration:LIST:STORe (on page 6-41)

[:SENSe[1]]:CONFiguration:LIST:SIZE?

This command returns the size (number of configuration points) of a measure configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:SIZE? <name>
```

`<name>`	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration points) of a measure configuration list..

The size of the list is equal to the number of configuration points in a configuration list.

Examples

`:SENSe:CONF:LIST:SIZE? "MyMeasList"`	Returns the number of configuration points in a measure configuration list named `MyMeasList`.

Also see

Configuration lists (on page 3-33)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 6-37)

[:SENSe[1]]:CONFiguration:LIST:STORe

This command stores the active measure settings into the named configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Saved settings	Not applicable

Usage

```
[:SENSe[1]]:CONFiguration:LIST:STORe <name>
[:SENSe[1]]:CONFiguration:LIST:STORe <name>, <point>
```

`<name>`	A string that represents the name of a measure configuration list
`<point>`	A specific configuration point in the configuration list

Details

Use this command to store the active settings to a configuration point in a configuration list. If you do not include the <point> parameter, the configuration point will append to the end of the list.

Refer to Instrument settings stored in a measure configuration list (on page 3-36) for a complete list of measure settings that the instrument stores in a measure configuration list.

Example

`:SENSe:CONF:LIST:STOR "MyConfigList"`	Stores the active settings of the instrument to the end of the configuration list named `MyConfigList`.
`:SENSe:CONF:LIST:STOR "MyConfigList", 5`	Stores the active settings of the instrument to the configuration list named `MyConfigList` in configuration point 5.

Also see

Configuration lists (on page 3-33)
[:SENSe[1]]:CONFiguration:LIST:CREate (on page 6-37)

[:SENSe[1]]:COUNt

This command sets the number of measurements to be performed when a measurement is requested.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1

Usage

```
[:SENSe[1]]:COUNt <n>
[:SENSe[1]]:COUNt DEFault
[:SENSe[1]]:COUNt MINimum
[:SENSe[1]]:COUNt MAXimum
[:SENSe[1]]:COUNt?
[:SENSe[1]]:COUNt? DEFault
[:SENSe[1]]:COUNt? MINimum
[:SENSe[1]]:COUNt? MAXimum
```

`<n>`	The number of measurements (1 to 300,000)

Details

This command sets the number of measurements that are taken when a measurement is requested. This command does not affect the trigger model.

NOTE

To get better feedback from the instrument, use the Simple Loop trigger model template instead of using the count command.

Example

`COUN 10`	Make ten measurements.

Also see

:MEASure:<function>? (on page 6-4)

[:SENSe[1]]:<function>:AVERage:COUNt

This command sets the number of measurements that are averaged when filtering is enabled for the function measurements.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	10

Usage

```
[:SENSe[1]]:<function>:AVERage:COUNt <n>
[:SENSe[1]]:<function>:AVERage:COUNt?
[:SENSe[1]]:<function>:AVERage:COUNt? DEFault
[:SENSe[1]]:<function>:AVERage:COUNt? MINimum
[:SENSe[1]]:<function>:AVERage:COUNt? MAXimum
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The number of readings required for each filtered measurement (1 to 100)

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. The larger the filter count, the more filtering that is performed.

NOTE

If you send this command without the `<function>` parameter, it sets the filter count for all functions.

Example 1

`CURR:AVER:COUNT 10`
`CURR:AVER:TCON MOV`
`CURR:AVER ON`
	For current measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 2

`RES:AVER:COUNT 10`
`RES:AVER:TCON MOV`
`RES:AVER ON`
	For resistance measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 3

`VOLT:AVER:COUNT 10`
`VOLT:AVER:TCON MOV`
`VOLT:AVER ON`
	For voltage measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 4-22)
[:SENSe[1]]:<function>:AVERage[:STATe] (on page 6-44)
[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 6-45)

[:SENSe[1]]:<function>:AVERage[:STATe]

This command enables or disables the averaging filter for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
[:SENSe[1]]:<function>:AVERage[:STATe] <state>
[:SENSe[1]]:<function>:AVERage[:STATe]?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<state>`	The filter status; set to one of the following values: • Disable the averaging filter: `0` or `OFF` • Enable the averaging filter: `1` or `ON`

Details

This command enables or disables the averaging filter. When this is enabled, the measurements for the selected measurement function are averaged as set by the filter count and filter type.

NOTE

If you send this command without the `<function>` parameter, it sets the state of the averaging filter for all functions.

Example 1

`CURR:AVER:COUNT 10`
`CURR:AVER:TCON MOV`
`CURR:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 2

`RES:AVER:COUNT 10`
`RES:AVER:TCON MOV`
`RES:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 3

`VOLT:AVER:COUNT 10`
`VOLT:AVER:TCON MOV`
`VOLT:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 4-22)
[:SENSe[1]]:<function>:AVERage:COUNt (on page 6-43)
[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 6-45)

[:SENSe[1]]:<function>:AVERage:TCONtrol

This command set the type of averaging filter that is used for measurements for the selected function when the measurement filter is enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	REP

Usage

```
[:SENSe[1]]:<function>:AVERage:TCONtrol <type>
[:SENSe[1]]:<function>:AVERage:TCONtrol?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<type>`	The filter type to use when filtering is enabled; set to one of the following values: • Repeating filter: `REPeat` • Moving filter: `MOVing`

Details

This command selects the type of averaging filter: repeating average or moving average.

When the repeating average filter is selected, a set of measurements are made, which are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack has to be completely filled before an averaged sample can be produced.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

Note that when the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results, but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

NOTE

If you send this command without the `<function>` parameter, it sets the filter type for all functions.

Example 1

`CURR:AVER:COUNT 10`
`CURR:AVER:TCON MOV`
`CURR:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 2

`RES:AVER:COUNT 10`
`RES:AVER:TCON MOV`
`RES:AVER ON`
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Example 3

`VOLT:AVER:COUNT 10`
`VOLT:AVER:TCON MOV`
`VOLT:AVER ON`
	For voltage measurements, set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 4-22)
[:SENSe[1]]:<function>:AVERage:COUNt (on page 6-43)
[:SENSe[1]]:<function>:AVERage[:STATe] (on page 6-44)

[:SENSe[1]]:<function>:AZERo[:STATe]

This command enables or disables of the internal reference measurements (autozero) of the source-measure unit.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1 (ON)

Usage

```
[:SENSe[1]]:<function>:AZERo[:STATe] <state>
[:SENSe[1]]:<function>:AZERo[:STATe]?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<state>`	The status of autozero: • The instrument automatically checks reference and zero measurements and performs autozero when needed: `ON` or `1` • Disable autozero: `OFF` or `0`

Details

The analog-to-digital converter (ADC) uses a ratiometric A/D conversion technique. To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The Model 2450 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

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 disable autozero. You can use `[:SENSe[1]]:AZERo:ONCE` to force a one-time refresh of the reference measurements.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

NOTE

If you send this command without the `<function>` parameter, it sets autozero for all functions.

Example

`VOLT:AZER OFF`	Sets autozero off for voltage measurements.

Also see

Automatic reference measurements (on page 2-108)
[:SENSe[1]]:AZERo:ONCE (on page 6-36)

[:SENSe[1]]:<function>:DELay:USER<n>

This command sets a user-defined delay that can be used in the trigger model.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0.000000E+00

Usage

```
[:SENSe[1]]:<function>:DELay:USER<n> <delayTime>
[:SENSe[1]]:<function>:DELay:USER<n>?
[:SENSe[1]]:<function>:DELay:USER<n>? DEFault
[:SENSe[1]]:<function>:DELay:USER<n>? MAXimum
[:SENSe[1]]:<function>:DELay:USER<n>? MINimum
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The number that identifies this user delay (1 to 5)
`<delayTime>`	The time of the delay in seconds (0 to 10,000)

Details

To use this commands in a trigger model, assign the delay to the dynamic delay block.

Example

`:CURRent:DELay:USER1 .2`	Set user delay 1 to 0.2 seconds for current measurements.

Also see

:TRIGger:BLOCk:DELay:DYNamic (on page 6-140)

[:SENSe[1]]:<function>:NPLCycles

This command sets the time that the input signal is measured for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[:SENSe[1]]:<function>:NPLCycles <n>
[:SENSe[1]]:<function>:NPLCycles?
[:SENSe[1]]:<function>:NPLCycles? DEFault
[:SENSe[1]]:<function>:NPLCycles? MINimum
[:SENSe[1]]:<function>:NPLCycles? MAXimum
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The number of power-line cycles for each measurement: 0.01 to 10

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified in parameters that are based on the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50).

This command is set for the measurement of specific functions (current, resistance, or voltage).

The shortest amount of time (0.01 PLC) results in the fastest reading rate, but increases the reading noise and decreases the number of usable digits.

The longest amount of time (10 PLC) provides the lowest reading noise and more usable digits, but has the slowest reading rate.

Settings between the fastest and slowest number of PLCs are a compromise between speed and noise.

NOTE

If you send this command without the `<function>` parameter, it sets the NPLCs for all functions.

Example 1

`CURR:NPLC 0.5`	Sets the measurement time for current measurements to 0.0083 (0.5/60) seconds.

Example 2

`RES:NPLC 0.5`	Sets the measurement time for resistance measurements to 0.0083 (0.5/60) seconds.

Example 3

`VOLT:NPLC 0.5`	Sets the measurement time for voltage measurements to 0.0083 (0.5/60) seconds.

Also see

Using NPLCs to adjust speed and accuracy (on page 4-8)

[:SENSe[1]]:<function>:OCOMpensated

This command enables or disables offset compensation for resistance measurements.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
[:SENSe[1]]:RESistance:OCOMpensated <b>
[:SENSe[1]]:RESistance:OCOMpensated?
```

`<b>`	Turn on offset compensation: `1` or `ON` Turn off offset compensation: `0` or `OFF`

Details

The voltage offsets because of the presence of thermal EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

Example

`*RST`
`:SENS:FUNC "RES"`
`:SENS:RES:RANG:AUTO ON`
`:RES:OCOM ON`
`:COUNT 5`
`:OUTP ON`
`:TRAC:TRIG "defbuffer1"`
`:TRAC:DATA? 1, 5, "defbuffer1", SOUR, READ`
`:OUTP OFF`
	Reset the instrument.
	Set the measurement function to resistance.
	Turn offset-compensated ohms on.
	Turn the output on.
	Make a measurement.
	Turn the output off.

Also see

Offset-compensated ohms (on page 2-96)

[:SENSe[1]]:FUNCtion[:ON]

This command selects which type of measurement is active: current, voltage, or resistance.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	CURR

Usage

```
[:SENSe[1]]:FUNCtion[:ON] <function>
[:SENSe[1]]:FUNCtion[:ON]?
```

`<function>`	A string that contains the measurement function to make active: • Current: `CURRent[:DC]` • Voltage: `VOLTage[:DC]` • Resistance: `RESistance`

Details

Set this command to the type of measurement you want to make.

Reading this attribute returns the function that is presently active.

Examples

`:FUNC "VOLTage"`	Make the voltage measurement function the active function.

Also see

None

[:SENSe[1]]:<function>:RANGe:AUTO

This command determines if the range for measurements of the selected function is selected manually or automatically.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Saved setup Measure configuration list	1 (ON)

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO <state>
[:SENSe[1]]:<function>:RANGe:AUTO?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<state>`	Set the measurement range manually: `0` or `OFF` Set the measurement range automatically: `1` or `ON`

Details

This command determines how the measurement range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was selected automatically.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement. The instrument sets the range when a measurement is requested.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

`RES:RANG:AUTO ON`	Set the range to be selected automatically for resistance measurements.

Also see

[:SENSe[1]]:<function>:RANGe[:UPPer] (on page 6-53)
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 6-51)

[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit

This command selects the lower limit for measurements of the selected function when the range is selected automatically.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Current: 1e-8 Voltage: 20 Resistance: 20

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit <n>
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit?
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit? MINimum
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit? DEFault
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit? MAXimum
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The lower limit: • Current: 1e-8 to 1 amps • Voltage: 0.02 to 200 volts • Resistance: 2 to 2.0e8 ohms

Details

You can use this command when automatic range selection is enabled. It prevents the instrument from selecting a range that is below this limit. Because the lowest ranges generally require longer settling times, setting the low limit that is appropriate for your application but above the lowest possible range can make measurements require less settling time.

The lower limit must be less than the upper limit.

While you can send any value when you send this command, the instrument select the next highest range value. For example, if you send 15 for the lowest volt range, the instrument will be set to the 20 V range as the low limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`:VOLT:RANG:AUTO:LLIM 15` `:VOLT:RANG:AUTO:LLIM?`	Set the low range for voltage measurements to 20 V. Output: `2.000000E+01`

Also see

Ranges (on page 2-101)
[:SENSe[1]]:<function>:RANGe:AUTO (on page 6-50)

[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit

When autorange is selected, this command selects the upper limit for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Resistance: 2e5

Usage

```
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit <n>
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit?
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit? DEFault
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit? MINimum
[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit? MAXimum
```

`<function>`	The measurement function to which this setting applies: • Voltage (query only): `VOLTage[:DC]` • Resistance: `RESistance`
`<n>`	The upper limit: • Voltage: 0.02 to 200 volts • Resistance: 20 to 2.0e8 ohms

Details

You can use this command when automatic range selection is enabled to put an upper bound on the range that is used for resistance measurements.

The upper limit must be more than the lower limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`:SENSe:RESistance:RANGe:AUTO:ULIMit 20`	Set the upper limit to 20 ohms.

Also see

[:SENSe[1]]:<function>:RANGe:AUTO (on page 6-50)
[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 6-51)

[:SENSe[1]]:<function>:RANGe[:UPPer]

This command sets the positive full-scale value of the measurement range for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Current: 1e-4 Resistance: 2e5 Voltage: 20

Usage

```
[:SENSe[1]]:<function>:RANGe[:UPPer] <n>
[:SENSe[1]]:<function>:RANGe[:UPPer]?
[:SENSe[1]]:<function>:RANGe[:UPPer]? DEFault
[:SENSe[1]]:<function>:RANGe[:UPPer]? MINimum
[:SENSe[1]]:<function>:RANGe[:UPPer]? MAXimum
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	Set this command to a specific value or a preset value: Current: 1e-08 to 1 amps Resistance: 20 to 2e8 ohms Voltage: 0.02 to 200 volts

Details

When you assign a range value, the instrument is set on a fixed range that is large enough to measure the assigned value. The instrument selects the best range for measuring the maximum expected value.

This command is primarily intended to eliminate the time that is required by the instrument to select an automatic range.

Note that when you select a fixed range, an overrange condition can occur.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range, regardless of measurement range setting. However, the setting for the measure range is retained, and when the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the measurement function.

When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using. If you change the range while the output is off, the instrument does not update the hardware settings, but if you read the range setting, the return is the setting that will be used when the output is turned on. If you set a range while the output is on, the new setting takes effect immediately. When you assign a specific value to this command, the instrument is set on a fixed range that is large enough to measure the assigned value. The instrument selects the best range for measuring a value of `<n>`.

For example, for current measurements, if you expect a reading of approximately 50 mA, set `<n>` to 0.05 (or 50e-3) to select the 100 mA range. For voltage measurements, if you expect a reading of approximately 50 mV, let `<n>` = 0.05 (or 50e-3) to select the 200 mV range.

This command is primarily intended to eliminate the time that is required by the instrument to select an automatic range.

Note that when you select a fixed range, an overrange condition can occur.

This setting is ignored if the source function is the same as the measurement function. In that situation, the measurement range is the same as the source range, regardless of the setting of `[:SENSe[1]]:<function>:RANGe[:UPPer]`. However, the setting for the measurement range is retained, and when the source function is changed, the retained measurement range is used.

When you send the `[:SENSe[1]]:<function>:RANGe[:UPPer]` command, `[:SENSe[1]]:<function>:RANGe:AUTO` for the selected function is automatically set to disabled.

When you query this attribute, the instrument returns the positive full-scale value of the measurement range that the instrument is presently using. If you change the range while the output is off, the instrument does not update the hardware settings, but if you query the range setting, the return is the setting that will be used when the output is turned on. If you set a range while the output is on, the new setting takes effect immediately.

Example 1

`:SENS:CURR:RANG 10E-6`	Select the 10 μA range.

Example 2

`:SENS:RES:RANG 2E6`	Select the 2 MΩ range.

Example 3

`:SENS:VOLT:RANG 50e-3`	Select the 200 mV range.

Also see

Ranges (on page 2-101)
[:SENSe[1]]:<function>:RANGe:AUTO (on page 6-50)

[:SENSe[1]]:<function>:RELative

This command specifies the relative offset value for the measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0.0

Usage

```
[:SENSe[1]]:<function>:RELative <n>
[:SENSe[1]]:<function>:RELative?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<n>`	The relative offset value: • Current: −1 to 1 • Resistance: –200000000 to 200000000 • Voltage: −200 to 200

Details

This command specifies the relative offset value that is used for measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

Example

`CURR:REL .5` `CURR:REL:STAT ON`	Set the relative offset for current measurements to 0.5. Enable relative offset.

Also see

Relative offset (on page 3-1)
[:SENSe[1]]:<function>:RELative:ACQuire (on page 6-56)
[:SENSe[1]]:<function>:RELative:STATe (on page 6-57)

[:SENSe[1]]:<function>:RELative:ACQuire

This command acquires an internal measurement to store as the relative offset value.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

```
[:SENSe[1]]:CURRent[:DC]:RELative:ACQuire
[:SENSe[1]]:RESistance:RELative:ACQuire
[:SENSe[1]]:VOLTage[:DC]:RELative:ACQuire
```

Details

This command triggers the Model 2450 to make a new measurement for the selected function. This measurement is then stored as the new relative offset level setting.

When you send this command, the measurement is made without applying any math, limit test, or filter settings, even if they are set. It is a reading as if these settings are disabled.

After executing this command, you can use the `[:SENSe[1]]:<function>:RELative` command to see the last relative level value that was acquired or that was set.

If an error occurs during the measurement, the relative offset level remains at the last valid setting.

Example

FUNC "RES" RES:REL:ACQ RES:REL? RES:REL:STAT ON	Switch to resistance measurements. Acquire a relative offset value for resistance measurements. Query for the offset value. Turn relative offset on.

Also see

[:SENSe[1]]:<function>:RELative (on page 6-55)
[:SENSe[1]]:<function>:RELative:STATe (on page 6-57)

[:SENSe[1]]:<function>:RELative:STATe

This command enables or disables the relative offset value.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
[:SENSe[1]]:<function>:RELative:STATe <b>
[:SENSe[1]]:<function>:RELative:STATe?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<b>`	Disable the relative offset: `0` or `OFF` Enable the relative offset: `1` or `ON`

Details

This attribute enables or disables relative measurements. When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value calculated when you acquire the relative offset value.

Each returned measured relative reading is the result of the following calculation:

Display value = Actual measured value - Relative offset value

Example

`SENS:FUNC "VOLT"`
`SENS:VOLT:REL 5`
`:SENSe:VOLT:REL:STATe ON`
	Set the measurement function to volts with a relative offset of 5 V and enable the relative offset function.

Also see

[:SENSe[1]]:<function>:RELative (on page 6-55)
[:SENSe[1]]:<function>:RELative:ACQuire (on page 6-56)

[:SENSe[1]]:<function>:RSENse

This command selects local (2-wire) or remote (4-wire) sensing.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
[:SENSe[1]]:<function>:RSENse <b>
[:SENSe[1]]:<function>:RSENse?
```

`<function>`	The measurement function to which this setting applies: • Current: `CURRent[:DC]` • Resistance: `RESistance` • Voltage: `VOLTage[:DC]`
`<b>`	Disable remote sensing (2-wire): `0` or `OFF` Enable remote sensing (4-wire): `1` or `ON`

Details

This command determines if 2-wire (local) or 4-wire (remote) sensing is used.

When you use 4-wire sensing, voltages are measured at the device under test (DUT). For the source voltage, if the sensed voltage is lower than the programmed amplitude, the voltage source increases the voltage until the sensed voltage is the same as the programmed amplitude. This compensates for IR drop in the output test leads.

Using 4-wire sensing with voltage measurements eliminates any voltage drops that may be in the test leads between the Model 2450 and the DUT.

When you are using 2-wire sensing, voltage is measured at the output connectors.

When you are measuring resistance, you can enable 4-wire sensing to make 4-wire resistance measurements.

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

Example

`VOLT:RSEN ON`	Set the remote sense for voltage measurements.

Also see

Two-wire local sense connections (on page 2-75)
Four-wire remote sense connections (on page 2-76)

[:SENSe[1]]:<function>:UNIT

This command describes the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Current: AMP Voltage: VOLT Resistance: OHM Power: WATT

Usage

```
[:SENSe[1]]:CURRent:UNIT <currentMeasure>
[:SENSe[1]]:CURRent:UNIT?
[:SENSe[1]]:VOLTage:UNIT <voltageMeasure>
[:SENSe[1]]:VOLTage:UNIT?
```

`<currentMeasure>`	`OHM`, `WATT`, or `AMP`
`<voltageMeasure>`	`OHM`, `WATT`, or `VOLT`

Details

The change in measurement units is displayed when the next measurement occurs.

Example

`VOLT:UNIT WATT`	Changes the front-panel display and buffer readings for voltage measurements to be displayed as power readings in watts.

Also see

None

SOURce subsystem

The commands in the SOURce subsystem configure and control the current source and voltage source.

:SOURce[1]:CONFiguration:LIST:CATalog?

This command returns the name of one source configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Query

```
:SOURce:CONFiguration:LIST:CATalog?
```

Details

You can use this command to see the names of source configuration lists stored on the instrument.

This command returns one name each time you send it. This command returns an empty string if there are no more names to return. If the command returns an empty string the first time you send it, no source configuration lists have been created for the instrument.

Examples

`:SOUR:CONF:LIST:CAT?`	Send this command to return the name of one source configuration list stored on the instrument. To get all stored configuration lists, resend this command until it returns an empty string.

Also see

Configuration lists (on page 3-33)
:SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)

:SOURce[1]:CONFiguration:LIST:CREate

This command creates an empty source configuration list for sourcing.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:CREate <name>
```

`<name>`	A string that represents the name of a source configuration list

Details

This command creates an empty configuration list. To add configuration points to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list, use a saved setup to store the instrument settings, which include defined configuration lists.

Examples

`:SOURce:CONF:LIST:CRE "MySourceList"`
Creates a source configuration list named `MySourceList`.

Also see

Configuration lists (on page 3-33)
*SAV (on page 6-2)
:SOURce[1]:CONFiguration:LIST:STORe (on page 6-65)

:SOURce[1]:CONFiguration:LIST:DELete

This command deletes a source configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:DELete <name>
:SOURce[1]:CONFiguration:LIST:DELete <name>, <point>
```

`<name>`	A string that represents the name of a source configuration list
`<point>`	A specific configuration point in the configuration list

Details

Deletes a configuration list. If the point parameter is not specified, the entire configuration list is deleted. If the point parameter is specified, only the specified configuration point in the list is deleted.

Examples

`:SOURce:CONF:LIST:DEL "MySourceList"`	Deletes a configuration list named `MySourceList`.
`:SOURce:CONF:LIST:DEL "MySourceList", 2`	Deletes configuration point 2 in the configuration list named `MySourceList`.

Also see

Configuration lists (on page 3-33)
:SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)

:SOURce[1]:CONFiguration:LIST:QUERy?

This command returns a list of TSP commands that represent the parameters that are stored in the specified configuration point.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:QUERy? <name>, <point>
```

`<name>`	A string that represents the name of a source configuration list
`<point>`	A specific configuration point in the configuration list

Details

This command can only return data for one configuration point. To get data for additional configuration points, resend the command and specify different configuration points.

Refer to Instrument settings stored in a source configuration list (on page 3-38) for a complete list of source settings that the instrument stores in a source configuration list.

Example

`:SOUR:CONF:LIST:QUER? "MySourceList", 2`
Returns the TSP commands that represent the settings in configuration point 2.

Also see

Configuration lists (on page 3-33)
:SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)
Instrument settings stored in a source configuration list (on page 3-38)

:SOURce[1]:CONFiguration:LIST:RECall

This command recalls a specific configuration point in a specific source configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:RECall <name>, <point>
```

`<name>`	A string that represents the name of a source configuration list
`<point>`	A specific configuration point in the configuration list

Details

Use this command to recall the settings stored in a specific configuration point on a specific configuration list. If you do not specify a point when you send the command, it recalls the settings stored in the first configuration point on the specified configuration list.

Examples

`:SOURce:CONF:LIST:REC "MySourceList", 5`	Recalls configuration point 5 in a configuration list named MySourceList.
`:SOURce:CONF:LIST:RECall "MySourceList"`	Since a point was not specified, this command recalls configuration point 1 from a configuration list named `MySourceList`.

Also see

Configuration lists (on page 3-33)
:SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)

:SOURce[1]:CONFiguration:LIST:SIZE?

This command returns the size (number of configuration points) of a source configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Query

```
:SOURce[1]:CONFiguration:LIST:SIZE? <name>
```

`<name>`	A string that represents the name of a source configuration list

Details

This command returns the size (number of configuration points) of a source configuration list.

Examples

`:SOURce:CONFigure:LIST:SIZE? "MySourceList"`	Returns the number of configuration points in a source configuration list named `MySourceList`.

Also see

Configuration lists (on page 3-33)
:SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)

:SOURce[1]:CONFiguration:LIST:STORe

This command stores the active source settings into the named configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Saved settings	Not applicable

Usage

```
:SOURce[1]:CONFiguration:LIST:STORe <name>, <point>
```

`<name>`	A string that represents the name of a source configuration list
`<point>`	A specific configuration point in the configuration list

Details

Use this command to store the active settings to a configuration point in a configuration list. If you do not include the <point> parameter, the configuration point will append to the end of the list.

Refer to Instrument settings stored in a source configuration list (on page 3-38) for information about the settings this command stores.

Example

`:SOURce:CONF:LIST:CRE "biasLevel"`
`:SOURce:FUNC VOLT`
`:SOURce:VOLT:LEV 5`
`:SOURce:CONF:LIST:STORE "biasLevel"`
	Create a configuration list named `biasLevel`.
	Set the source function to voltage and the source voltage level to 5 volts.
	Store the configuration list and append it to the end of the `biasLevel` configuration list.

Also see

:SOURce[1]:CONFiguration:LIST:CREate (on page 6-61)

:SOURce[1]:<function>:DELay

This command sets the source delay.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	Not applicable

Usage

```
:SOURce[1]:<function>:DELay <n>
:SOURce[1]:<function>:DELay?
:SOURce[1]:<function>:DELay? DEFault
:SOURce[1]:<function>:DELay? MINimum
:SOURce[1]:<function>:DELay? MAXimum
```

`<function>`	The source function to which this setting applies: • Current: `CURRent` • Voltage: `VOLTage`
`<n>`	The delay in seconds (0 to 4)

Details

This command sets a delay for the selected source function. This delay is in addition to normal settling times.

After the programmed source is turned on, this delay allows the source level to settle before a measurement is taken.

If source autodelay is on, if you set a specific delay, it is turned off.

If source autodelay is on, the manual source delay setting is not saved in the source configuration list.

NOTE

If you send this command without the `<function>` parameter, it sets the delay for all functions.

Example

`SOUR:VOLT:DELay DEF`	Set the delay for the voltage source to the default value.

Also see

:SOURce[1]:<function>:DELay:AUTO (on page 6-67)

:SOURce[1]:<function>:DELay:AUTO

This command enables or disables the autodelay that occurs when the source is turned on.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	1 (ON)

Usage

```
:SOURce[1]:<function>:DELay:AUTO <state>
:SOURce[1]:<function>:DELay:AUTO?
```

`<function>`	The source function to which this setting applies: • Current: `CURRent` • Voltage: `VOLTage`
`<state>`	Disable the source auto delay: `ON` or `0` Enable the source auto delay: `OFF` or `1`

Details

When auto delay is turned on, the actual delay that is set depends on the range.

When source autodelay is on, if you set a source delay, the autodelay is turned off.

Example

`SOUR:CURR:DEL:AUTO OFF`	Turn off auto delay when current is being sourced.

Also see

:SOURce[1]:<function>:DELay (on page 6-66)

:SOURce[1]:<function>:DELay:USER<n>

This command sets a user-defined delay that can be used in the trigger model.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0.000000E+00

Usage

```
:SOURce[1]:<function>:DELay:USER<n> <delayTime>
:SOURce[1]:<function>:DELay:USER<n>?
:SOURce[1]:<function>:DELay:USER<n>? DEFault
:SOURce[1]:<function>:DELay:USER<n>? MAXimum
:SOURce[1]:<function>:DELay:USER<n>? MINimum
```

`<function>`	The source function to which this setting applies: • Current: `CURRent` • Voltage: `VOLTage`
`<n>`	The number that identifies this user delay (1 to 5)
`<delayTime>`	The time of the delay in seconds (0 to 10,000)

Details

To use this commands in a trigger model, assign the delay to the dynamic delay block.

Example

`:SOUR:VOLT:DEL:USER1 5`
`:TRIG:BLOC:SOUR:STAT 1, ON`
`:TRIG:BLOC:DEL:DYN 2, SOUR1`
 	
`:TRIG:BLOC:MEAS 3`
`:TRIG:BLOC:SOUR:STAT 4, OFF`
`:TRIG:BLOC:BRAN:COUN 5, 10, 1`
`:INIT`
	Set user delay for source 1 to 5 seconds.
	Set trigger block 1 to turn the source output on.
	Set trigger block 2 to a dynamic delay that calls source user delay 1.
	Set trigger block 3 to make a measurement.
	Set trigger block 4 to turn the source output off.
	Set trigger block 5 to branch to block 1 ten times.
	Start the trigger model.

Also see

:TRIGger:BLOCk:DELay:DYNamic (on page 6-140)

:SOURce[1]:<function>:HIGH:CAPacitance

This command enables or disables high capacitance feature.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0 (OFF)

Usage

```
:SOURce[1]:<function>:HIGH:CAPacitance <b>
:SOURce[1]:<function>:HIGH:CAPacitance?
```

`<function>`	The source function to which this setting applies: • Current: `CURRent` • Voltage: `VOLTage`
`<b>`	Turn high capacitance feature on: `ON` or `1` Turn high capacitance feature off: `OFF` or `0`

Details

When the instrument is measuring low current and is driving a capacitive load, you may see overshoot, ringing, and instability. You can enable the high capacitance feature to minimize these problems.

High-capacitance settings apply when operating using the 10 nA through the 100 mA current ranges. When operating using the 1 A range, the high-capacitance setting will not affect the instrument rise time or current measure settling time.

Example

`SOUR:CURR:HIGH:CAP ON`	Turn the high capacitance feature on when sourcing current.

Also see

High capacitance operation (on page 4-21)

:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]

This command immediately selects a fixed amplitude for the selected source function.

Type	Affected by	Where saved	Default value
Command	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	0

Usage

```
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] <n>
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]?
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]? DEFault
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]? MINimum
:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude]? MAXimum
```

`<function>`	The source function to which this setting applies: • Current: `CURRent` • Voltage: `VOLTage`
`<n>`	Current: −1.05 A to 1.05 A Voltage: −210 V to 210 V

Details

This command sets the output level of the voltage or current source. If the output is on, the new level is sourced immediately.

The sign of the source level dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

If a manual source range is selected, the level cannot exceed the specified range. For example, if the voltage source is on the 2 V range (auto range is disabled), you cannot set the voltage source amplitude to 3 V. When auto range is selected, the amplitude can be set to any level.

Example

`SOUR:FUNC VOLT` `SOUR:VOLT 1`	Set the instrument to source voltage and set it to source 1 V.

Also see

:SOURce[1]:<function>:RANGe (on page 6-74)
:SOURce[1]:<function>:RANGe:AUTO (on page 6-75)

:SOURce[1]:<function>:<x>LIMit[:LEVel]

This command selects the source limit for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	Voltage: 105 μA Current: 21 V

Usage

```
:SOURce[1]:CURRent:VLIMit[:LEVel] <n>
:SOURce[1]:CURRent:VLIMit[:LEVel]?
:SOURce[1]:CURRent:VLIMit[:LEVel]? DEFault
:SOURce[1]:CURRent:VLIMit[:LEVel]? MINimum
:SOURce[1]:CURRent:VLIMit[:LEVel]? MAXimum
:SOURce[1]:VOLTage:ILIMit[:LEVel] <n>
:SOURce[1]:VOLTage:ILIMit[:LEVel]?
:SOURce[1]:VOLTage:ILIMit[:LEVel]? DEFault
:SOURce[1]:VOLTage:ILIMit[:LEVel]? MINimum
:SOURce[1]:VOLTage:ILIMit[:LEVel]? MAXimum
```

`<n>`	Voltage: –1.05 A to 1.05 A Current: –210 to 210: Range for a limit set by the user

Details

This command sets the source limit for measurements. The Model 2450 cannot source levels that exceed this limit.

The values that can be set for this command are limited by the setting for the overvoltage protection limit.

This value can also be limited by the measurement range. If a specific measurement range is set, the limit must be more than 0.1% of the measurement range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

If you change the source range to a level that is not appropriate for this limit, the source limit is changed to a limit that is appropriate to the range and a warning is generated.

Example

`:SOUR:CURR:VLIM 15`	Set the voltage limit to 15 V

Also see

:SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73)
:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped? (on page 6-72)

:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped?

This command indicates if the source exceeded the limits that were set for the selected measurements.

Type	Affected by	Where saved	Default value
Query	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:CURRent:VLIMit[:LEVel]:TRIPped?
:SOURce[1]:VOLTage:ILIMit[:LEVel]:TRIPped?
```

Details

You can use this command check the limit state of the source.

If the limits were exceeded, the instrument clamps the source to keep the source within the set limits.

If the source did not exceed the set limits, the return is `0`. If the source did exceed the set limits, the return is `1`.

Example 1

`SOUR:CURR:VLIM:TRIP?`	Returns a value that indicates whether or not the source exceeded the current limits.

Example 2

`SOUR:VOLT:ILIM:TRIP?`	Return value indicates whether or not the source has exceeded the voltage limits.

Also see

:SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 6-71)

:SOURce[1]:FUNCtion[:MODE]

This command contains the source function, which can be voltage or current.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	`VOLT`

Usage

```
:SOURce[1]:FUNCtion[:MODE] <function>
:SOURce[1]:FUNCtion[:MODE]?
```

`<function>`	Voltage source function: `VOLTage` Current source function: `CURRent`

Details

Setting this command configures the instrument as either a voltage source or a current source.

Examples

`SOUR:FUNC CURR` `SOUR:FUNC?`	Set the source function of the instrument to be a current source and query the source function. Output: `CURR`

Also see

None

:SOURce[1]:<function>:PROTection[:LEVel]

This command sets the overvoltage protection setting of the source output.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	NONE

Usage

```
:SOURce[1]:VOLTage:PROTection[:LEVel] <x>
:SOURce[1]:VOLTage:PROTection[:LEVel]?
```

`<n>`	The overvoltage protection level, set as `<x>`, where `<x>` is `PROT2`, `PROT5`, `PROT10`, `PROT20`, `PROT40`, `PROT60`, `PROT80`, `PROT100`, `PROT120`, `PROT140`, `PROT160`, `PROT180`, or `NONE`

Details

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced.

This protection is in effect for both positive and negative output voltages.

When this attribute is used in a test sequence, it should be set before the turning the source on.

WARNING

Even with the overvoltage protection set to the lowest value, never touch anything connected to the terminals of the Model 2450 when the output is on. Always assume that a hazardous voltage (>30 V_{rms}) is present when the output is on. To prevent damage to the device under test (DUT) and external circuitry, do not set the voltage source to levels that exceed the overvoltage protection value.

Example

`sour:volt:prot prot40` `sour:volt:prot?`	Set the voltage source protection to 40 V and query the value. The output is: `PROT40`

Also see

Overvoltage protection (on page 2-98)

:SOURce[1]:<function>:PROTection[:LEVel]:TRIPped?

This command indicates if the overvoltage source protection feature is active.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SOURce[1]:VOLTage:PROTection[:LEVel]:TRIPped?
```

Details

When overvoltage protection is active, the instrument restricts the maximum voltage level that the instrument can source.

If the voltage source does not exceed the set limits, the return is `0`. If the voltage source exceeds the set limits, the return is `1`.

Example

`SOUR:VOLT:PROT:TRIP?`	If overvoltage protection is active, the output is: `1`

Also see

Overvoltage protection (on page 2-98)
:SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73)

:SOURce[1]:<function>:RANGe

This command selects the range for the source for the selected source function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	Current: 1e-04 Voltage: 20 mV

Usage

```
:SOURce[1]:<function>:RANGe <n>
:SOURce[1]:<function>:RANGe?
:SOURce[1]:<function>:RANGe? DEFault
:SOURce[1]:<function>:RANGe? MINimum
:SOURce[1]:<function>:RANGe? MAXimum
```

`<function>`	The source function to which this setting applies: • Current: `CURRent` • Voltage: `VOLTage`
`<n>`	Set to the maximum expected voltage or current to be sourced; the ranges are: • Current: −1 A to 1 A • Voltage: −200 V to 200 V

Details

This command manually selects the measurement range for the specified source.

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 20 mV, 200 mV, 2 V, 20 V, and 200 V.

When you read this value, the instrument returns the positive full-scale value that the instrument is presently using.

This command is intended to eliminate the time required by the automatic range selection.

To select the range, you can specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 50 mV, send 0.05 (or 50e-3) to select the 200 mV range.

NOTE

If automatic range selection is set to on, when you select a specific range, automatic is set to off. To set the range to automatic selection, use the source autorange command.

Example

`:SOURce:VOLTage:RANGe 3`	Send this command to source levels around 3 V. This example selects the 20 V range for the voltage source.

Also see

:SOURce[1]:<function>:RANGe:AUTO (on page 6-75)

:SOURce[1]:<function>:RANGe:AUTO

This command determines if the range is selected manually or automatically for the selected source function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	1 (ON)

Usage

```
:SOURce[1]:CURRent:RANGe:AUTO <b>
:SOURce[1]:CURRent:RANGe:AUTO?
:SOURce[1]:VOLTage:RANGe:AUTO <b>
:SOURce[1]:VOLTage:RANGe:AUTO?
```

`<b>`	Disable automatic range: `0` or `OFF` Enable automatic range: `1` or `ON`

Details

This command indicates the state of the range for the selected source. When automatic source range is disabled, the source range is set manually.

When automatic source range is enabled, the instrument selects the range that is most appropriate for the value that is being sourced. The output level controls the range. If you read the range after the output level is set, the instrument returns the range that the instrument chose as appropriate for that source level.

If the source range is set to a specific value from the front panel or a remote command, the setting for automatic range is set to disabled.

Example

`SOUR:CURR:RANG:AUTO ON`	Enable the automatic source range.

Also see

:SOURce[1]:<function>:RANGe (on page 6-74)

:SOURce[1]:<function>:READ:BACK

This command determines if the instrument records the measured source value or the configured source value when making a measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Source configuration list	Save settings Source configuration list	1 (ON)

Query

```
:SOURce[1]:VOLTage:READ:BACK <b>
:SOURce[1]:VOLTage:READ:BACK?
:SOURce[1]:CURRent:READ:BACK <b>
:SOURce[1]:CURRent:READ:BACK?
```

`<b>`	Disable read back: `0` or `OFF` Enable read back: `1` or `ON`

Details

When you use the configured source value (source readback off), the instrument records and displays the value that was configured. When you use the actual source value (source readback on), the instrument measures the actual source value immediately before making the device under test measurement.

Using source readback results in more accurate measurements, but also a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the device-under-test measurement. When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the device-under-test measurement.

Example

`*RST`	Reset the instrument to default settings.
`TRAC:MAKE "MyBuffer", 100`	Make a buffer named "MyBuffer" that can hold 100 readings.
`SOUR:FUNC VOLT`	Set source function to voltage.
`SENS:FUNC "CURR"`	Set the measurement function to current.
`SOUR:VOLT:READ:BACK ON`	Set read back on.
`SOUR:VOLT 10`	
`COUNT 100`	Set the instrument to take 100 readings.
`OUTP ON`	Turn the output on.
`READ? "MyBuffer"`	Take a measurement (100 readings).
`OUTP OFF`	Turn the output off.
`TRAC:DATA? 1, 100, "MyBuffer", SOUR, READ`	Get the source values and measurements from the buffer.

Also see

:SOURce[1]:FUNCtion[:MODE] (on page 6-72)

:SOURce[1]:LIST:<function>

This command allows you to set up a list of custom values for a sweep.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:LIST:CURRent <list>
:SOURce[1]:LIST:CURRent?
:SOURce[1]:LIST:VOLTage <list>
:SOURce[1]:LIST:VOLTage?
```

`<list>`	Current: –1.05 A to 1.05 A Voltage: –210 V to 210 V See **Details**

Details

This command defines a list of up to 100 source values for a source list. This list is used by the `:SOURce[1]:SWEep:<function>:LIST` command to define the source values for the sweep.

When you start the sweep, the instrument sequentially sources each current or voltage value in the list. A measurement is performed at each source level.

If there is an existing list, it is replaced by the new list.

When you send this command, the instrument creates a source configuration list named `CurrCustomSweepList` if the function is set to current or `VoltCustomSweepList` if the function is set to voltage.

To add source values to an existing list, use the `:SOURce[1]:LIST:<function>:APPend` command.

Example

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SENS:CURR:RSEN OFF
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 1
SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5
SOUR:SWE:VOLT:LIST 1, 0.2
INIT
*WAI

TRAC:DATA? 1, 6, "defbuffer1", SOUR, READ
```

This example will source 1 V, 5 V, 1 V, 5 V, 1 V, 5 V and measure the resulting current at each voltage point. The time duration of each voltage point is 200 ms.

Also see

:SOURce[1]:LIST:<function>:APPend (on page 6-79)
:SOURce[1]:SWEep:<function>:LIST (on page 6-85)

:SOURce[1]:LIST:<function>:APPend

This command adds values to the source list for the selected source function.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:LIST:CURRent:APPend <list>
:SOURce[1]:LIST:VOLTage:APPend <list>
```

`<list>`	Current: –1.05 to 1.05 Voltage: –210 to 210

Details

This adds up to 100 values to the list created with `:SOURce[1]:LIST:<function>`. The new values are added to the end of the existing values. You can have a total of 2500 values in a list, but you must append them in groups of 100.

If the list does not exist, this command creates one.

Example

```
*RST
SENS:FUNC "CURR"
SENS:CURR:RANG:AUTO ON
SENS:CURR:RSEN OFF
SOUR:FUNC VOLT
SOUR:VOLT:RANG 20
SOUR:VOLT:ILIM 1
SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5
SOUR:LIST:VOLT:APP 1, 5, 1, 5, 1, 5
SOUR:SWE:VOLT:LIST 1, 0.2
INIT
*WAI

TRAC:DATA? 1, 12, "defbuffer1", SOUR, READ
```

This example will create a source configuration list (`VoltCustomSweepList`) and source 1 V, 5 V six times and measure the resulting current at each voltage point. The duration of each voltage point is 200 ms.

Also see

:SOURce[1]:LIST:<function> (on page 6-78)

:SOURce[1]:LIST:<function>:POINts?

This command queries the length of the source list for the selected source function.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Query

```
:SOURce[1]:LIST:CURRent:POINts?
:SOURce[1]:LIST:VOLTage:POINts?
```

Details

This command returns the length of the specified source list. The response message indicates the number of source values in the list.

Example

`*RST`
`SENS:FUNC "CURR"`
`SENS:CURR:RANG:AUTO ON`
`SENS:CURR:RSEN OFF`
`SOUR:FUNC VOLT`
`SOUR:VOLT:RANG 20`
`SOUR:VOLT:ILIM 1`
`SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5`
`SOUR:SWE:VOLT:LIST 1, 0.2`
`INIT`
`*WAI`
 	
`TRAC:DATA? 1, 6, "defbuffer1", SOUR, READ`
`SOUR:LIST:VOLT:POIN?`
	This example will source 1 V, 5 V, 1 V, 5 V, 1 V, 5 V and measure the resulting current at each voltage point. The time duration of each voltage point is 200 ms.
	Check the number of points in the list.
	Output:
	`6`

Also see

:SOURce[1]:LIST:<function> (on page 6-78)
:SOURce[1]:LIST:<function>:APPend (on page 6-79)

:SOURce[1]:SWEep:<function>:LINear

This command sets up a linear sweep for a set number of measurement points.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>
:SOURce[1]:SWEep:<function>:LINear <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, <bufferName>
```

`<function>`	Voltage sweep: `VOLTage` Current sweep: `CURRent`
`<start>`	The voltage or current source level at which the sweep starts: • Current: -1.05 to 1.05 • Voltage: -210 to 210
`<stop>`	The voltage or current at which the sweep stops: • Current: -1.05 to 1.05 • Voltage: -210 to 210
`<points>`	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
`<delay>`	The delay between measurement points; default is `-1`, which enables autodelay, or a specific delay value from 50 µs to 10,000 seconds, or 0 for no delay
`<count>`	The number of times to run the sweep; default is `1`: • Infinite loop: `0` • Finite loop: `1` to `268435455`
`<rangeType>`	The source range that is used for the sweep: • Most sensitive source range for each source level in the sweep: `AUTO` • Best fixed range: `BEST` (default) • Present source range for the entire sweep: `FIXed`
`<failAbort>`	Abort the sweep if the source limit is exceeded: `ON` (default) Complete the sweep if the source limit is exceeded: `OFF`
`<dual>`	Determines if the sweep runs from start to stop and then from stop to start: • Sweep from start to stop only: `OFF` (default) • Sweep from start to stop, then stop to start: `ON`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it creates a source configuration list and populates the trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

Example

`*RST`	Reset the instrument to its defaults.
`SOUR:FUNC VOLT`	Set the source function to voltage.

`SOUR:VOLT:RANG 20`
`SENS:FUNC "CURR"`
`SENS:CURR:RANG 100e-6`
	Set the source range to 20 V. Set the measure function to current with a range of 100 μA.

`SOUR:SWE:VOLT:LIN 0, 10, 20, 1e-3, 1, FIXED`	Set up a linear sweep that sweeps from 0 to 10 volts in 20 points with a source delay of 1 ms, a sweep count of 1, and a fixed source range.
`INIT`	Start the sweep.

Also see

Sweep operation (on page 3-51)

:SOURce[1]:SWEep:<function>:LINear:STEP

This command sets up a linear source sweep configuration list and trigger model with a fixed number of steps.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>, <failAbort>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>
:SOURce[1]:SWEep:<function>:LINear:STEP <start>, <stop>, <steps>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, <bufferName>
```

`<start>`	The voltage or current source level at which the sweep starts: • Current: -1.05 to 1.05 • Voltage: -210 to 210
`<stop>`	The voltage or current at which the sweep stops: • Current: -1.05 to 1.05 • Voltage: -210 to 210
`<steps>`	The step size at which the source level will change; step size must be greater than 0
`<delay>`	The delay between measurement points; default is `-1`, which enables autodelay, or a specific delay value from 50 μs to 10,000 seconds; or 0 for no delay
`<count>`	The number of times to run the sweep; default is `1`: • Infinite loop: `0` • Finite loop: `1` to `268435455`
`<rangeType>`	The source range that is used for the sweep: • Most sensitive source range for each source level in the sweep: `AUTO` • Best fixed range: `BEST` (default) • Present source range for the entire sweep: `FIXed`
`<failAbort>`	Abort the sweep if the source limit is exceeded: `ON` (default) Complete the sweep if the source limit is exceeded: `OFF`
`<dual>`	Determines if the sweep runs from start to stop and then from stop to start: • Sweep from start to stop only: `OFF` (default) • Sweep from start to stop, then stop to start: `ON`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Detail

When the sweep is started, the instrument sources a specific voltage or current voltage to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it creates a trigger model with a uniform series of ascending or descending voltage or current changes, called steps. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the stop level, which is calculated from the number of steps. A measurement is performed at each source step (including the start and stop levels). At this level, the instrument performs another measurement and then stops the sweep.

The instrument uses the step size parameter to determine the number of source level changes. The source level changes in equal steps from the start level to the stop level. To avoid a setting conflicts error, make sure the step size is greater than the start value and less than the stop value. To calculate the number of source-measure points in a sweep, use the following formula:

$$step = \frac{stop - start}{points - 1}$$

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

Example

`*RST`	Reset the instrument to its defaults.
`SOUR:FUNC CURR`	Set the source function to current.

`SOUR:CURR:RANGE 1`
`SENS:FUNC "VOLT"`
`SENS:VOLT:RANGE 20`
	Set the source range to 1 A. Set the measure function to voltage with a range of 20 V.

`SOUR:SWE:CURR:LIN:STEP -1.05, 1.05, .25, 10e-3, 1, FIXED`	Set up a linear step sweep that sweeps from -1.05 A to 1.05 A in 0.25 A increments with a source delay of 1 ms, a sweep count of 1, and a fixed source range. The name of the configuration list that is created for this sweep is `CurrLinearSweep`.
`INIT`	Start the sweep.

Also see

Sweep operation (on page 3-51)

:SOURce[1]:SWEep:<function>:LIST

This command sets up a sweep based on a configuration list, which allows you to customize the sweep.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LIST <startIndex>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>, <failAbort>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>, <failAbort>,
   <bufferName>
:SOURce[1]:SWEep:<function>:LIST <startIndex>, <delay>, <count>, <failAbort>,
   <bufferName>, <configListName>
```

`<function>`	The source function: • Current: `CURRent` • Voltage: `VOLTage`
`<startIndex>`	The index in the configuration list where the sweep starts; default is `1`
`<delay>`	The delay between measurement points; default is 0 for no delay or you can set a specific delay value from 50 µs to 10,000 seconds
`<count>`	The number of times to run the sweep; default is `1`: • Infinite loop: `0` • Finite loop: `1` to `268435455`
`<failAbort>`	Determines if the sweep is stopped immediately if a limit is exceeded; options are: • Abort the sweep if a limit is exceeded: `ON` • Complete the sweep even if a limit is exceeded: `OFF`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
`<configListName>`	The name of the configuration list that the sweep uses; this must be defined before sending this command; the default for voltage sweeps is `VoltCustomSweepList`; for current sweeps, `CurrCustomSweepList`

Details

This command allows you to set up a custom sweep, using a configuration list to specify the source levels.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

A configuration list must be created before you send this command. You can either use the `:SOURce[1]:LIST:<function>` to set up the configuration list, or you can create your own configuration list.

To run the sweep, initiate the trigger model.

Example 1

`*RST`
`SENS:FUNC "CURR"`
`SENS:CURR:RANG:AUTO ON`
`SENS:CURR:RSEN OFF`
`SOUR:FUNC VOLT`
`SOUR:VOLT:RANG 20`
`SOUR:VOLT:ILIM 1`
`SOUR:LIST:VOLT 1, 5, 1, 5, 1, 5`
`SOUR:SWE:VOLT:LIST 1, 0.2`
`INIT`
`*WAI`
 	
`TRAC:DATA? 1, 6, "defbuffer1", SOUR, READ`
	This example uses the `:SOURce[1]:LIST:<function>` command to set up the configuration list that is used by the sweep.
	This example will source 1 V, 5 V, 1 V, 5 V, 1 V, 5 V and measure the resulting current at each voltage point. The time duration of each voltage point is 200 ms.

Example 2

```
SOUR:CONF:LIST:CRE "biasLevel"
SOUR:FUNC VOLT
SENS:FUNC "CURR"
SOUR:VOLT:LEV 5
SOUR:CONF:LIST:STORE "biasLevel"
SOUR:SWE:VOLT:LIST 1, .001, 1, 1, "defbuffer2", "biasLevel"
INIT
```

This example uses a user-defined configuration list.
Create a configuration list named `biasLevel`. Set the source function to 5 volts and the measure function to current.
Store the configuration list.
Set up a voltage sweep that uses the configuration list, starting at index point 1 with a delay of 1 ms. The sweep is to abort if the source limit is exceeded, store data in `defbuffer2`, and use the configuration list `biasLevel`.

Also see

Configuration lists (on page 3-33)
:INITiate[:IMMediate] (on page 6-124)
:SOURce[1]:LIST:<function> (on page 6-78)
Sweep operation (on page 3-51)

:SOURce[1]:SWEep:<function>:LOG

This command sets up a logarithmic sweep for a set number of measurement points.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, <bufferName>
:SOURce[1]:SWEep:<function>:LOG <start>, <stop>, <points>, <delay>, <count>,
   <rangeType>, <failAbort>, <dual>, <bufferName>, <asymptote>
```

`<function>`	The source function: • `CURRent` • `VOLTage`
`<start>`	The voltage or current source level at which the sweep starts: • Current: 1 pA to 1.05 • Voltage: 1 pV to 210
`<stop>`	The voltage or current at which the sweep stops: • Current: 1 pA to 1.05 • Voltage: 1 pV to 210
`<points>`	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
`<delay>`	The delay between measurement points; default is `-1`, which enables autodelay, or a specific delay value from 50 μs to 10,000 seconds, or 0 for no delay
`<count>`	The number of times to run the sweep; default is 1: • Infinite loop: `0` • Finite loop: `1` to `268435455`
`<rangeType>`	The source range that is used for the sweep: • Most sensitive source range for each source level in the sweep: `AUTO` • Best fixed range: `BEST` (default) • Present source range for the entire sweep: `FIXed`
`<failAbort>`	Abort the sweep if the source limit is exceeded: `ON` (default) Complete the sweep if the source limit is exceeded: `OFF`
`<dual>`	Determines if the sweep runs from start to stop and then from stop to start: • Sweep from start to stop only: `OFF` (default) • Sweep from start to stop, then stop to start: `ON`

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
`<asymptote>`	Default is 0; see **Details**

Details

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it creates a trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

The asymptote changes the inflection of the sweep curve and allows it to sweep through zero. You can use the asymptote parameter to customize the inflection and offset of the source value curve. Setting this parameter to zero provides a conventional logarithmic sweep. The asymptote value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep. The asymptote value must not be equal to or between the starting and ending values. It must be outside the range defined by the starting and ending values.

A configuration list must be created before you send this command. You can either use the `:SOURce[1]:LIST:<function>` to set up the configuration list, or you can create your own configuration list.

To run the sweep, initiate the trigger model.

Example

`*RST`	Reset the instrument to its defaults.
`SOUR:FUNC VOLT`	Set the source function to voltage.
`SOUR:VOLT:RANG 20`	Set the source range to 20 V.
`SENS:FUNC "CURR"`	Set the measure function to current.
`SENS:CURR:RANG 100e-6`	Set the current range to 100 µA.
`SOUR:SWE:VOLT:LOG .1, 10, 20, 1e-3, 1, FIXED`	Set up a log sweep that sweeps from .1 to 10 volts in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range.
`INIT`	Start the sweep.

Also see

:INITiate[:IMMediate] (on page 6-124)
Sweep operation (on page 3-51)

STATus subsystem

The STATus subsystem controls the status registers of the Model 2450. For additional information on the status model, see Status model (on page C-1).

:STATus:OPERation:CONDition?

This command reads the Operation Event Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation:CONDition?
```

Details

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page C-15).

Example

`:STAT:OPER:COND?`	Returns the contents of the Operation Condition Register.

Also see

Operation Event Register (on page C-8)

:STATus:OPERation:ENABle

This command sets or reads the contents of the Operation Event Enable Register of the status model.

Type	Affected by	Where saved	Default value
Command and query	STATus:PRESet	Nonvolatile memory	0

Usage

```
:STATus:OPERation:ENABle <n>
:STATus:OPERation:ENABle?
```

`<n>`	The status of the operation status register

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

When sending binary, preface `<n>` with `#b`. When sending hexadecimal, preface `<n>` with `#h`. No preface is needed when sending decimal values.

Example

`:STAT:OPER:ENAB #b0101000000000000`	Sets the 12 and 14 bits of the operation status enable register using a decimal value.
	You could also send the decimal value:
	`:STAT:OPER:ENAB 20480`
	Or the hexadecimal value:
	`:STAT:OPER:ENAB #h5000`

Also see

Operation Event Register (on page C-8)

:STATus:OPERation[:EVENt]?

This command reads the Operation Event Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation[:EVENt]?
```

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example

`stat:oper?`	Returns the contents of the Operation Event Register of the status model.

Also see

Operation Event Register (on page C-8)

:STATus:OPERation:MAP

This command allows you to map event numbers to bits in the Operation Event Registers.

Type	Affected by	Where saved	Default value
Command and query	Instrument reset Power cycle	Not saved	Not applicable

Usage

```
:STATus:OPERation:MAP <bitNumber>, <setEvent>
:STATus:OPERation:MAP <bitNumber>, <setEvent>, <clearEvent>
:STATus:OPERation:MAP? <bitNumber>
```

bitNumber	The bit number to check
setEvent	The event number mapped to set this bit; 0 if no mapping
clearEvent	The event number mapped to clear this bit; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

The query requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers. When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`:STATus:OPERation:MAP 0, 4917, 4918`	When event 4917 (the buffer is 0% filled) occurs, bit 0 is set in the condition register and the event register of the Operation Event Register. When event 4918 (buffer is 100% filled) occurs, bit 0 in the condition register is cleared.

Also see

Programmable status register sets (on page C-5)

:STATus:PRESet

This command resets all bits in the status model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:PRESet
```

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the enable status request enable (*SRE) to standard enable (*ESE).

Preset does not affect the event queue.

The Status Event Status Register is not affected by this command.

Example

STAT:PRES	Resets the registers.

Also see

Status model (on page C-1)

:STATus:QUEStionable:CONDition?

This command reads the Questionable Condition Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:QUEStionable:CONDition?
```

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page C-15).

Example

:STAT:QUES:COND?	Reads the Questionable Condition Register.

Also see

None

:STATus:QUEStionable:ENABle

This command sets or reads the contents of the questionable event enable register of the status model.

Type	Affected by	Where saved	Default value
Command and query	STATus:PRESet	Nonvolatile memory	0

Usage

```
:STATus:QUEStionable:ENABle <n>
:STATus:QUEStionable:ENABle?
```

`<n>`	The value of the register

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register is set.

For detail on interpreting the value of a register, see Understanding bit settings (on page C-15).

Example 1

`:STAT:QUES:ENAB 8` `:STAT:QUES:ENAB?`	Enable bit 4, Limit 3 Fail, when the limit test 3 failure value is exceeded. Check to see that the value was set.

Also see

None

:STATus:QUEStionable[:EVENt]?

This command reads the Questionable Event Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:QUEStionable[:EVENt]?
```

Details

This query reads the contents of the questionable status event register. After sending this command and addressing the Model 2450 to talk, a value is sent to the computer. This value indicates which bits in the appropriate register are set.

The Questionable Register can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set the Questionable Register to the sum of their decimal weights. For example, to set bits B12 and B13, set the Questionable Register to 12,288 (which is the sum of 4,096 + 8,192).

Example

`:STAT:QUES?`	Query the Questionable Register.

Also see

Questionable Event Register (on page C-7)

:STATus:QUEStionable:MAP

This command queries mapped event numbers or maps event numbers to bits in the event registers.

Type	Affected by	Where saved	Default value
Command and query			

Usage

```
:STATus:QUEStionable:MAP <bitNumber>, <setEvent>
:STATus:QUEStionable:MAP <bitNumber>, <setEvent>, <clearEvent>
:STATus:QUEStionable:MAP? <bitNumber>
```

`<bitNumber>`	The bit number to check (0 to 14)
`<setEvent>`	The event that, when detected, sets the bit
`<clearEvent>`	The event that, when detected, clears the bit

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`:STAT:QUES:MAP 0, 4917, 4918`	When event 4917 (the buffer is 0% filled) occurs, bit 0 is set in the condition register and the event register of the Questionable Event Register. When event 4918 (buffer is 100% filled) occurs, bit 0 in the condition register is cleared.

Also see

None

SYSTem subsystem

This subsystem contains commands that affect the overall operation of the instrument, such as access, beepers, communications, event logs, and time.

:SYSTem:ACCess

This command contains the type of access users have to the instrument through different interfaces.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	FULL

Usage

```
:SYSTem:ACCess <permissions>
:SYSTem:ACCess?
```

`<permissions>`	The level of access that is allowed: • Full access for all users from all interfaces: `FULL` • Allows access by one remote interface at a time with login and logout required from other interfaces: `EXCLusive` • Allows access by one remote interface at a time with passwords required on all interfaces: `PROTected` • Allows access by one interface at a time (including the front panel) with passwords required on all interfaces: `LOCKout`

Details

When access is set to full, the instrument accepts commands from any interface with no passwords required.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. To use another interface, log out of the present interface before logging into the new interface. You do not need a password with this access.

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

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

Under any access type, if a script is running on one interface when a command comes in from another interface, the command is ignored and the message "Script running from another interface" occurs.

The command `*idn?` is permitted from any interface in all access types.

Example

`:SYST:ACC LOCK`
`login admin`
`logout`
	Set the instrument access to locked out.
	Log into the interface using the default password.
	Log out of the interface.

Also see

:SYSTem:PASSword:NEW (on page 6-105)

:SYSTem:BEEPer[:IMMediate]

This command generates an audible tone.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:BEEPer[:IMMediate] <frequency>, <time>
```

`<frequency>`	The frequency of the beep (20 to 8000)
`<time>`	The amount of time to play the tone in seconds (0.001 to 100)

Details

You can use the beeper of the Model 2450 to provide an audible signal at a specific frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

Example

`:SYSTem:BEEPer 500, 1`	Beep at 500 Hz for one second

Also see

None

:SYSTem:CLEar

This command clears the event log.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Query

```
:SYSTem:CLEar
```

Details

This function removes all messages from the event log.

Also see

:SYSTem:ERRor[:NEXT]? (on page 6-98)

:SYSTem:COMMunication:LAN:CONFigure

This command specifies the LAN configuration for the instrument.

Type	Affected by	Where saved	Default value
Command and query	Rear panel LAN reset	Nonvolatile memory	AUTO

Usage

```
:SYSTem:COMMunication:LAN:CONFigure "AUTO"
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPaddress>,"
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPaddress>,<NETmask>"
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPaddress>,<NETmask>,<GATeway>"
:SYSTem:COMMunication:LAN:CONFigure?
```

AUTO	Use automatically configured LAN settings (default)
MANual	Use manually configured LAN settings
<IPaddress>	LAN IP address; must be a string specifying the IP address in dotted decimal notation; required if the mode is set to manual (default "0.0.0.0")
<NETmask>	The LAN subnet mask; must be a string in dotted decimal notation (default "255.255.255.0")
<GATeway>	The LAN default gateway; must be a string in dotted decimal notation (default "0.0.0.0")

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used. When specifying multiple parameters, do not use spaces after the commas.

The query form of the command returns the present settings in the order shown here:

```
AUTO or MANual,<IPaddress>,<NETmask>,<GATeway>
```

Example 1

```
SYST:COMM:LAN:CONF "MANUAL,192.168.0.1,255.255.240.0,192.168.0.3"
SYST:COMM:LAN:CONF?
```

Set the IP address to be set manually, with the IP address set to 192.168.0.1, the subnet mask to 255.255.240.0, and the gateway address to 192.168.0.3.

Query to verify the settings. The response to the query should be:

```
manual,192.168.0.1,255.255.240.0,192.168.0.3
```

Also see

:SYSTem:COMMunication:LAN:MACaddress? (on page 6-98)

:SYSTem:COMMunication:LAN:MACaddress?

This command queries the LAN MAC address.

Type	Affected by	Where saved	Default value
Query	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:COMMunication:LAN:MACaddress?
```

Details

The MAC address is a character string representing the instrument's MAC address in hexadecimal notation. The string includes colons that separate the address octets.

Example

`SYSTem:COMMunication:LAN:MACaddress?`	Returns the MAC address. For example, you might see: `00:60:1A:00:00:57`

Also see

:SYSTem:COMMunication:LAN:CONFigure (on page 6-97)

:SYSTem:ERRor[:NEXT]?

This command reads the oldest error message from the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor[:NEXT]?
```

Details

As error and status messages occur, they are placed in the event log. The event log is a first-in, first-out (FIFO) register that can hold up to 1000 messages.

This command returns the next entry from the event log.

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

`0,"No error;0,0,0"`

This command returns only error messages from the event log. To return information and warning messages, see `:SYSTem:EVENtlog:NEXT?` Note that if you have used `:SYSTem:ERRor[:NEXT]?` to check events, `:SYSTem:EVENtlog:NEXT?` shows the next event item after the last error that was returned by `:SYSTem:ERRor[:NEXT]?` You will not see warnings or information event log items that occurred before you used `:SYSTem:ERRor[:NEXT]?`

Example

`SYST:ERR:NEXT?`	Returns information on the next error in the event log. For example, if you sent a command without a parameter, the return is: `-109,"Missing parameter;1,1367794863,384492889"`

Also see

:SYSTem:EVENtlog:NEXT? (on page 6-101)

:SYSTem:ERRor:CODE[:NEXT]?

This command reads the oldest error code.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor:CODE[:NEXT]?
```

Details

This command returns the numeric code of the next error in the event log. The error is cleared from the queue after being read.

This command returns only error messages from the event log. To return information and warning messages, see `:SYSTem:EVENtlog:NEXT?`

Example

`SYST:ERR:CODE?`	Returns the error code of the next error in the event log. For example, if error -222, Parameter data out of range error, occurred, the output is: `-222`

Also see

:SYSTem:EVENtlog:NEXT? (on page 6-101)

:SYSTem:ERRor:COUNt?

This command returns the number of errors in the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor:COUNt?
```

Details

This returns the number of errors in the event log. It does not return other types of events, such as information messages. To return other types of events, use `:SYSTem:EVENtlog:COUNt?`

This command does not clear the errors from the event log.

Example

`SYST:ERR:COUN?`	If there are five errors in the event log, the output is: `5`

Also see

:SYSTem:EVENtlog:COUNt? (on page 6-100)

:SYSTem:EVENtlog:COUNt?

This command returns the number of events in the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:COUNt?
:SYSTem:EVENtlog:COUNt? <mask>
:SYSTem:EVENtlog:COUNt? <mask>, <mask>
```

`<mask>`	Limits the list of event log entries to specific types; set to: • `ERRor`: Returns the number of errors • `WARNing`: Returns the number of warnings • `INFormational`: Returns the number of informational messages • `ALL`: Returns all events

Details

You can use the mask parameter to limit the event log items that are counted.

This command does not clear the event log.

If you do not define a mask, all events are counted.

Example

`:SYST:EVEN:COUN? ERR`	Displays the present number of errors in the instrument event log. If there are three errors in the event log, output is: `3`

Also see

:SYSTem:CLEar (on page 6-96)
:SYSTem:EVENtlog:NEXT? (on page 6-101)
:SYSTem:EVENtlog:SAVE (on page 6-103)

:SYSTem:EVENtlog:NEXT?

This command returns the oldest message from the event log and removes it from the log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:NEXT?
:SYSTem:EVENtlog:NEXT? <mask>, <mask>
```

<mask>	Limits the event log entries that are returned to specific types; set to: • ERRor: Returns only the next error • WARNing: Returns only the next warning • INFormational: Returns only the next informational message • ALL: Returns any event

Details

As error and status messages occur, they are placed in the event log. The event log is a first-in, first-out (FIFO) register that can hold up to 1000 messages.

This command returns the next entry from the event log.

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

0,"No error;0,0,0"

Note that if you have used :SYSTem:ERRor[:NEXT]? to check events, :SYSTem:EVENtlog:NEXT? shows the next event item after the last error that was returned by :SYSTem:ERRor[:NEXT]? You will not see warnings or information event log items that occurred before you used :SYSTem:ERRor[:NEXT]?

The information that is returned is in the order:

<eventNumber>, <message>, <eventType>, <timeSeconds>, <timeNanoSeconds>

<eventNumber>	The event number
<message>	A description of the event
<eventType>	The type of event: • Error only: 1 • Warning only: 2 • Information only: 4
<timeSeconds>	The time in seconds
<timeNanoSeconds>	The fractional seconds

Example

SYST:EVEN:NEXT?	Returns information on the next event in the event log. For example, if you sent a command without a parameter, the return is: -109,"Missing parameter;1,1367794863,384492889"

Also see

:SYSTem:CLEar (on page 6-96)
:SYSTem:EVENtlog:SAVE (on page 6-103)

:SYSTem:EVENtlog:POST

This command allows you to post messages to the event log.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:POST <message>
:SYSTem:EVENtlog:POST <message>, <severityType>
```

`<message>`	A string that contains the message that will be associated with this event
`<severityType>`	The type of event that is generated; if nothing is defined, defaults to `INFormational`: • `INFormational` • `ERRor` • `WARNing`

Details

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

You must set the Log Warnings and Log Information options to be reported using the front panel to have the custom warning and information events placed into the event log.

Example

`SYST:EVEN:POST "my error", INF` `SYST:EVEN:NEXT?`	Posts an error named "my error." Output: `1003,"User: my error;4,1400469179,431599191"`

Also see

None

:SYSTem:EVENtlog:SAVE

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

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENtlog:SAVE <filename>
:SYSTem:EVENtlog:SAVE <filename>, <eventType>
:SYSTem:EVENtlog:SAVE <filename>, <eventType>, <eventType>
```

`<filename>`	A string that holds the name of the file to be saved
`<eventType>`	Limits the event log entries that are saved to specific types; set to: • `ERRor`: Saves only error entries • `WARNing`: Saves only warning entries • `INFormational`: Saves only informational messages • `ALL`: Saves all event log entries

Details

This command saves all event log entries since the last clear command to a USB flash drive.

You must insert the USB flash drive before sending this command.

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

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

Example

`SYST:EVEN:SAVE "July_error_log", ERR`	Saves the error events in the event log to a file on the USB flash drive named `July_error_log.csv`.

Also see

:SYSTem:CLEar (on page 6-96)

:SYSTem:GPIB:ADDRess

This command contains the GPIB address.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	18

Usage

```
:SYSTem:GPIB:ADDRess <n>
:SYSTem:GPIB:ADDRess?
```

`<n>`	The GPIB address of the instrument (0 to 30)

Details

The address can be set to any address value between 0 and 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow ample time for the command to be processed before attempting to communicate with the instrument again. After sending this command, make sure to use the new address to communicate with the instrument.

`*RST` does not affect the GPIB address.

Example

`:SYSTem:GPIB:ADDRess 26` `:SYSTem:GPIB:ADDRess?`	Sets the GPIB address and reads the address. Output: `2.600000e+01`

Also see

GPIB setup (on page 2-46)

:SYSTem:LFRequency?

This query returns the line frequency that was automatically detected.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:LFRequency?
```

Details

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.

Example

`:SYST:LFR?`	Check the line frequency.

Also see

None

:SYSTem:PASSword:NEW

This command stores the instrument password.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Nonvolatile memory	admin

Usage

```
:SYSTem:PASSword:NEW "password"
```

`<password>`	A string that contains the instrument password (maximum 30 characters)

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

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.

If you forget the password, you can reset the password to the default. On the front panel, press **MENU**. Under System, select **Manage**. Select **LAN and Password Reset**. You can also reset the password from the rear panel by inserting a straightened paper clip into hole below LAN RESET.

Example

`SYST:PASS:NEW "N3wpa55w0rd"`	Change the password of the instrument to `N3wpa55w0rd`.

Also see

:SYSTem:ACCess (on page 6-95)

:SYSTem:POSetup

This command selects the defaults that are used when you power on the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	RST

Usage

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

`<name>`	Which setup to restore on power up: • Power-up to `*RST` defaults: `RST` • Stored setup 0: `SAV0` • Stored setup 1: `SAV1` • Stored setup 2: `SAV2` • Stored setup 3: `SAV3` • Stored setup 4: `SAV4`

Details

When you select `RST`, the instrument restores settings to their default values when the instrument is powered on.

When you select a `SAV` option, the settings in the selected saved setup are applied when the instrument is powered on. The settings are saved using the `*SAV` command.

Example

`SYST:POS SAV1`	Set the instrument to restore the settings that are saved in the stored setup 1 when the instrument is powered on.

Also see

*SAV (on page 6-2)

:SYSTem:TIME

This command sets the absolute time of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	See **Details**

Usage

```
:SYSTem:TIME <year>, <month>, <day>, <hour>, <minute>, <second>
:SYSTem:TIME <hour>, <minute>, <second>
:SYSTem:TIME?
:SYSTem:TIME? 1
```

`<year>`	Year; must be more than 1970
`<month>`	Month (1 to 12)
`<day>`	Day (1 to 31)
`<hour>`	Hour in 24-hour time format (0 to 23)
`<minute>`	Minute (0 to 60)
`<second>`	Second (0 to 60)

Details

When queried without a parameter, this command returns the present timestamp value in seconds since January 1, 1970 to the nearest second.

If you query with `1`, this command returns the present timestamp in the format:

`<weekday> <month> <day> <hour>:<minute>:<second> <year>`

Where `<weekday>` is the day of the week.

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

If you use `os.time()` but do not specify the time (hour, minute, and second) options, they default to noon for that day. When called without a parameter (the first form), the function returns the current time.

Example

`syst:time 2014, 5, 12, 5, 51, 30`	Set the system time to May 12, 2014 at 05:51:30.

Also see

None

:SYSTem:VERSion?

Query the present SCPI version.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:VERSion?
```

Details

This query command returns the SCPI version.

Example

`SYSTem:VERSion?`	Query the version. An example of a return is: `1996.0`

Also see

None

TRACe subsystem

The TRACe subsystem contains commands that control the reading buffers.

:TRACe:ACTual?

This command contains the number of readings in the specified buffer.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:ACTual?
:TRACe:ACTual? <bufferName>
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command returns the number of readings stored in the buffer.

Example

TRACe:MAKE "testData", 200 COUN 10 MEASure:CURRent? "testData"	Creates 200 element reading buffer named testData. Set the measurement count to 10. Set the measurement function to current. Make readings, and store the readings in testData. Returns the 10th measurement reading after taking all 10 readings.
:TRACe:ACTual?	Returns the number of readings in defbuffer1. Example output: 850
:TRACe:ACTual? "testData"	Returns the number of readings in the buffer testData. Example output: 10

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:CLEar

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

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:CLEar
:TRACe:CLEar <bufferName>
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to defbuffer1

Example

TRACe:MAKE "testData", 200 MEASure:RESistance? "testData" TRACe:ACTual? "testData" TRACe:CLEar "testData" TRACe:ACTual? "testData"	Create user-defined buffer named testData. Take a measurement and store it in testData and returns the last reading measured. Verify that there is data in testData buffer. Output: 1 Clear testData buffer. Verify that testData is empty. Output: 0
TRACe:CLEar TRACe:CLEar "defbuffer1" TRACe:CLEar "defbuffer2"	Clear the default buffer. This command clears defbuffer1. Clear defbuffer1. Specify default buffer by name. Clear defbuffer2. Specify default buffer by name.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:DATA?

This query command returns specified data elements from a specified reading buffer.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:DATA? <startIndex>, <endIndex>
:TRACe:DATA? <startIndex>, <endIndex>, <bufferName>
:TRACe:DATA? <startIndex>, <endIndex>, <bufferName>, <bufferElements>
```

`<startIndex>`	Beginning index of the buffer to return
`<endIndex>`	Ending index of the buffer to return
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
`<bufferElements>`	A list of elements in the buffer to print; if nothing is specified, `READing` is used; see **Details** for a list of options

Details

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
DATE	The date when the data point was measured
FORMatted	The measured value as it appears on the front panel
FRACtional	The fractional seconds for the data point when the data point was measured
READing	The measurement reading based on the SENS:FUNC setting; if no buffer elements are defined, this option is used
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
SOURce	The source value; if readback is ON, then it is the readback value, otherwise it is the programmed source value (see :SOURce[1]:<function>:READ:BACK (on page 6-77))
SOURFORMatted	The source value as it appears on the display
SOURSTATus	The status information associated with sourcing
SOURUNIT	The unit of value associated with the source value
STATus	The status information associated with the measurement
TIME	The time for the data point
TSTamp	The timestamp for the data point
UNIT	The unit of measure associated with the measurement

> **NOTE**
>
> If you have `FORMat[:DATA]` set to `REAL` or `SREAL`, you will have fewer options for buffer elements. If you request one of the buffer elements, you will see the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

Example

`TRAC:MAKE "buf100", 100`	Create a buffer called `buf100` with a maximum size of 100.
`TRIGger:LOAD:LOOP:SIMPle 5, 0, "buf100"`	Set the instrument to configure the trigger model to loop, taking 5 readings with no delay, and store the readings in the `buf100` reading buffer.
`SOUR:VOLT 0.35`	Set the source level for voltage to 0.35.
`INIT`	Initiate the trigger model, which will make 5 readings and store them in `buf100`.

`TRAC:DATA? 1, 5, "buf100", READ, SOUR, REL`	Read the 5 data points, reading, programmed source, and relative time for each point.
	Output:
	`-0.000000,0.350000,0.000000;`
	`-0.000000,0.350000,0.266978,`
	`-0.000000,0.350000,0.443087,`
	`-0.000000,0.350000,0.704459,`
	`-0.000000,0.350000,0.881419`

`TRAC:DATA? 1, 5, "buf100", READ, REL`	Read 5 data points and include the reading and relative time for each data point.
	Output:
	`-0.000000,0.000000,`
	`-0.000000,0.266978,`
	`-0.000000,0.443087,`
	`-0.000000,0.704459,`
	`-0.000000,0.881419`

`TRAC:DATA? 1, 5, "buf100", REL`	Read 5 data points and include relative time for each data point.
	Output:
	`0.000000,0.266978;0.443087,`
	`0.704459,0.881419`

`TRAC:DATA 1,3`	Returns the first 3 reading values from `defbuffer1` reading buffer Output: `-0.000000,-0.000000,-0.000000`

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:DELete

This command deletes a user-defined reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:DELete <readingBuffer>
```

`<readingBuffer>`	A string that contains the name of the user-defined reading buffer to delete

Details

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

Example

`TRAC:DEL "testData"`	Delete the `testData` buffer.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:FILL:MODE

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

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1: CONT defbuffer2: CONT User-defined buffers: ONCE

Usage

```
:TRACe:FILL:MODE <fillType>, <bufferName>
:TRACe:FILL:MODE? <bufferName>
```

`<fillType>`	Fill continuous: `CONTinuous` Fill once: `ONCE`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

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

Use this command to read the fill mode of the specified reading buffer.

Use this command to set the fill mode of the specified reading buffer.

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

When a reading buffer is set to `ONCE`, the first new measurement is stored at `n+1`, where `n` is the number of readings stored in the buffer.

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

You can only change the fill mode of an empty reading buffer. Use `TRACe:CLEar` to empty the buffer.

Example

`TRACe:MAKE "testData", 100`	Create a user-defined reading buffer: `testData`, with the capacity for 100 measurements.
`TRACe:FILL:MODE? "testData"`	Query the fill mode setting for `testData`. Output: `ONCE`
`TRACe:FILL:MODE CONT, "testData"`	Set `testData` fill mode to continuous.
`TRACe:FILL:MODE? "testData"`	Query the fill mode setting for `testData`. Output: `CONT`
`TRACe:FILL:MODE?`	Query the fill mode setting for `defbuffer1`. Output: `CONT`

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:CLEar (on page 6-108)

:TRACe:LOG:STATe

This command indicates whether the reading buffer should log informational events.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1: 1 (ON) defbuffer2: 1 (ON) User-created buffer: 0 (OFF)

Usage

```
TRACe:LOG:STATe <logState>, <bufferName>
TRACe:LOG:STATe? <bufferName>
```

`<logState>`	Disable logging: `OFF` or `0` Enable logging: `ON` or `1`
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command indicates whether the reading buffer should log informational events.

Example

`TRACe:LOG:STATe?`	Query the log state of `defbuffer1`. Output: `1` Indicates that the log state is on.
`TRACe:MAKE "testData", 100`	Create a user-defined buffer named `testData`
`TRACe:LOG:STATe? "testData"`	Query the log state of `testData`. Output: `0` The output indicates that the log state is off.
`TRACe:LOG:STATe 1, "testData"`	Change the `testData` log state to on.
`TRACe:LOG:STATe? "testData"`	Query the log state of `testData`. Output: `1`

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:MAKE

This command creates a user-defined reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Saved setup	Not applicable

Usage

`:TRACe:MAKE <bufferName>, <bufferSize>`

`<bufferName>`	A user-supplied string that indicates the name of a user-defined buffer
`<bufferSize>`	A number that indicates the maximum number of readings that can be stored in `<bufferName>`; minimum is 10

Details

This command creates a user-defined reading buffer.

You cannot assign user-defined reading buffers the same name as an existing buffer, including `defbuffer1` or `defbuffer2`.

Example

`TRACe:MAKE "capTrace", 200`	This example creates a 200-element reading buffer named `capTrace`.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:POINts

This command contains the number of readings a buffer can store.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:POINts <newSize>
:TRACe:POINts <newSize>, <bufferName>
:TRACe:POINts?
:TRACe:POINts? <bufferName>
```

`<newSize>`	The new size for the buffer.
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

Use this command to read the number of measurements that a buffer can store.

Use this command to change the number of readings buffers can store. The number of readings that user-defined buffers can store initially is set when they are created. Default buffers can store 10000 measurements initially.

Example

`TRACe:MAKE "testData", 100`	Create a user-defined reading buffer: `testData`, with the capacity for 100 measurements.
`TRACe:POINts 300, "testData"`	Change the buffer capacity to 300.
`TRACe:POINts? "testData"`	Query the capacity of `testData`. Output: `300`
`TRACe:POINts?`	Query the capacity of the default buffer. Output: `10000`

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:SAVE

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

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:SAVE <fileName>, <bufferName>
:TRACe:SAVE <fileName>, <bufferName>, <timeFormat>
:TRACe:SAVE <fileName>, <bufferName>, <timeFormat>, <start>, <end>
```

`<fileName>`	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
`<timeFormat>`	Defines how date and time information from the buffer is saved in the file on the USB flash drive; the values are: • Dates, times, and fractional seconds are saved; the default value: `FORMat` • Relative time stamps are saved: `RELative` • Seconds and fractional seconds are saved: `RAW` • Time stamps are saved: `STAMp`
`<start>`	Defines the starting point in the buffer to start saving data
`<end>`	Defines the ending point in the buffer to stop saving data

Details

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

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

Example

`TRACe:MAKE "MyBuffer", 100`	Create a buffer called `MyBuffer` with a maximum size of 100.
`SENSe:COUNt 5`	Make five readings for each measurement request and return the data.
`MEASure:CURRent:DC? "MyBuffer"`	Make the measurements.
`TRACe:DATA? 1,5, "MyBuffer", READ, REL, SOUR`	Read the readings, relative timestamp, and source value for each point from 1 to 5. Output: `-0.000000,0.000000, 0.000000,-0.000000, 0.301759,0.000000, -0.000000,0.579068, 0.000000,-0.000000, 0.884302,0.000000, -0.000000,1.157444, 0.000000`
`TRACe:SAVE "/usb1/myData.csv", "MyBuffer"`	Save all reading and default time information from a buffer named `MyBuffer` to a file named `myData.csv` on the USB flash drive.
`TRACe:SAVE "/usb1/myDataRel.csv", "MyBuffer", REL`	Save all readings and relative time stamps from `MyBuffer` to a file named `myDataRel.csv` on the USB flash drive.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:SAVE:APPend (on page 6-118)

:TRACe:SAVE:APPend

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

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:SAVE:APPend <fileName>
:TRACe:SAVE:APPend <fileName>, <bufferName>
:TRACe:SAVE:APPend <fileName>, <bufferName>, <timeFormat>
:TRACe:SAVE:APPend <fileName>, <bufferName>, <timeFormat>, <start>, <end>
```

`<fileName>`	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
`<timeFormat>`	Defines how date and time information from the buffer is saved in the file; the values are: • Dates, times, and fractional seconds are saved; the default value: `FORMat` • Relative time stamps are saved: `RELative` • Seconds and fractional seconds are saved: `RAW` • Time stamps are saved: `STAMp`
`<start>`	Defines the starting point in the buffer to start saving data
`<end>`	Defines the ending point in the buffer to stop saving data

Details

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

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

The file extension `.csv` is appended to the filename if necessary. Any file extension other than `.csv` generates errors.

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

Example

`TRACe:MAKE "testData", 100`	Create a buffer called `testData`.
`SENSe:COUNt 5` `MEASure:CURRent:DC? "testData", READ, REL, SOUR`	Take 5 readings, return for the 5th point: the reading, relative timestamp, and source value. Store the buffer data in the `myData5.csv` file and return the fifth reading.
`TRACe:SAVE "/usb1/myData5.csv", "testData"`	
`TRACe:CLEAr`	Clear `defbuffer1`.
`MEASure:CURRent:DC?`	Take 5 readings, store them in `defbuffer1`, and return the fifth reading.
`TRACe:SAVE:APPend "/usb1/myData5.csv", "defbuffer1"`	Append all the readings stored in `defbuffer1` to the `myData5.csv` file.
`MEASure:CURRent:DC? "testData"`	Take 5 more readings, store them in `testData`, and return the fifth reading.
`TRACe:SAVE:APPend "/usb1/myData5.csv", "testData", RAW, 6, 10`	Append all the readings stored in positions 6 through 10 `testData` to the `myData5.csv` file using raw timestamps.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)

:TRACe:STATistics:AVERage?

This command returns the average of all readings added to the buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:AVERage?
:TRACe:STATistics:AVERage? <bufferName>
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command returns the average reading calculated from all of the readings in the specified reading buffer.

Example

`TRACe:STAT:AVERage?`	Returns the average reading for the readings in the default buffer `defbuffer1`.
`TRACe:STAT:AVERage? "testData"`	Returns the average reading for the readings in the user-defined buffer `testData`.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:STATistics:CLEar (on page 6-120)
:TRACe:STATistics:MAXimum? (on page 6-121)
:TRACe:STATistics:MINimum? (on page 6-121)
:TRACe:STATistics:PK2Pk? (on page 6-122)
:TRACe:STATistics:STDDev? (on page 6-123)

:TRACe:STATistics:CLEar

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

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
TRACe:STATistics:CLEar
TRACe:STATistics:CLEar <bufferName>
```

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

Example

`TRACe:STATistics:CLEar`	Clear all statistics in `defbuffer1`.
`TRACe:STATistics:CLEar "testData"`	Clears all statistics in a user-defined buffer named `testData`.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:STATistics:AVERage? (on page 6-119)
:TRACe:STATistics:MAXimum? (on page 6-121)
:TRACe:STATistics:MINimum? (on page 6-121)
:TRACe:STATistics:PK2Pk? (on page 6-122)
:TRACe:STATistics:STDDev? (on page 6-123)

:TRACe:STATistics:MAXimum?

This command returns the maximum reading value in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:MAXimum?
:TRACe:STATistics:MAXimum? <bufferName>
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Example

`TRACe:STAT:MAXimum?`	Returns the maximum reading value in the default buffer, `defbuffer1`.
`TRACe:STAT:MAXimum? "testData"`	Returns the maximum reading value in the user-defined buffer `testData`.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:STATistics:AVERage? (on page 6-119)
:TRACe:STATistics:CLEar (on page 6-120)
:TRACe:STATistics:MINimum? (on page 6-121)
:TRACe:STATistics:PK2Pk? (on page 6-122)
:TRACe:STATistics:STDDev? (on page 6-123)

:TRACe:STATistics:MINimum?

This command the minimum reading value in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:MINimum?
:TRACe:STATistics:MINimum? <bufferName>
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Example

TRACe:STAT:MINimum?	Returns the minimum reading value in the default buffer defbuffer1.
TRACe:STAT:MINimum? "testData"	Returns the minimum reading value in the user-defined buffer testData.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:STATistics:AVERage? (on page 6-119)
:TRACe:STATistics:CLEar (on page 6-120)
:TRACe:STATistics:MAXimum? (on page 6-121)
:TRACe:STATistics:PK2Pk? (on page 6-122)
:TRACe:STATistics:STDDev? (on page 6-123)

:TRACe:STATistics:PK2Pk?

This command returns the peak-to-peak value of all readings in the buffer.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:PK2Pk?
:TRACe:STATistics:PK2Pk? <bufferName>
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to defbuffer1

Example

TRACe:STAT:PK2Pk?	Returns the peak-to-peak reading value in the default buffer defbuffer1.
TRACe:STAT:PK2Pk? "testData"	Returns the peak-to-peak reading value in the user-defined buffer testData.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:STATistics:AVERage? (on page 6-119)
:TRACe:STATistics:CLEar (on page 6-120)
:TRACe:STATistics:MAXimum? (on page 6-121)
:TRACe:STATistics:MINimum? (on page 6-121)
:TRACe:STATistics:STDDev? (on page 6-123)

:TRACe:STATistics:STDDev?

This command returns the standard deviation of all readings in the buffer.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:STDDev?
:TRACe:STATistics:STDDev? <bufferName>
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Example

`TRACe:STAT:STDDev?`	Returns the standard deviation of the readings in the default buffer `defbuffer1`.
`TRACe:STAT:STDDev? "testData"`	Returns the standard deviation of the readings in the user-defined buffer `testData`.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
:TRACe:MAKE (on page 6-114)
:TRACe:STATistics:CLEar (on page 6-120)
:TRACe:STATistics:MAXimum? (on page 6-121)
:TRACe:STATistics:MINimum? (on page 6-121)
:TRACe:STATistics:PK2Pk? (on page 6-122)

:TRACe:TRIGger

This command makes readings and stores them in a buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:TRIGger
:TRACe:TRIGger <bufferName>
```

`<bufferName>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

Makes readings and stores them in the specified buffer.

Example

`TRACe:MAKE "MyBuffer", 100`
`TRACe:TRIG "MyBuffer"`
`TRACe:TRIG "MyBuffer"`
`TRACe:TRIG "MyBuffer"`
`TRACe:TRIG "MyBuffer"`
`TRACe:TRIG "MyBuffer"`
`TRACe:DATA? 1,5, "MyBuffer", rel`
	Create a buffer called `MyBuffer` with a maximum size of 100.
	Take readings and store them in `MyBuffer`.
	Recall the relative time when the data points were measured for the first five readings in the buffer.
	Example output:
	`0.000000,0.408402,0.816757,1.208823,1.617529`

Also see

:TRACe:MAKE (on page 6-114)

TRIGger subsystem

The commands in this subsystem configure and control the trigger operations, including the trigger model.

:INITiate[:IMMediate]

This command starts the trigger model.

Type	Affected by	Where saved	Default value
Command	Not applicable	Not applicable	Not applicable

Usage

```
:INITiate[:IMMediate]
```

Also see

Trigger model (on page 3-95)

:TRIGger:BLENder<n>:CLEar

This command clears the blender event detector and resets the overrun indicator of blender `<n>`.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLENder<n>:CLEar
```

`<n>`	The blender number (1 or 2)

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

`:TRIG:BLEN2:CLE`	Clears the event detector for blender 2.

Also see

None

:TRIGger:BLENder<n>:MODE

This command selects whether the blender performs OR operations or AND operations.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Trigger blender clear	Save settings	AND

Usage

```
:TRIGger:BLENder<n>:MODE <operation>
:TRIGger:BLENder<n>:MODE?
```

`<n>`	The blender number (1 or 2)
`<operation>`	The type of operation: • `OR` • `AND`

Details

This command selects whether the blender waits for any one event (`OR`) or waits for all selected events (`AND`) before signaling an output event.

Example 1

`:DIG:LINE3:MODE TRIG, IN`
`:DIG:LINE5:MODE TRIG, IN`
`:TRIG:BLEN1:MODE OR`
`:TRIG:BLEN1:STIM1 DIG3`
`:TRIG:BLEN1:STIM2 DIG5`
	Set digital I/O lines 3 and 5 as trigger in lines. Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

:TRIGger:BLENder<n>:STIMulus<m> (on page 6-127)

:TRIGger:BLENder<n>:OVERrun?

This command indicates whether or not an event was ignored because of the event detector state.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLENder<n>:OVERrun?
```

`<n>`	The blender number (1 or 2)

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

`:TRIG:BLEN1:OVER?`	If an event was ignored, the output is `1`. If an event was not ignored, the output is `0`.

Also see

:TRIGger:BLENder<N>:CLEar (on page 6-125)

:TRIGger:BLENder<n>:STIMulus<m>

This command specifies which events trigger the blender.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Trigger blender clear	Save settings	NONE

Usage

```
:TRIGger:BLENder<n>:STIMulus<m> <event>
:TRIGger:BLENder<n>:STIMulus<m>?
```

`<n>`	The blender number (1 or 2)
`<m>`	The stimulus input number (1 to 4)
`<event>`	See **Details**

Details

Trigger events	
Event description	**Event constant**
No trigger event	`NONE`
Front-panel TRIGGER key press	`DISPlay`
Notify trigger block `<n>` (1 to 8) generates a trigger event when the trigger model executes it	`NOTify<n>`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line `<n>` (1 to 6)	`DIGio<n>`
Line edge detected on TSP-Link synchronization line `<n>` (1 to 3)	`TSPLink<n>`
Appropriate LXI trigger packet is received on LAN trigger object `<n>` (1 to 8)	`LAN<n>`
Trigger event blender `<n>` (1 to 2), which combines trigger events	`BLENder<n>`
Trigger timer `<n>` (1 to 4) expired	`TIMer<n>`
Source limit condition occurs	`SLIMit`

Example

`:DIG:LINE3:MODE TRIG, IN`
`:DIG:LINE5:MODE TRIG, IN`
`:TRIG:BLEN1:MODE OR`
`:TRIG:BLEN1:STIM1 DIG3`
`:TRIG:BLEN1:STIM2 DIG5`
	Set digital I/O lines 3 and 5 as trigger in lines. Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

:TRIGger:BLENder<n>:MODE (on page 6-125)

:TRIGger:BLOCk:BRANch:ALWays

This command defines a trigger model block that always goes to a specific block.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:ALWays <blockNumber>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<branchToBlock>`	The block number of the trigger model block to execute when the trigger model reaches this block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by `<branchToBlock>`.

Example

`TRIG:BLOC:BRAN:ALW 9, 20`	When the trigger model reaches block 9, it will always branch to block 20.

Also see

None

:TRIGger:BLOCk:BRANch:COUNter

This command branches to a specified block a specified number of times.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:COUNter <blockNumber>, <targetCount>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<targetCount>`	The number of times to repeat
`<branchToBlock>`	The block number of the trigger model block to execute when the counter is less than to the `<targetCount>` value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value.

Example

`TRIG:LOAD:EMPTY`
`TRIG:BLOC:BUFF:CLEAR 1`
`TRIG:BLOC:MEAS 2`
`TRIG:BLOC:BRAN:COUN 3, 5, 2`
`TRIG:BLOC:DEL:CONS 4, 1`
`TRIG:BLOC:BRAN:COUN 5, 3, 2`
	Reset trigger model settings.
	Clear `defbuffer1` at the beginning of the trigger model.
	Loop and take 5 readings.
	Delay a second.
	Loop three more times back to block 2.
	At end of execution, 15 readings are stored in `defbuffer1`.

Also see

:TRIGger:BLOCk:BRANch:COUNter:COUNt? (on page 6-129)

:TRIGger:BLOCk:BRANch:COUNter:COUNt?

This command returns the count that the trigger model is on.

Type	Affected by	Where saved	Default value
Query only	Recall settings Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:COUNter:COUNt? <blockNumber>
```

`<blockNumber>`	The sequence of the block in the trigger model

Details

The query returns the number of times the trigger model has looped. The counter is defined by `:TRIGger:BLOCk:BRANch:COUNter`.

Example

`TRIG:LOAD:EMPTY`
`TRIG:BLOC:BUFF:CLEAR 1`
`TRIG:BLOC:MEAS 2`
`TRIG:BLOC:BRAN:COUN 3, 5, 2`
`TRIG:BLOC:DEL:CONS 4, 1`
`TRIG:BLOC:BRAN:COUN 5, 3, 2`
`INIT`
`TRIG:BLOCK:BRAN:COUN:COUN? 5`
	Reset trigger model settings.
	Clear `defbuffer1` at the beginning of the trigger model.
	Loop and take 5 readings.
	Delay a second.
	Loop three more times back to block 2.
	At end of execution, 15 readings are stored in `defbuffer1`.
	Check to see which count the trigger model has completed.

Also see

:TRIGger:BLOCk:BRANch:COUNter (on page 6-128)

:TRIGger:BLOCk:BRANch:DELTa

This command defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:DELTa <blockNumber>, <targetDifference>, <branchToBlock>
:TRIGger:BLOCk:BRANch:DELTa <blockNumber>, <targetDifference>, <branchToBlock>,
   <measureBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<targetDifference>`	The value against which the block compares the difference between the measurements
`<branchToBlock>`	The block number of the trigger model block to execute when the difference between the measurements is less than the <targetDifference>
`<measureBlock>`	The block number of the measurement block that makes the measurements to be compared

Details

This block calculates the difference between the last two measurements from a measure block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measurement block, it will compare measurements of a measure block that precedes the branch delta block. For example, if you have a measure block, a wait block, another measure block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure block.

Example

`TRIG:BLOC:BRAN:DELT 5, 0.5, 7, 4`	Configure trigger block 5 to compare the differences between the measurements made in block 4. If the difference between them is less the 0.5, branch to block 7.

Also see

Delta (on page 3-100)

:TRIGger:BLOCk:BRANch:EVENt

This command branches to a specified block when a specified trigger event occurs.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:EVENt <blockNumber>, <event>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<event>`	The event that must occur before the trigger block will act
`<branchToBlock>`	The block number of the trigger model block to execute when the specified event occurs

Details

The branch-on-event building block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

The event can be one of the events shown in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 8) generates a trigger event when the trigger model executes it	NOTify<n>
A command interface trigger occurred: • Any remote interface: *TRG • GPIB only: GET bus command • VXI-11: VXI-11 command device_trigger	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (1 to 2), which combines trigger events	BLENder<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
Source limit condition occurs	SLIMit

Example

```
:TRIG:BLOC:BRAN:EVEN 6, DISP, 2
```

When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the next block in the trigger model).

Also see

On event (on page 3-99)

:TRIGger:BLOCk:BRANch:LIMit:CONStant

This command defines a trigger model block that branches to a block outside the normal trigger model flow if a measurement meets preset criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:LIMit:CONStant <blockNumber>, <limitType>, <limitA>,
   <limitB>, <branchToBlock>, <measureBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<limitType>`	The type of limit (`ABOVe`, `BELow`, `INSide`, or `OUTside`)
`<limitA>`	The limit that the measurement is tested against; if *`limitType`* is set to: • `ABOVe`: This value is ignored • `BELow`: The measurement must be below this value • `INSide`: This is the low limit that the measurement is compared against • `OUTside`: This is the low limit that the measurement is compared against
`<limitB>`	The upper limit that the measurement is tested against; if *`limitType`* is set to: • `ABOVe`: The measurement must be above this value • `BELow`: This value is ignored • `INSide`: This is the high limit that the measurement is compared against • `OUTside`: This is the high limit that the measurement is compared against
`<branchToBlock>`	The block number of the trigger model block to execute when the measurement meets the defined criteria
`<measureBlock>`	The block number of the measurement block that makes the measurement to be compared

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B. Limit A must be set, but is ignored when this type is selected.
- Below: The measurement is below the value set by limit A. Limit B must be set, but is ignored when this type is selected.
- Inside: The measurement is inside the values set by limits A and B. Limit A must be the low value and Limit B must be the high value.
- Outside: The measurement is outside the values set by limits A and B. Limit A must be the low value and Limit B must be the high value.

The measurement block must be a measurement building block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from the measurement building block is used.

Example

`TRIGger:BLOCk:BRANch:LIMit:CONStant 5, OUTside, 0.15, 0.65, 8, 4`	
Configure trigger block 5 to check for measurements in block 4 to be outside of the limits defined by 0.15 and 0.65. If they are, branch to block 8.	

Also see

Constant limits (on page 3-99)

:TRIGger:BLOCk:BRANch:LIMit:DYNamic

This command defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:LIMit:DYNamic <blockNumber>, <limitType>, <limitNumber>,
    <branchToBlock>, <measureBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<limitType>`	The type of limit (`ABOVe`, `BELow`, `INSide`, or `OUTside`)
`<limitNumber>`	The limit number (`1` or `2`)
`<branchToBlock>`	The block number of the trigger model block to execute when the limits are met
`<measureBlock>`	The block number of the measurement block that makes the measurement to be compared

Details

The branch-on-user-limits block goes to a specified building block if a measurement meets the criteria set by this command.

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values. You set these limit threshold values as separate settings. Limit 1 and limit 2 are stored in the measurement configuration list. You can set them to different values in different indices of the measurement configuration list to allow you to step through different values. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

The type of limit can be:

- Above: The measurement is above the value set by the limit low value. The high value is not used when this type is selected.
- Below: The measurement is below the value set by the limit high value. The low value is not used when this type is selected.
- Inside: The measurement is inside the low and high values set for the limit.
- Outside: The measurement is outside the low and high values set for the limit.

The measurement block must be a measurement building block that occurs in the trigger model before the branch-on-constant-limits block.

Example

`CALC2:LIM1:STAT ON`
`CALC2:LIM1:LOW -5.17`
`CALC2:LIM1:UPP -4.23`
`TRIG:BLOC:BRAN:LIM:DYN 9, IN, 1, 12, 7`
	Set the limits on with a low limit of -5.17 and a high limit of -4.23. Set trigger block 9 to test if the limit is inside those limits based on the measurement reading at block 7. If the measurement is within the limits, go to block 12.
	This example is a snippet of a larger example. For the complete code, see Diode grading with immediate binning example (on page 3-83).

Also see

Dynamic limits (on page 3-100)
:CALCulate2:<function>:LIMit<y>:LOWer[:DATA] (on page 6-16)
:CALCulate2:<function>:LIMit<y>:UPPer[:DATA] (on page 6-18)

:TRIGger:BLOCk:BRANch:ONCE

This command causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:ONCE <blockNumber>, <branchToBlock>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<branchToBlock>`	The block number of the trigger model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time the trigger model encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when the trigger model reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model encounters this block.

Example

```
:TRIG:BLOC:BRAN:ONCE 2, 4
```

When the trigger model reaches block 2, the trigger model goes to block 4 instead of proceeding to the default sequence of block 3.

Also see

Once (on page 3-100)

:TRIGger:BLOCk:BRANch:ONCE:EXCLuded

This command causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BRANch:ONCE:EXCLuded <blockNumber>, <branchToBlock>
```

<blockNumber>	The sequence of the block in the trigger model
<branchToBlock>	The block number of the trigger model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded building block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts.

Example

:TRIG:BLOC:BRAN:ONCE:EXCL 2, 4
When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

Once excluded (on page 3-101)

:TRIGger:BLOCk:BUFFer:CLEar

This command defines a trigger model block that clears the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:BUFFer:CLEar <blockNumber>
:TRIGger:BLOCk:BUFFer:CLEar <blockNumber>, <bufferName>
```

`blockNumber`	The sequence of the block in the trigger model
`bufferName`	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used (`defbuffer1`)

Details

When the trigger model reaches the buffer clear trigger block, the instrument empties the buffer that is specified by the command. The specified buffer can be the default buffer or a buffer that you defined. Assigning the name in the buffer clear trigger block does not create a buffer; it only references an existing buffer.

Readings that are made after the buffer is cleared are added to the beginning of the buffer.

You must create the buffer before you define this block.

If no buffer name is assigned, the instrument clears default buffer 1.

Example

`TRIG:LOAD:EMPTy`
`TRIG:BLOC:BUFF:CLE 1`
`TRIG:BLOC:MEAS 2`
`TRIG:BLOC:BRAN:COUN 3, 5, 2`
`TRIG:BLOC:DEL:CONS 4, 1`
`TRIG:BLOC:BRAN:COUN 5, 3, 2`
	Reset trigger model settings.
	Clear `defbuffer1` at the beginning of the trigger model.
	Loop and take 5 readings.
	Delay a second.
	Loop three more times back to block 2.
	At end of execution, 15 readings are stored in `defbuffer1`.

Also see

Reading-buffer clear building block (on page 3-95)
:TRACe:MAKE (on page 6-114)

:TRIGger:BLOCk:CONFig:NEXT

This command recalls the settings at the next index point of a source or measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:CONFig:NEXT <blockNumber>, <configurationList>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<configurationList>`	The source or measure configuration list from which to recall settings

Details

When the trigger model reaches a configuration recall next building block, the settings at the next index point in a configuration list are restored.

Each time this block is encountered, the settings at the next index point in the configuration list are recalled and take effect before the next step executes. When the last index point in the list is reached, it returns to the first point.

Example

`TRIG:BLOC:CONF:NEXT 12, "SOURCE_LIST"`	Set trigger block 12 to restore the settings from the next index point that is stored in the configuration list `SOURCE_LIST`.

Also see

Configuration lists (on page 3-33)

:TRIGger:BLOCk:CONFig:PREVious

This command defines a trigger model block that recalls the settings stored at the previous index point in a measure or source configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

`:TRIGger:BLOCk:CONFig:PREVious <blockNumber>, <configurationList>`

`<blockNumber>`	The sequence of the block in the trigger model
`<configurationList>`	The measure or source configuration list from which to recall settings

Details

The configuration list previous index trigger block type recalls the previous index point in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

Each time the trigger model reaches a configuration list previous block, it goes backward one index point. When the first point in the list is reached, it goes to the last index point in the configuration list.

Example

`TRIG:BLOC:CONF:PREV 14, "SOURCE_LIST"`	Set trigger block 14 to restore the settings from the previous index point that is stored in the configuration list `SOURCE_LIST`.

Also see

Configuration lists (on page 3-33)

:TRIGger:BLOCk:CONFig:RECall

This command recalls the system settings that are stored in a measure or source configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:CONFig:RECall <blockNumber>, <configurationList>
:TRIGger:BLOCk:CONFig:RECall <blockNumber>, <configurationList>, <point>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<configurationList>`	The measure or source configuration list to recall
`<point>`	The point in the configuration list to recall; defaults to 1 if not selected

Details

When the trigger model reaches a configuration recall building block, the settings in the specified configuration list are recalled.

You can restore a specific set of configuration settings in the configuration list by defining the index.

Example

`SOUR:CONF:LIST:CRE "biasLevel"`
`SOUR:FUNC VOLT`
`SENS:FUNC "CURR"`
`SOUR:VOLT:LEV 5`
`SOUR:CONF:LIST:STORE "biasLevel"`
`TRIG:BLOCK:CONF:RECALL 1, "biasLevel", 1`
	Create a configuration list named `biasLevel`. Set the source function to 5 volts and the measure function to current.
	Store the configuration list.
	Recall the configuration list as block 1 of the trigger model.

Also see

Configuration lists (on page 3-33)
:SOURce[1]:CONFiguration:LIST:STORe (on page 6-65)

:TRIGger:BLOCk:DELay:CONStant

This command adds a constant delay to the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:DELay:CONStant <blockNumber>, <time>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<time>`	The amount of time to delay in seconds (50 µs to 10000 s, or zero)

Details

When the trigger model reaches a delay building block, it stops operation for the amount of time set by the delay. This delay is a fixed amount of time. If other delays have been set, this delay is in addition to the other delays.

Example

`SOUR:CONF:LIST:CRE "ampLevel"` `SOUR:CONF:LIST:CRE "biasLevel"`	Create configuration lists named `ampLevel` and `biasLevel`.
`SOUR:FUNC VOLT` `SENS:FUNC "CURR"`	Set the source function to 5 volts and the measurement function to current.
`SOUR:VOLT:LEV 5` `SOUR:CONF:LIST:STORE "ampLevel"`	Store the settings in the `ampLevel` configuration list.
`SOUR:VOLT:LEV 0`	Set the voltage level to 0.
`SOUR:CONF:LIST:STORE "biasLevel"`	Store the setting in the `biasLevel` configuration list.
`TRIG:BLOC:SOUR:STATE 1, ON`	Set block 1 to turn the source output on.
`TRIG:BLOCK:CONF:RECALL 2, "ampLevel", 1`	Set block 2 to recall the `ampLevel` settings.
`TRIG:BLOC:DEL:CONS 3, 0.1`	Set block 3 to provide a constant delay of 0.1 seconds.
`TRIG:BLOCK:MEAS 4`	Set block 4 to make a measurement.
`TRIG:BLOCK:CONF:RECALL 5, "biasLevel", 1`	Set block 5 to recall the `biasLevel` settings.
`TRIG:BLOCK:DEL:CONS 6, 0.2`	Set block 6 to provide a constant delay of 0.2.
`TRIG:BLOCK:BRAN:COUN 7,19,2` `INIT`	Set block 7 to repeat block 2 nineteen times.

Also see

None

:TRIGger:BLOCk:DELay:DYNamic

This command adds a delay to the execution of the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:DELay:DYNamic <blockNumber>, <userDelay>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<userDelay>`	The number of the user delay to recall: • `SOURce<n>`, where `<n>` is the number of the user delay (1 to 5) set by `:SOURce[1]:<function>:DELay:USER<n>` • `MEASure<n>`, where `<n>` is the number of the user delay (1 to 5) set by `[:SENSe[1]]:<function>:DELay:USER<n>`

Details

When the trigger model reaches a delay building block, it stops the trigger model for the amount of time set by the delay.

The delay time is set by the user delay command.

Example

`:SOUR:VOLT:DEL:USER1 5`	Set user delay for source 1 to 5 seconds.
`:TRIG:BLOC:SOUR:STAT 1, ON`	Set trigger block 1 to turn the source output on.
`:TRIG:BLOC:DEL:DYN 2, SOUR1`	Set trigger block 2 to a dynamic delay that calls source user delay 1.
`:TRIG:BLOC:MEAS 3`	Set trigger block 3 to make a measurement.
`:TRIG:BLOC:SOUR:STAT 4, OFF`	Set trigger block 4 to turn the source output off.
`:TRIG:BLOC:BRAN:COUN 5, 10, 1`	Set trigger block 5 to branch to block 1 ten times.
`:INIT`	Start the trigger model.

Also see

[:SENSe[1]]:<function>:DELay:USER<n> (on page 6-47)
:SOURce[1]:<function>:DELay:USER<n> (on page 6-68)

:TRIGger:BLOCk:DIGital:IO

This command defines a trigger model block that sets the lines on the digital I/O port high or low.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:DIGital:IO <blockNumber>, <bitPattern>
:TRIGger:BLOCk:DIGital:IO <blockNumber>, <bitPattern>, <bitMask>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<bitPattern>`	Sets the value that specifies the bit pattern
`<bitMask>`	Specifies the bit mask; if omitted, all lines are driven

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern that is specified as an integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The optional bit mask defines the bits in the pattern that are driven high or low. If the bit for a line is set to 1, the line is driven high. If the bit is set to 0, the line is driven low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63) or omit this parameter.

For this command to function as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

`:DIGital:LINE3:MODE DIG,OUT`
`:DIGital:LINE4:MODE DIG,OUT`
`:DIGital:LINE5:MODE DIG,OUT`
`:DIGital:LINE6:MODE DIG,OUT`
`:TRIG:BLOC:DIG:IO 4, 20, 60`
	The first four lines of code configures digital I./O lines 3 through 6 as digital outputs.
	Trigger block `4` is then configured with a bit pattern of `20` (digital I/O lines 3 and 5 high). The optional bit mask is specified as `60` (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

:DIGital:LINE<n>:MODE (on page 6-19)
Digital I/O bit weighting (on page 3-67)
Digital I/O port (on page 3-62)

:TRIGger:BLOCk:LIST?

This command returns the settings for all trigger model building blocks.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLOCk:LIST?
```

Details

This returns the settings for the trigger model.

Example

`:TRIG:BLOC:LIST?`	Returns the settings for the trigger model. A typical output is:
	`1) SOURCE_OUTPUT          OUTPUT: ON`
	`2) CONFIG_RECALL          CONFIG_LIST: ampLevel  INDEX: 1`
	`3) DELAY_CONSTANT         DELAY: 0.100000`
	`4) MEASURE                BUFFER: defbuffer1`
	`5) CONFIG_RECALL          CONFIG_LIST: biasLevel  INDEX: 1`
	`6) DELAY_CONSTANT         DELAY: 0.200000`
	`7) BRANCH_COUNTER         VALUE: 19  BRANCH_BLOCK: 2`

Also see

None

:TRIGger:BLOCk:LOG:EVENt

This command logs an event during trigger model execution.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
TRIGger:BLOCk:LOG:EVENt <blockNumber>, <eventNumber>, <message>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<eventNumber>`	The event number: • `INFO<n>` • `WARNing<n>` • `ERRor<n>` Where `<n>` is 1 to 4; you can define up to four of each type
`<message>`	A string up to 32 characters

Details

This block allows you to log an event in the event log when the trigger model is running. Insert the block into the trigger model. When the trigger model executes the block, the event is logged.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

Also see

None

:TRIGger:BLOCk:MEASure

This command defines a trigger block that makes a measurement.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
TRIGger:BLOCk:MEASure <blockNumber>
TRIGger:BLOCk:MEASure <blockNumber>, <bufferName>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<bufferName>`	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used

Details

When the trigger model reaches the measurement block:

1. The instrument makes a reading.
2. The trigger model waits for the measurement to complete.
3. The instrument places the measurement into the specified reading buffer. If no buffer is specified, the reading is placed into the default buffer (`defbuffer1`).

If you are defining a specific reading buffer, you must create it before you define this block.

Example

`TRIG:LOAD:EMPTY`
`TRIG:BLOC:BUFF:CLEAR 1`
`TRIG:BLOC:MEAS 2`
`TRIG:BLOC:BRAN:COUN 3, 5, 2`
`TRIG:BLOC:DEL:CONS 4, 1`
`TRIG:BLOC:BRAN:COUN 5, 3, 2`
	Reset trigger model settings.
	Clear `defbuffer1` at the beginning of the trigger model.
	Loop and take 5 readings.
	Delay a second.
	Loop three more times back to block 2.
	At end of execution, 15 readings are stored in `defbuffer1`.

Also see

Measure building block (on page 3-96)
:TRACe:MAKE (on page 6-114)

:TRIGger:BLOCk:NOP

This command creates a placeholder that performs no action in the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

`TRIGger:BLOCk:NOP <blockNumber>`

`<blockNumber>`	The sequence of the block in the trigger model

Details

If you remove a trigger model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

`TRIG:BLOC:NOP 5`	Set block number 5 to be a no operation block.

Also see

None

:TRIGger:BLOCk:NOTify

This command defines a trigger model block that generates a trigger event and immediately continues to the next block.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:NOTify <blockNumber>, <notifyID>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<notifyID>`	The identification number of the notification; 1 to 8

Details

When the trigger model reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

You can define up to eight notify blocks in a trigger model. You can reference the event that the notify block generates by other commands to assign a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

When you call this event, you use the format `NOTIFY` followed by the notify ID. For example, if you assign `<notifyID>` as `4`, you would refer to it as `NOTIFY4` in the command that references it.

Example

`:TRIG:BLOC:NOT 5, 2` `:TRIG:BLOC:BRAN:EVEN 6, NOTIFY2, 2`	Define trigger model block 5 to be the notify 2 event. Assign the notify 2 event to be the trigger for stimulus for the branch event for block 6.

Also see

Notify building block (on page 3-97)

:TRIGger:BLOCk:SOURce:STATe

This command defines a trigger block that turns the output source on or off.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:SOURce:STATe <blockNumber>, <state>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<state>`	Enable the source: `ON` Disable the source: `OFF`

Details

The source building block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as `SOURCE_OUTPUT`.

Example

`TRIG:BLOC:SOUR:STAT 1, 1`
`TRIG:BLOC:DEL:CONS 2, 0.01`
`TRIG:BLOC:MEAS 3`
`TRIG:BLOC:BRAN:COUN 4, 100, 2`
`TRIG:BLOC:SOUR:STAT 5, 0`
	This example turns the output on.
	Delay 10 ms.
	Make a measurement.
	Loop and take 100 readings.
	The output is turned off after the 100 readings are made.

Also see

Wait building block (on page 3-96)

:TRIGger:BLOCk:WAIT

This commands defines a trigger model block that waits for an event before allowing the trigger model to continue.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCk:WAIT <blockNumber>, <event>
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <logic>, <event>
:TRIGger:BLOCk:WAIT <blockNumber>, <event>, <logic>, <event>, <event>
```

`<blockNumber>`	The sequence of the block in the trigger model
`<event>`	The event that must occur before the trigger block will act
`<logic>`	If each event must occur before the trigger model continues: `AND` If at least one of the events must occur before the trigger model continues: `OR`

Details

You can use the wait block to synchronize measurements with other instruments and devices.

Events that you can set the instrument to wait for include:

- Digital input/output signals, such as DB-9 and TSP-Link
- LAN events
- Blenders

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. When the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again.

The instrument clears the memory of the recorded event when the trigger model is at the start block and when the trigger model exits the wait block.

You can have up to eight wait blocks in a trigger model.

All items in the list are subject to the same action — you cannot combine `AND` and `OR` logic in a single command.

The events can be one of the events shown in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	`NONE`
Front-panel TRIGGER key press	`DISPlay`
Notify trigger block `<n>` (1 to 8) generates a trigger event when the trigger model executes it	`NOTify<n>`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line `<n>` (1 to 6)	`DIGio<n>`
Line edge detected on TSP-Link synchronization line `<n>` (1 to 3)	`TSPLink<n>`
Appropriate LXI trigger packet is received on LAN trigger object `<n>` (1 to 8)	`LAN<n>`

Trigger events	
Event description	**Event constant**
No trigger event	`NONE`
Trigger event blender `<n>` (1 to 2), which combines trigger events	`BLENder<n>`
Trigger timer `<n>` (1 to 4) expired	`TIMer<n>`
Source limit condition occurs	`SLIMit`

Example

`:TRIGger:BLOCk:WAIT 9, DISP`	Set trigger model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing.

Also see

Wait building block (on page 3-96)

:TRIGger:DIGital<n>:IN:CLEar

This command clears the trigger event on a digital input line.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:DIGital<n>:IN:CLEar
```

`<n>`	Digital I/O trigger line (1 to 6)

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to `false`).

For this command to function as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

`:TRIG:DIG2:IN:CLE`	Clears the trigger event detector on I/O line 2.

Also see

:DIGital:LINE<n>:MODE (on page 6-19)
Digital I/O port (on page 3-62)
:TRIGger:DIGital<n>:IN:OVERrun? (on page 6-151)

:TRIGger:DIGital<n>:IN:EDGE

This command sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	FALL

Usage

```
:TRIGger:DIGital<n>:IN:EDGE <triggerLogic>
:TRIGger:DIGital<n>:IN:EDGE?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<triggerLogic>`	The trigger logic value: `FALLing`, `RISing`, or `EITHer` See **Details** for descriptions of values

Details

Set `<triggerLogic>` to one of the following values:

Trigger mode values

Value	Description
`FALLing`	Detects falling-edge triggers as input when the line is configured as an input or open drain.
`RISing`	Detects rising-edge triggers as input when the line is configured as an open drain.
`EITHer`	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain.

When the line is programmed to be used as a trigger line (see the mode command), the output state of the I/O line is controlled through the trigger logic specified by this command.

To directly control the line state, set the mode of the line to digital and use the write command. When in digital mode with the line configured for open drain, the edge setting asserts a TTL low-pulse for output. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse for output.

Example

`:DIG:LINE4:MODE TRIG,IN` `:TRIG:DIG4:IN:EDGE RIS`	Sets the input trigger mode for I/O line 4 to detect rising-edge triggers as input.

Also see

:DIGital:LINE<n>:MODE (on page 6-19)
:DIGital:WRITe <n> (on page 6-21)
:TRIGger:DIGital<n>:IN:CLEar (on page 6-149)
Digital I/O port (on page 3-62)

:TRIGger:DIGital<n>:IN:OVERrun?

This command returns the event detector overrun status.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

`:TRIGger:DIGital<n>:IN:OVERrun?`

`<n>`	Digital I/O trigger line (1 to 6)

Details

This command returns the event detector overrun status as `0` (false) or `1` (true).

If this is 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

`TRIG:DIG1:IN:OVER?`	Returns `0` if no overruns have occurred or `1` if an overrun has occurred.

Also see

:DIGital:LINE<n>:MODE (on page 6-19)
Digital I/O port (on page 3-62)

:TRIGger:DIGital<n>:OUT:LOGic

This command sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	POS

Usage

```
:TRIGger:DIGital<n>:OUT:LOGic <logicType>
:TRIGger:DIGital<n>:OUT:LOGic?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<logicType>`	The output trigger logic of the trigger event generator: `POSitive` or `NEGative`

Details

This command configures the trigger event generator to assert a TTL pulse for output logic; positive is a high pulse, negative is a low pulse.

Example

`:DIG:LINE4:MODE TRIG, OUT` `:TRIG:DIG4:OUT:LOG NEG`	Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

:DIGital:LINE<n>:MODE (on page 6-19)
Digital I/O port (on page 3-62)

:TRIGger:DIGital<n>:OUT:PULSewidth

This command describes the length of time that the trigger line is asserted for output triggers.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	10e-6 (10 μs)

Usage

```
:TRIGger:DIGital<n>:OUT:PULSewidth <width>
:TRIGger:DIGital<n>:OUT:PULSewidth?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<width>`	Pulse length (0 to 100,000 s)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely.

Example

`DIG:LINE1:MODE TRIG, OUT` `TRIG:DIG1:OUT:PULS 200`	Set digital line 1 to trigger out. Set the pulse to 200.

Also see

:DIGital:LINE<n>:MODE (on page 6-19)
:DIGital:WRITe <n> (on page 6-21)
Digital I/O port (on page 3-62)

:TRIGger:DIGital<n>:OUT:STIMulus

This command selects the event that causes a trigger to be asserted on the digital output line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:DIGital<n>:OUT:STIMulus <event>
:TRIGger:DIGital<n>:OUT:STIMulus?
```

`<n>`	Digital I/O trigger line (1 to 6)
`<event>`	The event to use as a stimulus; see **Details**

Details

The digital trigger pulsewidth command determines how long the trigger is asserted. The trigger stimulus for a digital I/O line may be set to one of the existing trigger events, which are described in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	`NONE`
Front-panel TRIGGER key press	`DISPlay`
Notify trigger block `<n>` (1 to 8) generates a trigger event when the trigger model executes it	`NOTify<n>`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line `<n>` (1 to 6)	`DIGio<n>`
Line edge detected on TSP-Link synchronization line `<n>` (1 to 3)	`TSPLink<n>`
Appropriate LXI trigger packet is received on LAN trigger object `<n>` (1 to 8)	`LAN<n>`
Trigger event blender `<n>` (1 to 2), which combines trigger events	`BLENder<n>`
Trigger timer `<n>` (1 to 4) expired	`TIMer<n>`
Source limit condition occurs	`SLIMit`

Example

`:TRIG:DIG2:OUT:STIMulus TIM3`	Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.

Also see

:DIGital:LINE<n>:STATe (on page 6-20)
:TRIGger:DIGital<n>:OUT:LOGic (on page 6-151)
Digital I/O port (on page 3-62)

:TRIGger:LAN<n>:IN:CLEar

This command clears the event detector for a trigger.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
TRIGger:LAN<n>:IN:CLEar
```

`<n>`	The LAN event number (1 to 8)

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the previous history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

`TRIG:LAN5:IN:CLE`	Clears the event detector with LAN packet 5.

Also see

None

:TRIGger:LAN<n>:IN:EDGE

This command sets the trigger operation and detection mode of the specified LAN event.

Type	Affected by	Where saved	Default value
Command and query	LAN trigger <n> reset Recall settings	Save settings Nonvolatile memory	EITH

Usage

```
:TRIGger:LAN<n>:IN:EDGE <mode>
:TRIGger:LAN<n>:IN:EDGE?
```

`<n>`	The LAN event number (1 to 8)
`<mode>`	The trigger mode; see the **Details** for more information

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values		
Mode	**Trigger packets detected as input**	**LAN trigger packet generated for output with a…**
`EITHer`	Rising or falling edge (positive or negative state)	negative state
`FALLing`	Falling edge (negative state)	negative state
`RISing`	Rising edge (positive state)	positive state

Example

`:TRIG:LAN2:IN:EDGE FALL`	Set the LAN trigger mode for event 2 to falling.

Also see

Digital I/O (on page 3-62)
TSP-Link System Expansion Interface (on page 3-118)

:TRIGger:LAN<n>:IN:OVERrun?

This command indicates the overrun status of the event detector.

Type	Affected by	Where saved	Default value
Query only	TRIGger:LAN<n>:IN:CLEar	Not applicable	Not applicable

Usage

```
:TRIGger:LAN<n>:IN:OVERrun?
```

`<n>`	The LAN event number (1 to 8)

Details

This attribute indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

The trigger overrun state for the specified LAN packet is returned as `1` (true) or `0` (false).

Example

`TRIG:LAN5:IN:OVER?`	Checks the overrun status of a trigger on LAN5 and outputs the value, such as: `0`

Also see

:TRIGger:LAN<n>:IN:CLEar (on page 6-154)

:TRIGger:LAN<n>:OUT:CONNect:STATe

This command prepares the event generator for outgoing trigger events.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LAN<n>:OUT:CONNect:STATe <b>
:TRIGger:LAN<n>:OUT:CONNect:STATe?
```

`<n>`	The LAN event number (1 to 8)
`<b>`	Do not send event messages: `OFF` or `0` Prepare to send event messages: `ON` or `1`

Details

When this is set to on, the instrument prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event are changed.

When this is set to `OFF`, for TCP connections, this closes the TCP connection.

Example

`:TRIGger:LAN1:OUT:PROTocol MULT` `:TRIGger:LAN1:OUT:CONNect:STATe ON`	Set the protocol to multicast and prepare the event generator to send event messages.

Also see

:TRIGger:LAN<n>:OUT:IP:ADDRess (on page 6-158)
:TRIGger:LAN<n>:OUT:PROTocol (on page 6-159)

:TRIGger:LAN<n>:OUT:IP:ADDRess

This command specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	"0.0.0.0"

Usage

```
:TRIGger:LAN<n>:OUT:IP:ADDRess <address>
:TRIGger:LAN<n>:OUT:IP:ADDRess?
```

`<n>`	The LAN event number (1 to 8)
`<address>`	A string that represents the LAN address in dotted decimal notation

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

`TRIG:LAN1:OUT:IP:ADDR "192.0.32.10"`	Use IP address `192.0.32.10` to connect the LAN trigger.

Also see

:TRIGger:LAN<n>:OUT:CONNect:STATe (on page 6-157)

:TRIGger:LAN<n>:OUT:LOGic

This command sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	NEG

Usage

```
:TRIGger:LAN<n>:OUT:LOGic <logicType>
:TRIGger:LAN<n>:OUT:LOGic?
```

`<n>`	The LAN event number (1 to 8)
`<logicType>`	`POSitive` `NEGative`

Example

`TRIG:LAN1:OUT:LOG POS`	Set the logic to positive.

Also see

None

:TRIGger:LAN<n>:OUT:PROTocol

This command sets the LAN protocol to use for sending trigger messages.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	TCP

Usage

```
:TRIGger:LAN<n>:OUT:PROTocol
:TRIGger:LAN<n>:OUT:PROTocol?
```

`<n>`	The LAN event number (1 to 8)
`<protocol>`	The protocol to use for messages from the trigger: • `TCP` • `UDP` • `MULTicast`

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

`:TRIG:LAN1:OUT:PROT TCP` `:TRIG:LAN1:OUT:CONN:STAT`	Set the LAN protocol for trigger messages to be TCP and re-connect the LAN trigger event generator.

Also see

:TRIGger:LAN<n>:OUT:CONNect:STATe (on page 6-157)
:TRIGger:LAN<n>:OUT:IP:ADDRess (on page 6-158)

:TRIGger:LAN<n>:OUT:STIMulus

This command specifies events that cause this trigger to assert.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:LAN<n>:OUT:STIMulus <eventID>
:TRIGger:LAN<n>:OUT:STIMulus?
```

`<n>`	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)
`<eventID>`	The LAN event identifier used to trigger the event

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored and the action overrun is set.

Trigger events	
Event description	**Event constant**
No trigger event	`NONE`
Front-panel TRIGGER key press	`DISPlay`
Notify trigger block `<n>` (1 to 8) generates a trigger event when the trigger model executes it	`NOTify<n>`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line `<n>` (1 to 6)	`DIGio<n>`
Line edge detected on TSP-Link synchronization line `<n>` (1 to 3)	`TSPLink<n>`
Appropriate LXI trigger packet is received on LAN trigger object `<n>` (1 to 8)	`LAN<n>`
Trigger event blender `<n>` (1 to 2), which combines trigger events	`BLENder<n>`
Trigger timer `<n>` (1 to 4) expired	`TIMer<n>`
Source limit condition occurs	`SLIMit`

Example

`TRIG:LAN1:OUT:STIM TIM1`	Set the timer 1 trigger event as the source for the LAN packet 1 trigger stimulus.

Also see

:TRIGger:LAN<n>:OUT:CONNect:STATe (on page 6-157)

:TRIGger:LOAD:CONFiguration:LIST

This command loads a predefined trigger model configuration that uses source and measure configuration lists.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LOAD:CONFiguration:LIST <measureConfigList>, <sourceConfigList>
:TRIGger:LOAD:CONFiguration:LIST <measureConfigList>, <sourceConfigList>, <delay>
:TRIGger:LOAD:CONFiguration:LIST <measureConfigList>, <sourceConfigList>, <delay>,
   <readingBuffer>
```

`<measureConfigList>`	The configuration list that contains the measurement settings to be used for this trigger model
`<sourceConfigList>`	The configuration list that contains the source settings to be used for this trigger model
`<delay>`	The delay time before the measurement (seconds); default is 0
`<readingBuffer>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This trigger model template incorporates a source configuration list and measure configuration list.

You must set up the configuration lists before loading the trigger model.

You can also set a delay and reading buffer.

Example

`TRIG:LOAD:CONF:LIST "MyMeasList", "src config"`
Load the configuration list trigger model, using the measurement configuration list `MyMeasList` and the source configuration list `src config`.

Also see

None

:TRIGger:LOAD:EMPTy

This command resets the trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LOAD:EMPTy
```

Details

When you load this predefined trigger model, any existing trigger model settings are reset.

Any existing trigger blocks are deleted when you execute this command.

Example

`TRIG:LOAD:EMPTY`
`TRIG:BLOC:BUFF:CLEAR 1`
`TRIG:BLOC:MEAS 2`
`TRIG:BLOC:BRAN:COUN 3, 5, 2`
`TRIG:BLOC:DEL:CONS 4, 1`
`TRIG:BLOC:BRAN:COUN 5, 3, 2`
	Reset trigger model settings.
	Clear `defbuffer1` at the beginning of execution of the trigger model.
	Loop and take 5 readings.
	Delay a second.
	Loop three more times back to block 2.
	At the end of execution, 15 readings are stored in `defbuffer1`.

Also see

None

:TRIGger:LOAD:LOOP:DURation

This command loads a predefined trigger model configuration that makes continuous measurements for a specified amount of time

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LOAD:LOOP:DURation <duration>
:TRIGger:LOAD:LOOP:DURation <duration>, <delay>
:TRIGger:LOAD:LOOP:DURation <duration>, <delay>, <reading buffer>
```

`<duration>`	The amount of time for which to take measurements (0 to 100,000 seconds)
`<delay>`	The delay time before the measurement (50 µs to 10000 seconds); default is 0 for no delay
`<reading buffer>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

When you load this predefined trigger model, you can specify amount of time to make a measurement and the length of the delay before the measurement.

Example

`*RST`
`SOUR:FUNC VOLT`
`SOUR:VOLT 5`
`SENS:FUNC "CURR"`
	Reset the instrument. Set the instrument to source voltage at 5 V. Set to measure current.

`TRIG:LOAD:LOOP:DUR 10, 0.01`	Load the duration loop trigger model to take measurements for 10 seconds with a 10 ms delay before each measurement.
`INIT`	Start the trigger model.

Also see

None

:TRIGger:LOAD:LOOP:SIMPle

This command loads a predefined trigger model configuration.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LOAD:LOOP:SIMPle <count>
:TRIGger:LOAD:LOOP:SIMPle <count>, <delay>
:TRIGger:LOAD:LOOP:SIMPle <count>, <delay>, <reading buffer>
```

`<count>`	The number of measurements to make
`<delay>`	The time before the measurement (50 µs to 10000 seconds); default is 0 for no delay
`<readingBuffer>`	A string that indicates the reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you defined in the count parameter.

Example

`*RST`
`SENS:FUNC "CURR"`
`SENS:CURR:RANG:AUTO ON`
	Reset the instrument and set it to measure current with automatic range setting.

`SOUR:FUNC VOLT`
`SOUR:DEL 0.1`
`SOUR:VOLT 5`
	Source 5 volts with a source delay of 0.1 s.

`SOUR:VOLT:ILIM 0.01`	Set a current limit of 0.01 A.
`TRIG:LOAD:LOOP:SIMP 10`	Set a simple trigger loop with a count of 10.
`OUTP ON`	Turn the output on.
`INIT`	Start the trigger model.
`*WAI`	Postpone execution of subsequent commands until all previous commands are finished.
`TRAC:DATA? 1, 10, "defbuffer1", SOUR, READ, REL`	Read data and store the source, reading, and relative time.

Also see

None

:TRIGger:LOAD:TRIGger:EXTernal

This command loads a predefined trigger model configuration.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LOAD:TRIGger:EXTernal <digInLine>, <digOutLine>, <count>
:TRIGger:LOAD:TRIGger:EXTernal <digInLine>, <digOutLine>, <count>, <delay>
:TRIGger:LOAD:TRIGger:EXTernal <digInLine>, <digOutLine>, <count>, <delay>,
   <readingBuffer>
```

`<digInLine>`	The digital input line (1 to 6); also the event that the trigger model will wait on in block 1
`<digOutLine>`	The digital output line (1 to 6)
`<count>`	Number of times to wait for the trigger and send out the trigger
`<delay>`	The time in seconds before the measurement is made; default is 0
`<readingBuffer>`	The reading buffer that will store the measurement data, default is `defbuffer1`

Details

This trigger model waits for a digital I/O event to occur, makes a measurement, and issues a notify event.

Also see

None

:TRIGger:STATe?

This command returns the present state of the trigger model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:STATe?
```

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model is presently executing.

The trigger model states are:

- Idle: The trigger model is stopped
- Running: The trigger model is running
- Waiting: The trigger model has been in the same wait block for more than 100 ms
- Empty: The trigger model is selected, but no blocks are defined
- Building: Blocks have been added.
- Failed: The trigger model is stopped because of an error.
- Aborting: The trigger model is stopping because of a user request.
- Aborted: The trigger model is stopped because of a user request.

Example

`:TRIG:STAT?`	An example output if the trigger model is inactive and ended at block 9 would be: `IDLE;IDLE;9`

Also see

None

:TRIGger:TIMer<n>:CLEar

This command clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:TIMer<n>:CLEar
```

`<n>`	Trigger timer number (1 to 4)

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

`:TRIG:TIM1:CLEar`	Clears trigger timer 1.

Also see

:TRIGger:TIMer<n>:COUNt (on page 6-167)
:TRIGger:TIMer<n>:STARt:OVERrun? (on page 6-170)

:TRIGger:TIMer<n>:COUNt

This command sets the number of events to generate each time the timer detects its trigger event or is enabled as a timer or alarm.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1

Usage

```
TRIGger:TIMer<n>:COUNt <count>
TRIGger:TIMer<n>:COUNt?
```

`<n>`	A trigger timer number (1 to 4)
`<count>`	The number of times to repeat the trigger

Details

If *count* is set to a number greater than 1, the timer automatically starts the next delay at the expiration of the previous delay.

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

This command should not be used with the trigger model.

Example

`TRIG:TIM2:COUN 4`	Set the number of events to generate for trigger timer 2 to four.

Also see

None

:TRIGger:TIMer<n>:DELay

These commands set and read the timer delay.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	10e-6 (10 µs)

Usage

```
TRIGger:TIMer<n>:DELay <interval>
TRIGger:TIMer<n>:DELay?
```

`<n>`	Trigger timer number (1 to 4)
`<interval>`	Delay interval in seconds

Details

Each time the timer is triggered, it uses this delay period.

Reading this command returns the delay interval that will be used the next time the timer is triggered.

Example

`TRIG:TIM2:DEL 50E-6`	Set trigger timer 2 to delay for 50 µs.

Also see

None

:TRIGger:TIMer<n>:STARt:FRACtional

This command configures an alarm or a time in the future when the timer will start.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:TRIGger:TIMer<n>:STARt:FRACtional <time>
:TRIGger:TIMer<n>:STARt:FRACtional?
```

`<n>`	The trigger timer number (1 to 4)
`<time>`	The time in seconds (0 to 2147483647)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time in the past or if it is in the future.

Example

`TRIG:TIM1:STAR:SEC 60`
`TRIG:TIM1:START:FRAC 0.5`
`TRIG:TIM2:STAT ON`
	Set the timer for 60.5 seconds.
	Enable the trigger timer for timer 3.

Also see

:TRIGger:TIMer<n>:STARt:SEConds (on page 6-171)
:TRIGger:TIMer<n>:STATe (on page 6-173)

:TRIGger:TIMer<n>:STARt:GENerate

This command specifies when timer events are generated.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (OFF)

Usage

```
TRIGger:TIMer<n>:STARt:GENerate <b>
TRIGger:TIMer<n>:STARt:GENerate?
```

`<n>`	Trigger timer number (1 to 4)
`<b>`	Generate a timer event when the timer delay elapses: `OFF` or `0` Generate a timer event when the timer starts and when the delay elapses: `ON` or `1`

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event `TIMERN`.

Example

`TRIG:TIM3:STAR:GEN ON`	Set trigger timer 3 to generate an event when the timer starts and when the timer delay elapses.

Also see

None

:TRIGger:TIMer<n>:STARt:OVERrun?

This command indicates if an event was ignored because of the event detector state.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:TIMer<n>:STARt:OVERrun?
```

`<n>`	Trigger timer number (1 to 4)

Details

This attribute indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

`TRIG:TIM1:STAR:OVER?`	Checks the overrun status on trigger timer 1.

Also see

None

:TRIGger:TIMer<n>:STARt:SEConds

This command configures an alarm or a time in the future when the timer will start.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:TRIGger:TIMer<n>:STARt:SEConds <time>
:TRIGger:TIMer<n>:STARt:SEConds?
```

`<n>`	The trigger timer number (1 to 4)
`<time>`	The time in seconds (0 to 2147483647)

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time in the past or if it is in the future.

Example

`TRIG:TIM1:STAR:SEC 60`
`TRIG:TIM1:START:FRAC 0.5`
`TRIG:TIM2:STAT ON`
	Set the timer for 60.5 seconds.
	Enable the trigger timer for timer 3.

Also see

:TRIGger:TIMer<n>:STATe (on page 6-173)

:TRIGger:TIMer<n>:STARt:STIMulus

This command describes the event that starts the trigger timer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:TIMer<n>:STARt:STIMulus <event>
:TRIGger:TIMer<n>:STARt:STIMulus?
```

`<n>`	Trigger timer number (1 to 4)
`<event>`	The event that will start the trigger timer

Details

Set this attribute any trigger event to start the timer when that event occurs.

Set this attribute to zero (0) to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

Trigger events	
Event description	**Event constant**
No trigger event	`NONE`
Front-panel TRIGGER key press	`DISPlay`
Notify trigger block `<n>` (1 to 8) generates a trigger event when the trigger model executes it	`NOTify<n>`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`COMMand`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line `<n>` (1 to 6)	`DIGio<n>`
Line edge detected on TSP-Link synchronization line `<n>` (1 to 3)	`TSPLink<n>`
Appropriate LXI trigger packet is received on LAN trigger object `<n>` (1 to 8)	`LAN<n>`
Trigger event blender `<n>` (1 to 2), which combines trigger events	`BLENder<n>`
Trigger timer `<n>` (1 to 4) expired	`TIMer<n>`
Source limit condition occurs	`SLIMit`

Example

`*RST`
`DIG:LINE1:MODE TRIG,IN`
`DIG:LINE2:MODE TRIG,OUT`
`TRIG:TIM1:DEL 35e-3`
`TRIG:TIM1:STAR:STIM  DIG1`
`TRIG:DIG2:OUT:STIM TIM1`
	Reset the instrument to default settings. Set digital I/O line 1 for use as a trigger input.
	Set digital I/O line 2 for use as a trigger output.
	Set timer 1 to delay 35 ms.
	Set timer 1 to start delaying once the digital I/O 1 event is detected.
	Set digital I/O line 2 to output a pulse once the timer 1 event is detected.

Also see

None

:TRIGger:TIMer<n>:STATe

This command enables the trigger timer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (OFF)

Usage

```
:TRIGger:TIMer<n>:STATe <b>
:TRIGger:TIMer<n>:STATe?
```

`<n>`	Trigger timer number (1 to 4)
`<b>`	Disable the trigger timer (`OFF` or `0`) Enable the trigger timer (`ON` or `1`)

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

Example 1

`DIG:LINE3:MODE TRIG,OUT`
`TRIG:DIG3:OUT:STIM TIM2`
`SYSTem:TIME?`
`TRIG:TIM2:START:SECONDS <current time> + 60`
`TRIG:TIM2:STAT ON`
	To configure timer 2 for an alarm to fire 1 minute from now and output a pulse on digital I/O line 3, query to get the current time. Add 60 seconds to that value and use that to configure the start seconds. Enable the timer.

Example 2

`*RST`
`DIG:LINE5:MODE TRIG,OUT`
`TRIG:DIG5:OUT:STIM TIM3`
`TRIG:TIM3:DEL 3e-3`
`TRIG:TIM3:COUNT 5`
`TRIG:TIM3:STAT ON`
	Configure timer 3 to generate 5 pulses on digital I/O line 5 that are 3 ms apart.

Example 3

`*RST`
`DIG:LINE3:MODE TRIG,IN`
`DIG:LINE5:MODE TRIG,OUT`
`TRIG:DIG5:OUT:STIM TIM3`
`TRIG:TIM3:DEL 3e-3`
`TRIG:TIM3:COUNT 5`
`TRIG:TIM3:START:STIM DIG3`
`TRIG:TIM3:STAT ON`
	Configure timer 3 to generate 5 pulses on digital I/O line 5 that are 3 ms apart when a digital input is detected on digital line 3.

Also see

None

Section 7

Introduction to TSP operation

In this section:

Introduction to TSP operation

Instruments that are Test Script Processor (TSP®) enabled operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you would when using any other instrument.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without an external controller. You can load a series of TSP commands into the instrument using a remote computer or the front-panel USB drive . You can store these commands as a script that can be run later by sending a single command message to the instrument.

You do not have to choose between using conventional control or script control. You can combine these forms of instrument control in the way that works best for your test application.

Controlling the instrument by sending individual command messages

The simplest method of controlling an instrument through the communication interface is to send it a message that contains remote commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

TSP commands can be function-based or attribute-based. Function-based commands are commands that control actions or activities. Attribute-based commands define characteristics of an instrument feature or operation.

Constants are commands that represent fixed values.

Functions

Function-based commands control actions or activities. A function-based command performs an immediate action on the instrument.

Each function consists of a function name followed by a set of parentheses ( ). You should only include information in the parentheses if the function takes a parameter. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Example 1

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

Example 2

You can use the results of a function-based command directly or assign variables to the results for later access. The following code defines `x` and prints it.

`x = math.abs(-100)` `print(x)`	Output: `100`

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, some characteristics of TSP-enabled instruments are the model number (`localnode.model`) and the brightness of the front-panel display (`display.lightstate`).

Attributes can be read-only, read-write, or write-only. They can be used as a parameter of a function or assigned to another variable.

To set the characteristics, attribute-based commands define a value. For many attributes, the value is in the form of a number, enumerated type, or a predefined constant.

Example 1: Set an attribute using a number

`testData = buffer.make(500)` `testData.capacity = 600`	Use a function to create a buffer named `testData` with a capacity of 500, then use the *`bufferVar`*`.capacity` attribute to change the capacity to 1000.

Example 2: Set an attribute using an enumerated type

`display.lightstate = display.STATE_LCD_75`	This attribute controls the brightness of the front-panel display and buttons. Setting this attribute to `display.STATE_LCD_75` sets the brightness of the display and buttons to 75% of full brightness.

Example 3: Set an attribute using a constant

`format.data = format.REAL64`	Using the constant `REAL64` sets the print format to double precision floating point format.

Reading an attribute

To read an attribute, you can use the attribute as the parameter of a function or assign it to another variable.

Example 3: Read an attribute using a function

`print(display.lightstate)`	Reads the status of the light state by passing the attribute to the print function. If the display light state is set to 50%, the output is: `display.STATE_LCD_50`

Example 4: Read an attribute using a variable

`light = display.lightstate` `print(light)`	This reads the light state by assigning the attribute to a variable named `light`. If the display light state is set to 25%, the output is: `display.STATE_LCD_25`

Queries

Test Script Processor (TSP®) enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` and `printnumber()` commands generate output in the form of response messages. Each `print()` command creates one response message.

Example

`x = 10` `print(x)`	Example of an output response message: `1.00000e+01` Note that your output may be different if you set your ASCII precision setting to a different value.

USB flash drive path

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

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

```
"/usb1/"
```

Information on scripting and programming

If you need information about using scripts with your TSP-enabled instrument, see Fundamentals of scripting for TSP (on page 7-4).

If you need information about using the Lua programming language with the instrument, see Fundamentals of programming for TSP (on page 7-11).

Fundamentals of scripting for TSP

NOTE

Though it can improve your process to use scripts, you do not have to create scripts to use the instrument. Most of the examples in the documentation can be run by sending individual command messages. The next few sections of the documentation describe scripting and programming features of the instrument. You only need to review this information if you are using scripting and programming.

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument more efficiently.

Scripts offer several advantages compared to sending individual commands from the host controller (computer):

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs more quickly and efficiently when it processes scripts than it does when it processes individual commands.
- You can incorporate features such as looping and branching into scripts.
- Scripts allow the controller to perform other tasks while the instrument is running a script, enabling some parallel operation.
- Scripts eliminate repeated data transfer times from the controller.

In the instrument, the Test Script Processor (TSP®) scripting engine processes and runs scripts.

This section describes how to create, load, modify, and run scripts.

What is a script?

A script is a collection of instrument control commands and programming statements. Scripts that you create are referred to as **user scripts**.

Your scripts can be interactive. Interactive scripts display messages on the front panel of the instrument that prompt the operator to enter parameters.

Run-time and nonvolatile memory storage of scripts

Scripts are loaded into the run-time environment of the instrument. From there, they can be stored in the nonvolatile memory.

The run-time environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to store a value while the instrument is turned on. When you create a script, the instrument creates a global variable with the same name so that you can reference the script more conveniently. After scripts are loaded into the run-time environment, you can run and manage them from the front panel of the instrument or from a computer. Information in the run-time environment is lost when the instrument is turned off.

Nonvolatile memory is where information is stored even when the instrument is turned off. Save scripts to nonvolatile memory to save them even if the power is cycled. The scripts that are in nonvolatile memory are loaded into the run-time environment when the instrument is turned on.

Scripts are placed in the run-time environment at the following times:

- When they are run.
- When they are loaded over a remote command interface.
- When the instrument is turned on (if they are stored in nonvolatile memory).

For detail on the amount of available memory, see Memory considerations for the run-time environment (on page 7-38).

NOTE

If you make changes to a script in the run-time environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See Saving a script to nonvolatile memory (on page 7-8).

What can be included in scripts?

Scripts can include combinations of 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 8-7)). Lua is a scripting language that is described in Fundamentals of programming for TSP (on page 7-11).

Working with scripts

This section describes the basics of working with scripts.

You can create and manage scripts from the front panel or over a remote interface. Scripts can be saved in the instrument, on a computer, or on a USB flash drive.

Tools for managing scripts

You can use any of the following tools to manage scripts:

- The front-panel menu options and flash drive. For information, see Saving setups (on page 2-112).
- Messages sent to the instrument. For information, see Load a script by sending commands over a remote interface (on page 7-7).
- Keithley Instruments Test Script Builder (TSB) software (included on the Test Script Builder Software Suite CD-ROM that was included with your instrument). For more information, see Creating a new TSP project (on page 7-33).
- Your own development tool or program.

Script rules

Scripts must have a unique name. The name must not contain spaces.

You can have as many scripts as needed in the instrument. The only limitation is the amount of memory available to the run-time environment.

When a script is loaded into the run-time environment, a global variable with the same name as the script is created to reference the script.

Important points regarding scripts:

- If you load a new script with the same name as an existing script, an error message is generated. You must delete the existing script before you create a new script with the same name.
- If you revise a script and save it to the instrument with a new name, the previously loaded script remains in the instrument with the original name.
- Script names cannot have spaces.
- You can save scripts to nonvolatile memory in the instrument. Saving a script to nonvolatile memory allows the instrument to be turned off without losing the script. See Saving a script to nonvolatile memory (on page 7-8).

Loading a script into the instrument

You can load scripts using the through the front-panel display by copying them from a USB flash drive or over a remote interface using `loadscript` commands.

Loading a script using a USB flash drive

After loading a script onto a USB flash drive, you can copy the script using options on the front-panel display.

To load a script using a USB flash drive:

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

Load a script by sending commands over a remote interface

To load a script over the remote interface, you can use the `loadscript` and `endscript` commands.

The `loadscript` command starts the collection of messages that make up the script. When the instrument receives this command, it starts collecting all subsequent messages instead of running them immediately.

The `endscript` command tells the instrument to stop collecting messages. It then compiles the collection of messages into a script. This script is loaded into the run-time environment.

To load a script:

Send the `loadscript` command with a script name. This tells the instrument to start collecting messages for the script named `testInfo`:

```
loadscript testInfo
```

Send the script; this example displays text on the user display when the script is run:

```
display.settext(display.TEXT1, "Batch 233")
display.settext(display.TEXT2, "Test Information")
```

Send the command that tells the instrument that the script is complete:

```
endscript
```

Run the script by sending the script name followed by `()`:

```
testInfo()
```

The User Display swipe screen on the front panel shows the text "Batch 233 Test Information" when you run this script.

To save the script to nonvolatile memory, send the command:

```
testInfo.save()
```

In summary, to load a script by sending commands:

1. Send the command `loadscript` *`scriptName`*, where *`scriptName`* is the name of the script. The name must be a legal Lua variable name.
2. Send the commands that need to be included in the script.
3. Send the command `endscript`.
4. You can now run the script. Send the script name followed by `()`. For more information, see Running scripts using a remote interface (on page 7-8).

Running scripts using the front-panel interface

To run a script from the front-panel interface:

1. Press the **MENU** key.
2. Under Scripts, select **Run**. The RUN SCRIPTS window is displayed.
3. From the Available Scripts list, select the script you want to run.
4. Select **Run Selected**.

Running scripts using a remote interface

You can run any script using *scriptVar*.run(). Replace s*criptVar* with the name of a script that is in nonvolatile or run-time memory.

Saving a script to nonvolatile memory

You can save scripts to nonvolatile memory. To keep a script through a power cycle, you must save the script to nonvolatile memory.

To save a script to nonvolatile memory:

1. Create and load a script.
2. Send the command *scriptVar*.save(), where *scriptVar* is the name of the script.

Example: Save a user script to nonvolatile memory

test1.save()	Assume a script named test1 has been loaded. test1 is saved into nonvolatile memory.

Saving a script to a USB drive

You can save scripts to a USB flash drive.

To save a script to an external USB drive:

1. Load a script.
2. Insert a flash drive into the USB port on the front panel.
3. Send the command *scriptVar*.save("/usb1/*filename*.tsp"), where *scriptVar* is the variable referencing the script and *filename* is the name of the file.

Rename a script

To rename a script in the runtime environment:

1. Load the script into the runtime environment with a different name.
2. Delete the previous version of the script.

To rename a script in nonvolatile memory:

Send the commands:

```
scriptVar = script.load(file)
scriptVar.save()
```

Where:

scriptVar is the name of variable that references the script

file is the path and file name of the script file to load

For example, to load a script named test8 from the flash drive and save it to nonvolatile memory, send the commands:

```
test8 = script.load("/usb1/test8.tsp")
test8.save()
```

NOTE

If the new name is the same as a name that is already used for a script, an error message is displayed.

Retrieve a user script from the instrument

You can review user scripts that are in the nonvolatile memory of the instrument and retrieve them.

To get a list of the scripts that are in the instrument, send:

```
for name in script.user.catalog() do
   print(name)
end
```

To retrieve a script, use `scriptVar.source`, where *scriptVar* is the name of the script you want to retrieve. For example, to retrieve a script named `contactTest`, you would send:

```
print(contactTest.source)
```

The command is returned as a single string. The `loadscript` and `endscript` keywords are not included.

Deleting a user script using a remote interface

Deleting a user script deletes the script from the instrument.

To delete a script from the instrument:

Send the command:

```
script.delete("name")
```

Where: *name* is the user-defined name of the script.

Example: Delete a user script

`script.delete("test8")`	Delete a user script named `test8` from the instrument.

Autoexec script

The autoexec script runs automatically when the instrument is turned on. To create an autoexec script, save a new script that is named `autoexec`. The `autoexec` script is automatically saved to nonvolatile memory. See Saving a script to nonvolatile memory (on page 7-8).

To set up the autoexec script from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. Select **Copy to Power Up**. A dialog box opens that explains that the script has been copied to the autoexec script and will run every time the instrument is restarted.
4. Select **OK**.

To save the autoexec script, send the command:

```
autoexec.save()
```

Commands that cannot be used in scripts

Though an instrument accepts the following commands, you cannot use these commands in scripts.

General commands that cannot be used in scripts:

- `abort`
- `endflash`
- `endscript`
- `flash`
- `loadscript`
- `loadandrunscript`
- `login`
- `logout`
- `prevflash`

Common commands that cannot be used in scripts are shown in the following table with equivalent commands that can be used.

Unavailable commands with TSP equivalents

Common commands	TSP equivalent commands
`*CLS`	`eventlog.clear() status.clear()`
`*ESE`	`status.standard.enable`
`*ESE?`	`print(status.standard.enable)`
`*ESR?`	`print(status.standard.event)`

`*IDN?`	`print(localnode.model)`
	`print(localnode.serialno)`
	`print(localnode.version)`

`*LANG`	No equivalent
`*LANG?`	No equivalent
`*OPC`	`opc()`
`*OPC?`	`waitcomplete() print([[1]])`
`*RST`	`reset()`
`*SRE`	`status.request_enable`
`*SRE?`	`print(status.request_enable)`
`*STB?`	`print(status.condition)`
`*TRG`	No equivalent
`*TST?`	`print([[0]])`
`*WAI`	`waitcomplete()`

Fundamentals of programming for TSP

To conduct a test, a computer (controller) is programmed to send sequences of commands to an instrument. The controller orchestrates the actions of the instrumentation. The controller is typically programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

To take advantage of the advanced features of the instrument, you can add programming commands to your scripts. Programming commands control script execution and provide tools such as variables, functions, branching, and loop control.

The Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands.

What is Lua?

Lua is a programming language that can be used with TSP-enabled instruments. Lua is an efficient language with simple syntax that is easy to learn.

Lua is also a scripting language, which means that scripts are compiled and run when they are sent to the instrument. You do not compile them before sending them to the instrument.

Lua basics

This section contains the basics about the Lua programming language to allow you to start adding Lua programming commands to your scripts quickly.

Comments

Comments start anywhere outside a string with a double hyphen (--). If the text immediately after a double hyphen (--) is anything other than double left brackets (`[[`), the comment is a short comment, which continues only until the end of the line. If double left brackets (`[[`) follow the double hyphen (--), it is a long comment, which continues until the corresponding double right brackets (`]]`) close the comment. Long comments may continue for several lines and may contain nested `[[ ... ]]` pairs. The table below shows how to use code comments.

Using code comments

Type of comment	Comment delimiters	Usage	Example
Short comment	`--`	Use when the comment text fits on a single line.	`--Turn off the front-panel display.` `display.lightstate = display.STATE_LCD_OFF`

Long comment	`--[[ ]]`	Use when the comment text is longer than one line.	`--[[Display a menu with three menu items.`
			`If the second menu item is selected,`
			`the selection will be given the value`
			`Test2.]]`

Function and variable name restrictions

You cannot use Lua reserved words and top level command names for function or variable names.

You cannot use the following Lua reserved words for function or variable names. If you attempt to assign these, the error message -285, "TSP Syntax error at line x: unexpected symbol near '*word*' " is displayed, where *word* is the Lua reserved word.

Lua reserved words		
and	for	or
break	function	repeat
do	if	return
else	in	then
elseif	local	true
end	nil	until
false	not	while

Values and variable types

In Lua, you use variables to store values in the run-time environment for later use.

Lua is a dynamically-typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been assigned a value.

Variable types

Variables can be one of the following types.

Variable types and values

Variable type returned	Value	Notes
`"nil"`	not declared	The type of the value `nil`, whose main property is to be different from any other value; usually it represents the absence of a useful value.
`"boolean"`	true or false	Boolean is the type of the values `false` and `true`. In Lua, both `nil` and `false` make a condition `false`; any other value makes it `true`.
`"number"`	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers. Hexadecimal and binary values are also handled as the number type in TSP.
`"string"`	sequence of words or characters	
`"function"`	a block of code	Functions perform a task or compute and return values.
`"table"`	an array	New tables are created with { } braces. For example, {1, 2, 3.00e0}.

To determine the type of a variable, you can call the `type()` function, as shown in the examples below.

NOTE

The output you get from these examples may vary depending on the data format that is set.

Example: Nil

```
x = nil
print(x, type(x))
```

```
nil      nil
```

Example: Boolean

```
y = false
print(y, type(y))
```

```
false      boolean
```

Example: Hex constant

You can enter hexadecimal values, but to return a hexadecimal value, you must create a function, as shown in this example. Note that hexadecimal values are handled as a number type.

```
hex = function (i) return "0x"..string.format("%X", i) end
print(hex(0x54|0x55))
print(hex(0x54&0x66))
```

Set the format to return hexadecimal values, then `OR` two hexadecimal values and `AND` two hexadecimal values.
Output:

```
0x55
0x44
```

Example: Binary constant

Binary values are returned as floating point decimal values. Note that binary values are handled as a number type.

`x = 0b0000000011111111`
`y = 0B1111111100000000`
`print(x, type(x))`
`print(y, type(y))`
	`255 number`
	`65280 number`

Example: String and number

`x = "123"`
`print(x, type(x))`
 	
`x = x + 7`
`print(x, type(x))`
	`123      string`
	
	Adding a number to x forces its type to number.
	`1.30     number`

Example: Function

`function add_two(first_value,`
`   second_value)`
`   return first_value + second_value`
`end`
`print(add_two(3, 4), type(add_two))`
	`7     function`

Example: Table

`atable = {1, 2, 3, 4}`
`print(atable, type(atable))`
`print(atable[1])`
`print(atable[4])`
	Defines a table with four numeric elements.
	Note that the "table" value (shown here as `a096cd30`) will vary.
	
	`table: a096cd30      table`
	`1`
	`4`

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the run-time environment.

Operators

You can compare and manipulate Lua variables and constants using operators.

Arithmetic operators

Operator	Description
+	addition
–	subtraction
*	multiplication
/	division
-	negation (for example, c = –a)
^	exponentiation

Relational operators

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~= !=	not equal
==	equal

Bitwise operators

Operator	Description
&	AND
|	OR
^^	exclusive OR
<<	bitwise shift left
>>	bitwise shift right
!	logical NOT

Logical and bitwise operators

The logical operators in Lua are `and`, `or`, and `not`. All logical operators consider both `false` and `nil` as false and anything else as true.

The operator `not` always returns `false` or `true`.

The conjunction operator `and` returns its first argument if the first argument is `false` or `nil`; otherwise, `and` returns its second argument. The disjunction operator `or` returns its first argument if this value is different from `nil` and `false`; otherwise, `or` returns its second argument. Both `and` and `or` use shortcut evaluation, that is, the second operand is evaluated only if necessary.

NOTE

The example output you get may vary depending on the data format settings of the instrument.

Example 1

`print(10 or eventlog.next())`	`10`
`print(nil or "a")`	`a`
`print(nil and 10)`	`nil`
`print(false and eventlog.next())`	`false`
`print(false and nil)`	`false`
`print(false or nil)`	`nil`
`print(10 and 20)`	`20`

Example 2

```
hex = function (i) return "0x"..string.format("%X", i) end
print(hex(0x54 | 0x55))
print(hex(0x54 & 0x66))
```

Set the format to return hexadecimal values, then `OR` two hexadecimal values and `AND` two hexadecimal values.

Output:

```
0x55
0x44
```

Example 3

```
hex = function (i) return "0x"..string.format("%X", i) end
a, b= 0b01010100, 0b01100110
print(hex(a), "&", hex(b), "=", hex(a & b))
```

Set the format to return hexadecimal values, define binary values for `a` and `b`, then `AND` `a` and `b`.

Output:

```
0x54 & 0x66 = 0x44
```

String concatenation

String operators

Operator	Description
`..`	Concatenates two strings. If either argument is a number, it is coerced to a string (in a reasonable format) before concatenation.

Example: Concatenation

`print(2 .. 3)`	`23`
`print("Hello " .. "World")`	`Hello World`

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority):

- `^` (exponentiation)
- `not`, `-` (unary), `!` (logical NOT)
- `*`, `/`, `<<`, `>>`
- `+`, `-`, `&`, `|`, `^^`
- `..` (concatenation)
- `<`, `>`, `<=`, `>=`, `~=`, `!=`, `==`
- `and`
- `or`

You can use parentheses to change the precedences in an expression. The concatenation ("`..`") and exponentiation ("`^`") operators are right associative. All other binary operators are left associative. The examples below show equivalent expressions.

Equivalent expressions

`reading + offset < testValue/2+0.5`	=	`(reading + offset) < ((testValue/2)+0.5)`
`3+reading^2*4`	=	`3+((reading^2)*4)`
`Rdg < maxRdg and lastRdg <= expectedRdg`	=	`(Rdg < maxRdg) and (lastRdg <= expectedRdg)`
`-reading^2`	=	`-(reading^2)`
`reading^testAdjustment^2`	=	`reading^(testAdjustment^2)`

Functions

With Lua, you can group commands and statements using the `function` keyword. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple values.

You can use functions to form expressions that calculate and return a value. Functions can also act as statements that execute specific tasks.

Functions are first-class values in Lua. That means that functions can be stored in variables, passed as arguments to other functions, and returned as results. They can also be stored in tables.

Note that when a function is defined, it is stored in the run-time environment. Like all data that is stored in the run-time environment, the function persists until it is removed from the run-time environment, is overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or in Lua code in either of the following forms:

```
function myFunction(parameterX) functionBody end
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function.
- *parameterX*: Parameter names. To use multiple parameters, separate the names with commas.
- *functionBody* is the code that is executed when the function is called.

To execute a function, substitute appropriate values for *parameterX* and insert them into a message formatted as:

```
myFunction(valueForParameterX, valueForParameterY)
```

Where *valueForParameterX* and *valueForParameterY* represent the values to be passed to the function call for the given parameters.

NOTE

The output you get from these examples will vary depending on the data format settings of the instrument.

Example 1

`function add_two(first_value,`
`   second_value)`
`   return first_value + second_value`
`end`
`print(add_two(3, 4))`
	Creates a variable named `add_two` that has a variable type of function.
	Output:
	`7`

Example 2

`add_three = function(first_value,`
`   second_value, third_value)`
`   return first_value + second_value +`
`      third_value`
`end`
`print(add_three(3, 4, 5))`
	Creates a variable named `add_three` that has a variable type of function.
	Output:
	`12`

Example 3

`function sum_diff_ratio(first_value,`
`   second_value)`
`   psum = first_value + second_value`
`   pdif = first_value - second_value`
`   prat = first_value / second_value`
`   return psum, pdif, prat`
`end`
`sum, diff, ratio = sum_diff_ratio(2, 3)`
`print(sum)`
`print(diff)`
`print(ratio)`
	Returns multiple parameters (sum, difference, and ratio of the two numbers passed to it).
	Output:
	`5`
	`-1`
	`0.66666666666667`

Create functions using scripts

You can use scripts to define functions. Scripts that define a function are like any other script: They do not cause any action to be performed on the instrument until they are executed. The global variable of the function does not exist until the script that created the function is executed.

A script can consist of one or more functions. Once a script has been run, the computer can call functions that are in the script directly.

For detail on creating functions, see Fundamentals of scripting for TSP (on page 7-4).

Conditional branching

Lua uses the `if`, `else`, `elseif`, `then`, and `end` keywords to do conditional branching.

Note that in Lua, `nil` and `false` are `false` and everything else is `true`. Zero (0) is `true` in Lua.

The syntax of a conditional block is as follows:

```
if expression then
   block
elseif expression then
   block
else
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

Example: If

`if 0 then`
`   print("Zero is true!")`
`else`
`   print("Zero is false.")`
`end`
	Output:
	`Zero is true!`

Example: Comparison

`x = 1`
`y = 2`
`if x and y then`
`   print("Both x and y are true")`
`end`
	Output:
	`Both x and y are true`

Example: If and else

```
x = 2
if not x then
   print("This is from the if block")
else
   print("This is from the else block")
end
```

Output:

```
This is from the else
block
```

Example: Else and elseif

```
x = 1
y = 2
if x and y then
   print("'if' expression 2 was not false.")
end

if x or y then
   print("'if' expression 3 was not false.")
end

if not x then
   print("'if' expression 4 was not false.")
else
   print("'if' expression 4 was false.")
end

if x == 10 then
   print("x = 10")
elseif y > 2 then
   print("y > 2")
else
   print("x is not equal to 10, and y is not greater than 2.")
end
```

Output:

```
'if' expression 2 was not false.
'if' expression 3 was not false.
'if' expression 4 was false.
x is not equal to 10, and y is not greater than 2.
```

Loop control

If you need to repeat code execution, you can use the Lua `while`, `repeat`, and `for` control structures. To exit a loop, you can use the `break` keyword.

While loops

To use conditional expressions to determine whether to execute or end a loop, you use `while` loops. These loops are similar to Conditional branching (on page 7-19) statements.

```
while expression do
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: While

```
list = {
   "One", "Two", "Three", "Four", "Five", "Six"}
print("Count list elements on numeric index:")
element = 1
while list[element] do
   print(element, list[element])
   element = element + 1
end
```

This loop exits when `list[element] = nil`.

Output:

```
Count list elements on
   numeric index:
1   One
2   Two
3   Three
4   Four
5   Five
6   Six
```

Repeat until loops

To repeat a command, you use the `repeat` ... `until` statement. The body of a `repeat` statement always executes at least once. It stops repeating when the conditions of the `until` clause are met.

```
repeat
   block
until expression
```

Where:

- *`block`* consists of one or more Lua statements
- *`expression`* is Lua code that evaluates to either `true` or `false`

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Repeat until

```
list = {
   "One", "Two", "Three", "Four", "Five", "Six"}
print("Count elements in list using repeat:")
element = 1
repeat
   print(element, list[element])
   element = element + 1
until not list[element]
```

Output:

```
Count elements in list
   using repeat:
1  One
2  Two
3  Three
4  Four
5  Five
6  Six
```

For loops

There are two variations of `for` statements supported in Lua: numeric and generic.

NOTE

In a `for` loop, the loop expressions are evaluated once, before the loop starts.

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Numeric for

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
---------- For loop -----------
print("Counting from one to three:")
for element = 1, 3 do
   print(element, list[element])
end
print("Counting from one to four, in steps of two:")
for element = 1, 4, 2 do
   print(element, list[element])
end
```

The numeric `for` loop repeats a block of code while a control variable runs through an arithmetic progression.

Output:

```
Counting from one to three:
1   One
2   Two
3   Three
Counting from one to four, in steps of two:
1   One
3   Three
```

Example: Generic for

```
days = {"Sunday",
   "Monday",     "Tuesday",
   "Wednesday",  "Thursday",
   "Friday",     "Saturday"}

for i, v in ipairs(days) do
   print(days[i], i, v)
end
```

The generic `for` statement works by using functions called iterators. On each iteration, the iterator function is called to produce a new value, stopping when this new value is nil.

Output:

```
Sunday      1    Sunday
Monday      2    Monday
Tuesday     3    Tuesday
Wednesday   4    Wednesday
Thursday    5    Thursday
Friday      6    Friday
Saturday    7    Saturday
```

Break

The `break` statement can be used to terminate the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A break ends the innermost enclosing loop.

Return and break statements can only be written as the last statement of a block. If it is necessary to return or break in the middle of a block, an explicit inner block can be used.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Break with while statement

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a `break` value (`breakValue`) so that the `break` statement is used to exit the `while` loop before the value of k reaches `0`.

Output:

```
Going to break and k = 3
```

Example: Break with while statement enclosed by comment delimiters

`local numTable = {5, 4, 3, 2, 1}`
`local k = table.getn(numTable)`
`--local breakValue = 3`
`while k > 0 do`
`   if numTable[k] == breakValue then`
`      print("Going to break and k = ", k)`
`      break`
`   end`
`   k = k - 1`
`end`
`if k == 0 then`
`   print("Break value not found")`
`end`
	This example defines a break value (`breakValue`), but the break value line is preceded by comment delimiters so that the break value is not assigned, and the code reaches the value `0` to exit the `while` loop.
	Output:
	`Break value not found`

Example: Break with infinite loop

`a, b = 0, 1`
`while true do`
`   print(a, b)`
`   a, b = b, a + b`
`   if a > 500 then`
`      break`
`   end`
`end`
	This example uses a `break` statement that causes the while loop to exit if the value of `a` becomes greater than 500.
	Output:
	`0     1`
	`1     1`
	`1     2`
	`2     3`
	`3     5`
	`5     8`
	`8     13`
	`13    21`
	`21    34`
	`34    55`
	`55    89`
	`89    144`
	`144   233`
	`233   377`
	`377   610`

Tables and arrays

Lua makes extensive use of the data type table, which is a flexible array-like data type. Table indices start with 1. Tables can be indexed not only with numbers, but with any value except `nil`. Tables can be heterogeneous, which means that they can contain values of all types except `nil`.

Tables are the sole data structuring mechanism in Lua. They may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, and so on. To represent records, Lua uses the field `name` as an index. The language supports this representation by providing `a.name` as an easier way to express `a["name"]`.

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Loop array

`atable = {1, 2, 3, 4}`
`i = 1`
`while atable[i] do`
`   print(atable[i])`
`   i = i + 1`
`end`
	Defines a table with four numeric elements.
	Loops through the array and prints each element.
	The Boolean value of `atable[index]` evaluates to `true` if there is an element at that index. If there is no element at that index, `nil` is returned (`nil` is considered to be `false`).
	Output:
	`1`
	`2`
	`3`
	`4`

Standard libraries

In addition to the standard programming constructs described in this document, Lua includes standard libraries that contain useful functions for string manipulation, mathematics, and related functions. Test Script Processor (TSP®) scripting engine instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP scripting engine. These libraries are automatically loaded when the TSP scripting engine starts and do not need to be managed by the programmer.

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 the "Lua memory management" topic below 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 the "Lua memory management" topic below 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 between 2 and 36, inclusive. In bases above 10, the letter `A` (in either upper or lower case) represents 10, `B` represents 11, and so forth, with `Z` representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(x)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(v)`	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. Returns a string with length equal to the number of arguments, in which each character has the internal numeric code equal to its corresponding argument.

`string.format(`
`formatstring,`
`...)`
	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the modifiers `*`, `l`, `L`, `n`, `p`, and `h` are not supported and there is an extra option, `q`. The `q` option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written.
	For example, the call:
	`string.format('%q', 'a string with "quotes" and \n new line')`
	will produce the string:
	`"a string with \"quotes\" and \`
	`new line"`
	The options `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `X`, and `x` all expect a number as argument. `q` and `s` expect a string. This function does not accept string values containing embedded zeros, except as arguments to the `q` option.

`string.len(s)`	Receives a string and returns its length. The empty string `""` has length `0`. Embedded zeros are counted, so `"a\000bc\000"` has length `5`.
`string.lower(s)`	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
`string.rep(s, n)`	Returns a string that is the concatenation of *n* copies of the string *s*.
`string.sub(s, i)` `string.sub(s, i, j)`	Returns the substring of *s* that starts at *i* and continues until *j*; *i* and *j* can be negative. If *j* is absent, it is assumed to be equal to `-1` (which is the same as the string length). In particular, the call `string.sub(s, 1, j)` returns a prefix of *s* with length *j*, and `string.sub(s, -i)` returns a suffix of *s* with length *i*.
`string.upper(s)`	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions

Function	Description
`math.abs(x)`	Returns the absolute value of *x*.
`math.acos(x)`	Returns the arc cosine of *x*.
`math.asin(x)`	Returns the arc sine of *x*.
`math.atan(x)`	Returns the arc tangent of *x*.
`math.atan2(y, x)`	Returns the arc tangent of *y/x*, but uses the signs of both parameters to find the quadrant of the result (it also handles correctly the case of *x* being zero).
`math.ceil(x)`	Returns the smallest integer larger than or equal to *x*.
`math.cos(x)`	Returns the cosine of *x*.
`math.deg(x)`	Returns the angle *x* (given in radians) in degrees.
`math.exp(x)`	Returns the value e^x.
`math.floor(x)`	Returns the largest integer smaller than or equal to *x*.
`math.frexp(x)`	Returns `m` and `e` such that $x = m2^e$, where `e` is an integer and the absolute value of `m` is in the range `[0.5, 1]` (or zero when *x* is zero).
`math.ldexp(x, n)`	Returns $m2^e$ (`e` should be an integer).
`math.log(x)`	Returns the natural logarithm of *x*.
`math.log10(x)`	Returns the base-10 logarithm of *x*.
`math.max(x, ...)`	Returns the maximum value among its arguments.
`math.min(x, ...)`	Returns the minimum value among its arguments.
`math.pi`	The value of π (3.141592654).
`math.pow(x, y)`	Returns x^y (you can also use the expression `x^y` to compute this value).
`math.rad(x)`	Returns the angle *x* (given in degrees) in radians.

`math.random()`
`math.random(m)`
`math.random(m, n)`
	This function is an interface to the simple pseudorandom generator function `rand` provided by ANSI C.
	When called without arguments, returns a uniform pseudorandom real number in the range `[0,1]`. When called with an integer number *m*, `math.random()` returns a uniform pseudorandom integer in the range `[1, m]`. When called with two integer numbers *m* and *n*, `math.random()` returns a uniform pseudorandom integer in the range `[m, n]`.

`math.randomseed(x)`	Sets *x* as the seed for the pseudorandom generator: equal seeds produce equal sequences of numbers.
`math.sin(x)`	Returns the sine of *x*.
`math.sqrt(x)`	Returns the square root of *x*. (You can also use the expression `x^0.5` to compute this value.)
`math.tan(x)`	Returns the tangent of *x*.

Test Script Builder (TSB)

Keithley Instruments Test Script Builder (TSB) is a software tool included with your Model 2450. You can install and use TSB to develop scripts for TSP-enabled instruments.

Installing the TSB software

To install the Test Script Builder (TSB) software:

1. Close all programs.
2. Place the Test Script Builder Software Suite CD into your CD-ROM drive or start the software from the downloaded file.
3. Follow the on-screen instructions.

If you are using the CD-ROM and the web browser does not start automatically and display a screen with software installation links, open the installation file (`setup.exe`) located on the CD-ROM to start installation.

Installing the TSB add-in

When you install the Test Script Builder Software Suite, all available updates for TSB Add-in software are also installed. This includes any additional tools for the Test Script Builder Integrated Development Environment (TSB), and also Model 2450-specific examples and help files (see Installing the TSB software (on page 7-29)). In addition to the software suite, a separate add-in is provided on the product specific CD. You can use this add-in to update previous TSB software installations.

Before installing the TSB Add-in software, you must install the TSB software.

To install the TSB Add-in software:

1. Close all programs.
2. Place the Product Information CD into your CD-ROM drive.
3. Double-click the Add-in to start installation.
4. Follow the on-screen instructions.

If your web browser does not start automatically and display a screen with software installation links, open the installation file (`setup.exe`) located on the CD-ROM to start installation.

Using Test Script Builder (TSB)

Keithley Instruments Test Script Builder (TSB) is a software tool that simplifies building test scripts. You can use TSB to perform the following operations:

- Send remote commands and Lua statements
- Receive responses (data) from commands and scripts
- Upgrade instrument firmware
- Create, manage, and run user scripts
- Debug scripts
- Import factory scripts to view or edit and convert to user scripts

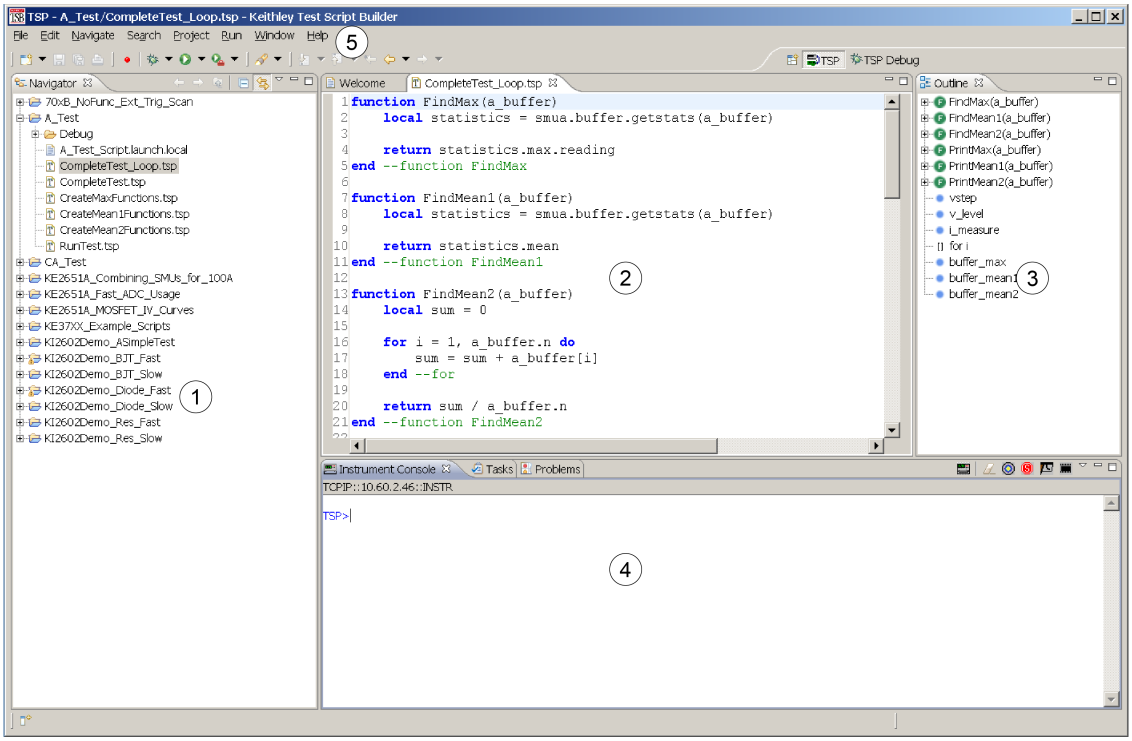
The Keithley Instruments Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands. For more information about using the Lua scripting language with Keithley TSP-enabled instruments, refer to the Fundamentals of programming for TSP (on page 7-11) section.

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 come with a number of 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 126: 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, click the plus (+) next to the project folder. To hide the folder contents, click the minus (–) 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 scripts are written, modified, and debugged.

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, click the tab that contains the script in the script editor area.

Outline view

The outline view allows you to navigate through the structure of the active script in the script editor. Double-clicking a variable name or icon causes the first instance of the variable in the active script to be highlighted.

This view shows:

- Names of local and global variables
- Functions referenced by the active script in the script editor
- Parameters
- Loop control variables
- Table variables
- Simple assignments to table fields

Programming interaction

This part of the workspace is where you interact with the scripts that you are building in Test Script Builder (TSB). The actual contents of the programming interaction area of the workspace can vary.

You can send commands from the Instrument Console command line, retrieve data, view variables and errors, and view and set breakpoints when using the debug feature.

Connecting an instrument in TSB

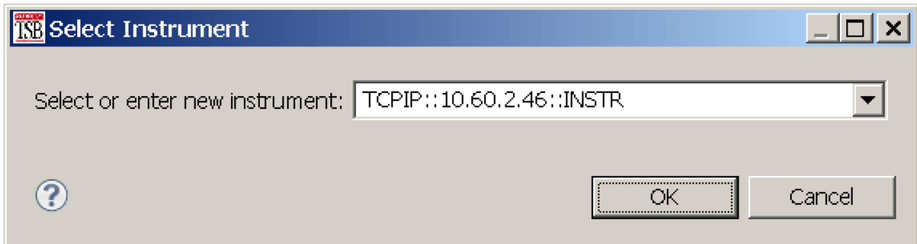
To connect the Test Script Builder software to an instrument:

1. Click the **Open Instrument** icon in the script editor toolbar.

Figure 127: Opening an instrument connection in TSB

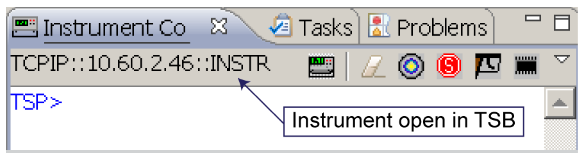
2. The Select Instrument dialog box opens. Select an existing instrument from the list, or type the VISA resource ID of the instrument in the **Select or enter new instrument** box.

Figure 128: Select Instrument dialog box



3. Click **OK**. You briefly see the Opening Resource dialog box, and then the instrument is visible in the Instrument Console.

Figure 129: Instrument connected in TSB

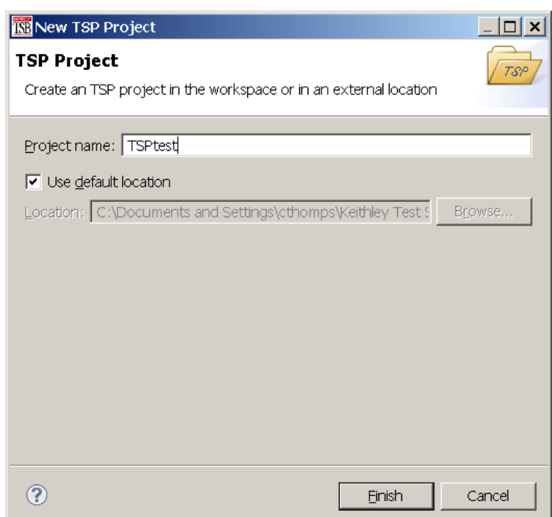


Creating a new TSP project

To create a new Test Script Processor (TSP®) project:

1. On the **File** menu in the TSP perspective, select **New > TSP Project**. The New TSP Project dialog box opens.

Figure 130: New TSP Project dialog box



2. Type a name for your project in the **Project name** box.
3. Select the location to create the new project.
4. Click **Finish**. The new project appears in the list of projects in the project navigator, and a file named `main.tsp` is created in the project. You can rename the `.tsp` file.
5. If you do not want to build your project automatically when it is saved or run, from the **Project** menu, clear **Build Automatically**.

NOTE

If you make changes to your project and do not build it before you run it, the Problems tab may not appear when problems are encountered.

Adding a new TSP file to a project

To add a new TSP file to a project:

1. Select the **File** menu and select **New > TSP File**. The New TSP File dialog box opens.
2. Select the project folder where you want to save the file.
3. Enter a name in the **File name** box.
4. Click **Finish**.

Running a script

You can run a script in the Test Script Builder (TSB) software using any of the following methods:

- Run a script that is open in the script editor area
- Run scripts that are listed in the Navigator area that are not currently open in the script editor window
- Run a collection of scripts by creating a run configuration (see Creating a run configuration (on page 7-35))

NOTE

When you use any of the run controls to run a script, the area that has focus in the workspace is important. For example, if the Navigator area is active (the tab is shaded) when you click the **Run** icon, the script file that is highlighted in the Navigator area is run instead of the active script in the script editor area.

The following list describes the most commonly used controls to run scripts in TSB:

- Right-click in the script editor area and select **Run Editor Contents** to run the active script as it currently appears in the script editor
- Right-click in the script editor area and select **Run As > 1 TSP File** to run the last saved version of the active script in the script editor as a `.tsp` file
- Select an action from the **Run** menu at the top of the TSB software interface

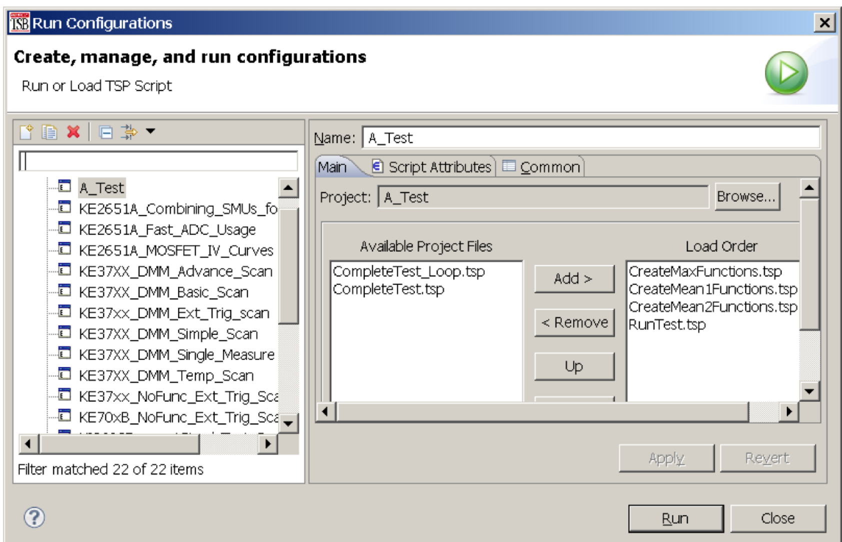
Creating a run configuration

A run configuration allows you to download multiple script files to an instrument and execute them as a single script.

To create a run configuration:

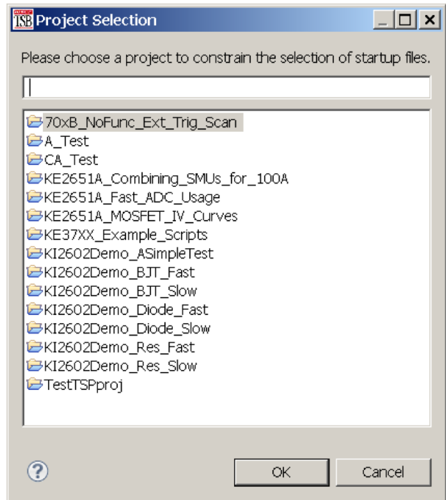
1. On the **Run** menu, select **Run Configurations**. The Run Configurations dialog box opens.
2. The left pane of the dialog box lists existing run and debug configurations. Select the script where the Run Configuration will be saved.
3. Click the **New launch configuration** icon  at the top left of the dialog box. By default, a new configuration is created with the name `New_configuration`.

Figure 131: Run Configurations dialog box



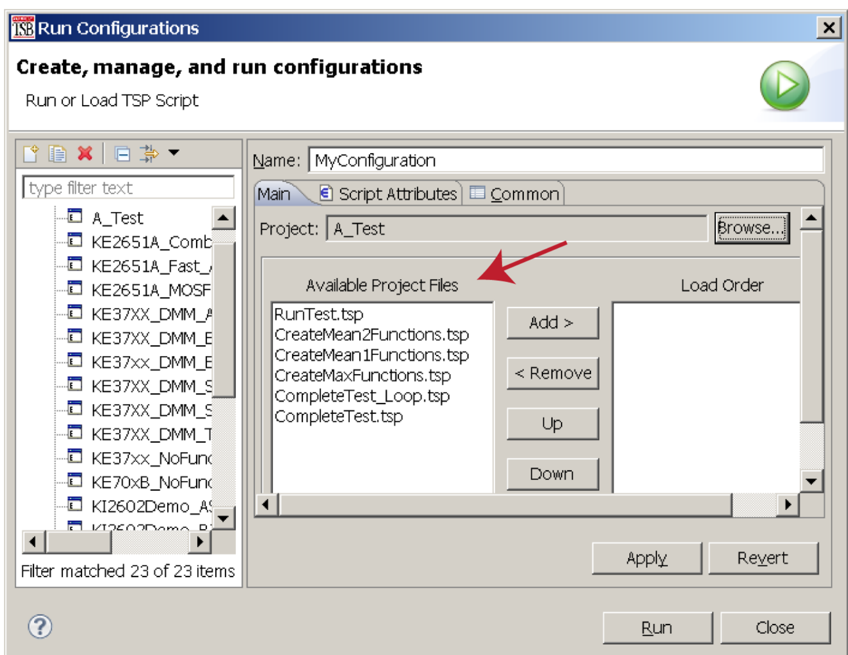
4. In the **Name** box, enter the name of your new run configuration.
5. Click the **Browse** button next the Project box.
6. Select a project from the list of available projects
7. Click **OK**.

Figure 132: Project Selection dialog box



The TSP files for the selected project are added to the Available Project Files list on the Main tab.

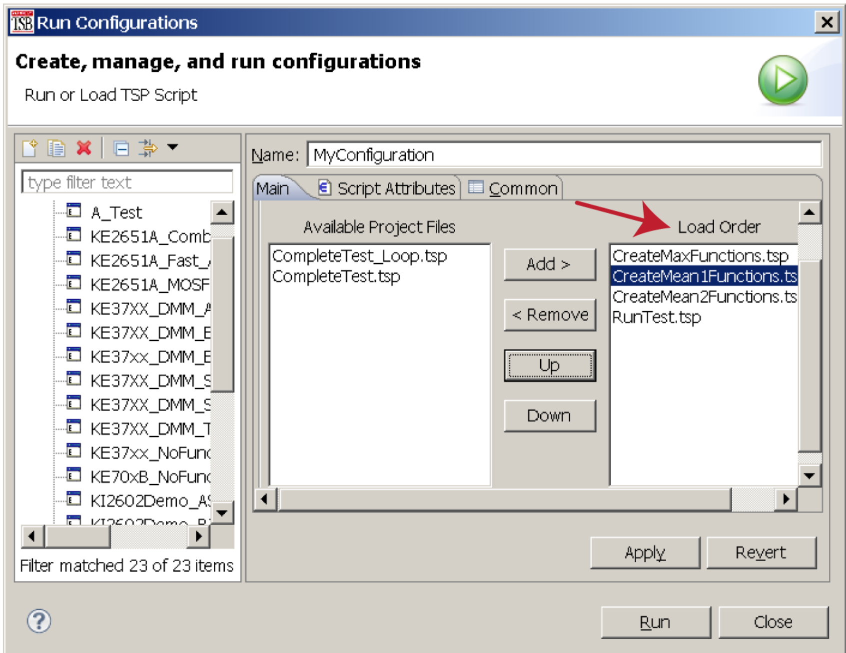
Figure 133: Available files for selected project



8. Select the files you want to add to the run configuration and click **Add** to add them to the Load Order list.

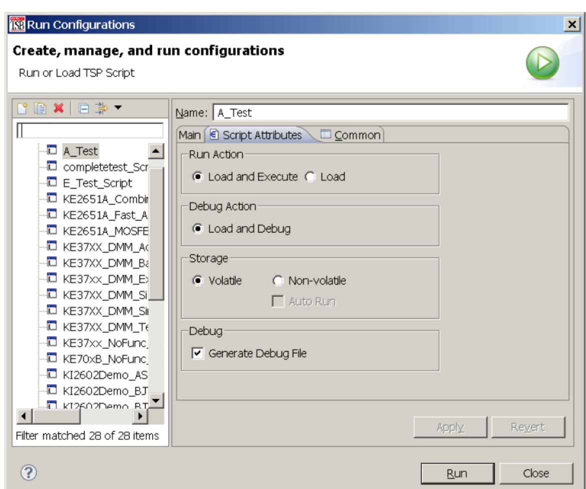
9. To change the load order of the TSP files, select the files you want to move and click **Up** or **Down** until the files are in the correct order.

Figure 134: Selected TSP files load order



10. Click **Apply**.
11. Click the **Script Attributes** tab.
12. Select one of the following:
    - **Load and Execute:** If you select this option, which is the default selection, the script automatically loads into the instrument's volatile memory (run-time environment) and executes when you click **Run**.
    - **Load:** If you select this option, the script is loaded into the instrument's volatile memory when you click **Run**, but is not executed until you manually run it. To manually run it from the command line in the Instrument Console, type `MyConfiguration.run()` (where *`MyConfiguration`* is the name of your configuration).

Figure 135: Script Attributes tab



13. In the Storage area of the Script Attributes tab, select **Volatile** or **Non-volatile**. For products that support autorun scripts, if you select Non-volatile, you can select **Auto Run** to have the script run automatically when the instrument is turned on.

    Note that all scripts are initially stored in the instrument's volatile memory and are lost if you turn the instrument power off and then on again. If you want to keep the script on the instrument through a power cycle, select **Non-volatile** storage.

14. In the Debug area of the Script Attributes tab, you can select **Generate Debug File**. When you select this option, a Debug subfolder is created in your test folder, and a file with a `.DBG` extension is created in that folder. Note that this is a feature of the Eclipse platform, and you will not use this file to debug your script. It simply contains all the scripts in your run configuration, so that you can see them together in the order in which they will load.
15. Click **Close** or **Run**. The run configuration is added to the run configurations list.

NOTE

To run the last used run configuration, click the **Run** icon in the main TSB toolbar. To run a different run configuration, right-click in the script editor area and select **Run As > Run Configurations**. Select a different run configuration, and then click **Run** in the Run Configurations dialog box.

Memory considerations for the run-time environment

The Model 2450 reserves 32 MB of memory for dynamic run-time use.

Note that the run-time environment includes user-created reading buffers and active sweep configurations. The amount of memory used by a reading buffer is approximately 30 bytes for each reading.

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 * 30) + (2500 * 30) = 105,000 bytes or 105 kilobytes

Reading buffers for remote nodes consume memory on the remote node, not the local node.

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

It is possible for the memory used for the run-time environment, sweep configuration and reading buffers to exceed 32 MB. When this occurs, there is a risk that memory allocation errors will occur and 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 run-time environment and return the instrument to a known state. Unsaved scripts and reading buffers will be lost.

Suggestions for increasing the available memory

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.
- Consider removing unused reading buffers.
- Consider resizing reading buffers to a smaller size.
- Reduce the number of TSP-Link® nodes.
- Delete unneeded global variables from the run-time environment by setting them to `nil`.
- Adjust the `collectgarbage()` settings in Lua. See Lua memory management (on page 7-26) for more information.
- Review scripts to improve their memory usage. In particular, you can see memory gains by changing string concatenation lines into a Lua table of string entries. You can then use the `table.concat()` function to create the final string concatenation.

TSP command reference

In this section:

TSP command programming notes

This section contains general information about using TSP commands.

TSP syntax rules

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

Upper and lower case

Instrument commands are case sensitive.

Function and attribute names are in lowercase characters.

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

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

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

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

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

White space

You can send commands with or without white spaces.

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

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

Parameters for functions

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

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

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

The command below resets the local node (no parameters are needed):

```
localnode.reset()
```

Multiple parameters

Multiple parameters must be separated by commas.

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

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

Time and date values

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

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

Local and remote control

The instrument can be controlled locally or remotely.

When the instrument is controlled locally, you can operate the instrument using the front-panel controls. When it is controlled remotely, you operate the instrument through a controller (usually a computer). When the instrument is first powered on, it is controlled locally.

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

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

Remote control

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

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

To switch to remote control, do one of the following actions:

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

Local control

To change to local control, do one of the following actions:

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

Using the TSP command reference

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

Figure 136: Example instrument command description

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	Instrument reset Restore configuration	Saved setup	beeper.ON

Usage

```
state = beeper.enable
beeper.enable = state
```

state	Disable the beeper: `beeper.OFF` Enable the beeper: `beeper.ON`

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 7-8)

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

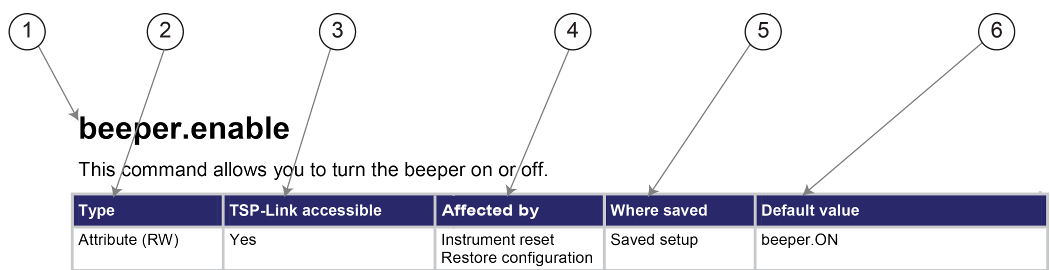
- Command name, brief description, and summary table
- Usage
- Details
- Example
- Also see

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

Command name, brief description, and summary table

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

Figure 137: 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 about commands, see Introduction to TSP operation (on page 7-1).

3 **TSP-Link accessible**. Indicates whether or not the command can be accessed through a TSP-Link network (Yes or No).

4 **Affected by**. This columns lists commands or actions that can change the value of the command. These include the following actions.

- **Power cycle:** The command settings are not saved through a power cycle.
- **Restore configuration:** If you restore a configuration script, this setting changes to the stored setting.
- **Instrument reset:** When you reset the instrument, this command is reset to its default value. Reset can be done from the front panel or when you send `reset()`, `localnode.reset()`, or `*RST`.
- **Source configuration list:** If you recall a source configuration list, this setting changes to the stored setting.
- **Measure configuration list:** If you recall a measure configuration list, this setting changes to the stored setting.
- **Function:** This command changes value when the function is changed (for example, changing from a voltage source to a current source or changing from a current measurement to a resistance measurement).

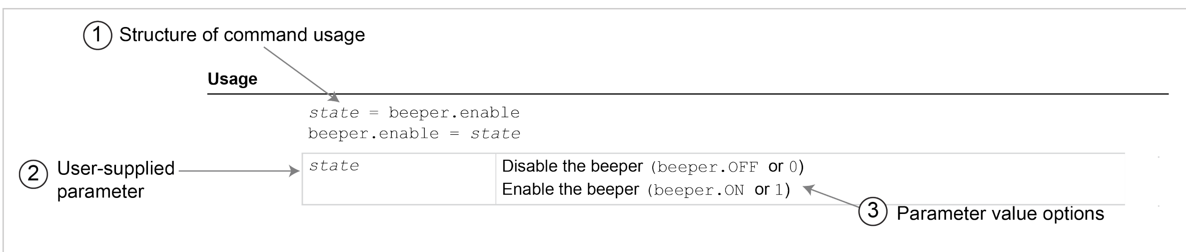
5 **Where saved**. Indicates where the command settings reside once they are used on an instrument. Options include:
- **Not saved:** Command is not saved and must be typed each time you use it.
- **Nonvolatile memory:** The command is stored in a storage area in the instrument where information is saved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

6 **Default value:** Lists the default value or constant for the command. The parameter values are defined in the Usage or Details sections of the command description.

Command usage

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

Figure 138: Command usage section



1 **Structure of command usage:** Shows how the parts of the command should be organized. If a parameter is shown to the left of the command, it is the return when you print the command. Information to the right are the parameters or other items you need to enter.

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

Figure 139: Details section of command listing

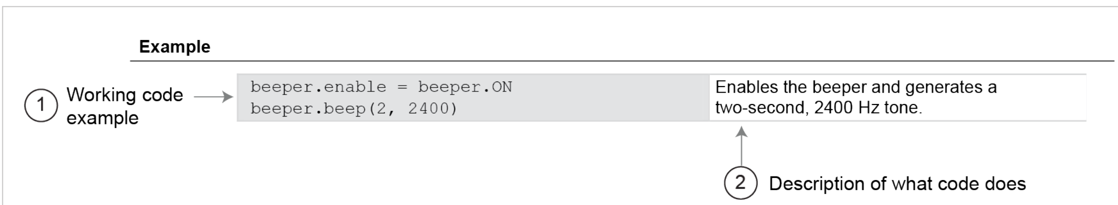
Details

This attribute enables or disables the beeper.
Disabling the beeper also disables front-panel key clicks.

Example section

The Example section of the remote command description shows some simple examples of how the command can be used.

Figure 140: Code examples in command listings



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

Related commands and information

The Also See section of the remote command description lists additional commands that are related to the command being described.

Figure 141: Links to related commands and information

Also see

beeper.beep() (on page 7-8)

TSP commands

beeper.beep()

This function generates an audible tone.

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

Usage

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

duration	The amount of time to play the tone in seconds (0.001 to 100)
frequency	The frequency of the beep (20 to 8000)

Details

You can use the beeper of the Model 2450 to provide an audible signal at a specific frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

Example

`beeper.beep(2, 2400)`	Generates a two-second, 2400 Hz tone.

Also see

None

buffer.clearstats()

This function clears all statistics from the specified buffer.

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

Usage

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

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

Details

This command clears the statistics. This command does not clear the readings.

Example

`buffer.clearstats(defbuffer1)`	Clears statistics from `defbuffer1`.
`buffer.clearstats(testData)`	Clears statistics from `testData`.

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

buffer.delete()

This function deletes a user-defined reading buffer.

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

Usage

```
buffer.delete(readingBufferVariable)
```

readingBufferVariable	The name of a user-defined reading buffer

Details

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

Example

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

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

Also see

buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

buffer.getstats()

This function returns statistics from a specified reading buffer.

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

Usage

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

bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`
statsVar	A table with the following entries: `n`, `min`, `max`, `mean`, and `stddev`; see **Details** for information on the entries

Details

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

The instrument automatically updates reading buffer statistics as data is added to the reading buffer. When the reading buffer is configured to fill continuously and overwrite older data with new data, the buffer statistics include the data that was overwritten.

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

The *`statistics`* parameter contains the values described in the following table.

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

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

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

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

Use the following commands to print these entries:

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

The commands that return minimum and maximum values each also return tables. These tables contain the following values:

Attribute	Description
`reading`	The reading value
`timestamp`	The `min.timestamp` is the timestamp of the minimum data point in the buffer and the `max.timestamp` is the timestamp of the maximum data point in the buffer

Example

`print(buffer.getstats(defbuffer1))`	Get statistics on `defbuffer1`. Returns a table.
	Output:
	`17ac630`
	The table has the following entries in it:
	`n, min, max, mean, stddev`

`defBufStats = buffer.getstats(defbuffer1)` `print(defBufStats)`	Assign the name `defBufStats` to the table. Get statistics for the default reading buffer named `defbuffer1` Returns the `defBufStats` table with the following entries in it: `n, min, max, mean, stddev`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

buffer.make()

This function creates a user-defined reading buffer.

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

Usage

`bufferVar` `= buffer.make(`*`bufferSize`*`)`

`bufferVar`	The name of a user-defined buffer
`bufferSize`	The maximum number of readings that can be stored in *`bufferVar`*; minimum is 10

Details

This function creates a user-defined reading buffer.

You cannot assign user-defined reading buffers the same name as an existing buffer, including `defbuffer1` or `defbuffer2`.

Example

`capTest2 = buffer.make(200)`	Creates a 200-element reading buffer named `capTest2`.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

buffer.save()

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

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

Usage

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

bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
fileName	A string that contains the name of the file on the USB flash drive to which to save the reading buffer
timeFormat	Defines how date and time information from the buffer is saved in the file on the USB flash drive; the values are: • Save dates, times, and fractional seconds; `buffer.SAVE_FORMAT_TIME` or `1` • Saves relative time stamps; `buffer.SAVE_RELATIVE_TIME` or `2` • Saves seconds and fractional seconds; `buffer.SAVE_RAW_TIME` or `4` • Saves time stamps; `buffer.SAVE_TIMESTAMP_TIME` or `8`
start	Defines the starting point in the buffer to start saving data
end	Defines the ending point in the buffer to stop saving data

Details

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

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

Examples of valid destination file names:

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

Invalid destination filename examples:

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

— The period is not followed by `csv`.

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

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

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

— Parameter error: filename must not contain two periods or extensions.

Example 1

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

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

Example 2

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

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

Example 3

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

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

Example 4

```
buffer.save(defbuffer1, "/usb1/defbuf1data", 4)
```

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

Also see

buffer.make() (on page 8-11)
buffer.saveappend() (on page 8-14)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

buffer.saveappend()

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

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

Usage

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

bufferVar	The name of a reading buffer from which you want to append data to the specified file
fileName	The name of the file on the USB flash drive to which the reading buffer will be appended
timeFormat	Indicates how date and time information from the buffer is saved in the file on the USB flash drive; the values are: • Save dates, times, and fractional seconds: `buffer.SAVE_FORMAT_TIME` or `1` • Saves relative time stamps: `buffer.SAVE_RELATIVE_TIME` or `2` • Saves seconds and fractional seconds: `buffer.SAVE_RAW_TIME` or `4` • Saves time stamps: `buffer.SAVE_TIMESTAMP_TIME` or `8`
start	Indicates the starting point within the buffer to save data
end	Indicates the ending point within the buffer to save data

Details

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

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

The file extension `.csv` is appended to the filename if necessary. Any file extension other than `.csv` generates errors.

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

Examples of valid destination file names:

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

Invalid destination filename examples:

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

— The period is not followed by `csv`.

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

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

Example 1

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

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

Example 2

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

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

Example 3

```
buffer.saveappend(defbuffer1, "/usb1/defbuf1data", buffer.SAVE_RAW_TIME, 1, 10)
```

Append readings and raw time stamps from `defbuffer1` to a file named `defbuf1data` on the USB flash drive. Uses the constant. 1 and 10 are start and end data points in the buffer

Example 4

```
buffer.saveappend(defbuffer1, "/usb1/defbuf1data", 4)
```

Append readings and raw time stamps from `defbuffer1` to a file named `defbuf1data` on the USB flash drive. Uses the numeric value.

Example 5

```
buffer.saveappend(defbuffer1, "/usb1/defbuf1data", 4, 1,10)
```

Append readings and raw time stamps from `defbuffer1` to a file named `defbuf1data` on the USB flash drive. 1 and 10 are start and end data points in the buffer.

Also see

Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
buffer.make() (on page 8-11)
buffer.save() (on page 8-12)

bufferVar.capacity

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

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

Usage

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

bufferCapacity	The maximum number of readings the buffer can store; the additive of all reading buffers cannot exceed 1,000,000
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer

Details

Use this command to read the number of measurements that a buffer can store.

Use this command to change the number of readings buffers can store. The number of readings that user-defined buffers can store initially is set when they are created. Default buffers can store 10000 measurements initially.

Use the `buffer.make()` command to create user-defined buffers and set their initial sizes.

Example

`reset()`
`testData = buffer.make(500)`
`capTest = buffer.make(300)`
	Create two user-defined reading buffers: *`testData`* and *`capTest`*.

`bufferCapacity = capTest.capacity`	Create a variable called `bufferCapacity` to hold the capacity of the `capTest` buffer.
`print(bufferCapacity)`	Print `bufferCapacity`. Output: `300`
`print(testData.capacity)`	Print the capacity of `testData`. Output: `500`
`testData.capacity = 600`	Changes the capacity of `testData` to 600.
`print(testData.capacity)`	Print the capacity of `testData`. Output: `600`
`print(defbuffer1.capacity)`	Print the capacity of the default buffer `defbuffer1`. Output: `10000`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.clear()

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

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

Usage

bufferVar.clear()

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

Example

reset() testData = buffer.make(50) trigger.model.load("SimpleLoop", 3, 0, testData) trigger.model.initiate() waitcomplete() printbuffer(1, testData.n, testData)	Create a reading buffer named testData, make three readings and store them in testData, and then view the readings. Print number of readings in testData. Output: -4.5010112303956e-10, -3.9923108222095e-12, -4.5013931471161e-10
testData.clear() print("Readings in buffer after clear =" .. testData.n)	Clear the readings in testData. Verify that there are no readings in testData. Output: Readings in buffer after clear = 0
trigger.model.initiate() waitcomplete() printbuffer(1, testData.n, testData)	Store three new readings in testData and view those when complete. Output: 4.923509754e-07, 3.332266330e-07, 3.974883867e-07

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.dates

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

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

Usage

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

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

Details

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

Example

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

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

Print the first reading date.

Example output:

03/01/2013

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

Example output:

03/01/2013, 03/01/2013, 03/01/2013

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.fillmode

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

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

Usage

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

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

Details

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

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

When a reading buffer is set to `buffer.FILL_CONTINUOUS`, the first new measurement is stored at *n*+1, where *n* is the number of readings stored in the buffer.

Example

`reset()`
`testData = buffer.make(50)`
`print(testData.fillmode)`
`testData.fillmode = buffer.FILL_CONTINUOUS`
`print(testData.fillmode)`
`defbuffer2.fillmode = buffer.FILL_ONCE`
`print(defbuffer2.fillmode)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	Print the fillmode setting for the `testData` buffer.
	Output:
	`0`
	Set fillmode to continuous.
	Print the fillmode setting for the `testData` buffer.
	Output:
	`1`
	Set `defbuffer2` fillmode to fill once and stop.
	Print the fillmode setting for the `defbuffer2` buffer.
	Output:
	`0`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.formattedreadings

This attribute contains the stored readings formatted as they appear on the front-panel display.

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

Usage

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

readings	Buffer reading formatted as it appears on the front-panel display for element *N*
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*`.n` command to determine the number of readings in the buffer

Details

This read-only attribute is an array (a Lua table) of strings that indicate the formatted reading as viewed on the front-panel display.

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.formattedreadings[1])`
`printbuffer(1, testData.n,`
`   testData.formattedreadings)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	
	Print the first reading formatted as it appears on the front-panel display.
	Example output:
	`-00.0e-6 nA`
	Print all readings in the reading buffer as they appear on the front-panel display.
	Example output:
	`-00.0e-6 nA, -000.0e-9 nA, -0.00e-6 nA`

Also see

bufferVar.readings (on page 8-24)
buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.fractionalseconds

This attribute contains the fractional portion of the timestamp (in seconds) when each reading occurred.

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

Usage

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

fractionalSec	The fractional portion of the timestamp (in seconds) when each reading occurred
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

This read-only attribute is an array (a Lua table) of the fractional portion of the timestamps, in seconds, when each reading occurred. Seconds are shown as fractions.

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 6, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.fractionalseconds[1])`
`printbuffer(1, 6, testData.fractionalseconds)`
	Create a reading buffer named `testData` and make six measurements.
	
	Print the fractional portion of the timestamp for the first reading in the buffer.
	Example output:
	`0.647118937`
	Print the fractional portion of the timestamp for the first six readings in the buffer.
	Example output:
	`0.647118937, 0.064543, 0.48196127, 0.89938724, 0.316800064, 0.734218263`

Also see

bufferVar.seconds (on page 8-26)
buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.logstate

This attribute indicates whether the reading buffer should log informational events.

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

Usage

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

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

Details

This command indicates whether the reading buffer should log informational events such as buffer full and buffer cleared.

Example

`reset()`
`MyBuffer = buffer.make(500)`
`print(defbuffer2.logstate)`
	Create the user-defined buffer `MyBuffer`.
	Print the logstate of `defbuffer2`.
	Output:
	1

`defbuffer2.logstate = buffer.ON`	Set the default `defbuffer2` to start logging support information.

`print(defbuffer2.logstate)`
`print(defbuffer1.logstate)`
`print(MyBuffer.logstate)`
	Print the logstate of both default buffers and the user-created buffer `MyBuffer`.
	Output:
	`1`
	`1`
	`0`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.n

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

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

Usage

`numberOfReadings` `=` *`bufferVar`*`.n`

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

Details

If you do not know how many readings are in a buffer, you can use the bufferVar.n attribute in other commands. For example, to print all of the readings in a buffer, use the following command:

```
printbuffer(1, defbuffer1.n, defbuffer1.readings)
```

However, when you use the `bufferVar.n` command, be aware of how much data is returned because the buffer in the receiving application (such as in LabView or VB net) needs to be big enough to receive all of the data.

Example

`reset()`
`testData = buffer.make(100)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.n)`
`print(defbuffer1.n)`
`print(defbuffer2.n)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	Print the number of readings in `testData`:
	Output:
	3
	Print the number of readings in `defbuffer1`:
	Example output:
	`66`
	Print the number of readings in `defbuffer2`:
	Example output:
	`0`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.readings

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

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

Usage

`readings = bufferVar.readings[N]`

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

Example 1

`reset()`
`tD = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, tD)`
`trigger.model.initiate()`
`waitcomplete()`
`print(tD.readings[1])`
	Create a reading buffer named `tD`, configure the instrument to make three measurements, and store the readings in the buffer.
	Print the first reading in `tD`.
	Output:
	`-9.6420389034124e-12`

Example 2

```
printbuffer(1, 3, tD.readings)
```

For the buffer created in Example 1, print the three readings in buffer.
Output:
```
-9.6420389034124e-12, -4.5509945811872e-10, -9.1078204006445e-12
```

Example 3

```
for x = 1, 3 do printbuffer(x, x, tD.readings, tD.sourcevalues, tD.relativetimestamps)
   end
```

For the buffer created in Example 1, print the 3 readings, including the measurement, source value, and relative time for each reading.
Output:
```
-9.6420389034124e-12, 2, 0
-4.5509945811872e-10, 2, 0.277194856
-9.1078204006445e-12, 2, 0.569614783
```

Also see

bufferVar.n (on page 8-23)
buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.relativetimestamps

This attribute contains the timestamps, in seconds, when each reading occurred relative to the timestamp of reading buffer entry number 1.

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

Usage

relativetimestamp = *bufferVar*.relativetimestamp[*N*]

relativetimestamp	The timestamps, in seconds
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

This read-only attribute is an array (a Lua table) of timestamps when each reading occurred relative to the timestamp of reading buffer entry number 1. These timestamps are equal to the time that has lapsed for each reading since the first reading was stored in the buffer. Therefore, the relative timestamp for entry number 1 in the buffer is 0.

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

`print(testData.relativetimestamps[1])`	Print the relative timestamp for the first reading in the buffer. Example output: `0`
`printbuffer(1, 3, testData.relativetimestamps)`	Print the relative timestamp for the reading 1 through 3 in the buffer. Example output: `0, 0.383541, 0.772005`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.seconds

This attribute contains the nonfractional seconds portion of the timestamp when the reading was stored in UTC format.

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

Usage

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

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

Details

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

Example 1

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 6, 0,`
`   testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.seconds[1])`
	Create a reading buffer named `testData`, configure the instrument to make six measurements, and store the readings in the buffer.
	Print the seconds portion of the first reading in `testData`.
	Example output:
	`1362261492`

Example 2

```
printbuffer(1, 6, testData.seconds)
```

For the buffer created in Example 1, print the seconds portion for readings 1 to 6 in `testData`.

Example output:

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

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.sourceformattedvalues

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

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

Usage

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

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

Details

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

Example

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

Example output:

```
-00.00041 mV
```

Example output:

```
-00.00041 mV, +00.00010 mV,
    -00.00033 mV, +00.00003
    mV, -00.00028 mV, -
    00.00045 mV
```

Also see

bufferVar.n (on page 8-23)
buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.sourcestatuses

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

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

Usage

statusInformation = *bufferVar*.sourcestatuses[*N*]

statusInformation	The status value when reading *N* of the specified buffer was acquired; Source status values are: • Complementary function output state: buffer.STAT_OUTPUT • Complementary function remote sense: buffer.STAT_SENSE • Complementary function power source limit: buffer.STAT_LIMIT • Overtemperature condition: buffer.STAT_OVER_TEMP
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

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

Buffer status bits for source function			
Bit	**Name**	**Decimal value**	**Description**
4	STAT_OVER_TEMP	16	Overtemperature condition
5	STAT_LIMIT	32	Source function was limited because the complementary function would be over the source limit
6	STAT_SENSE	64	Complementary function remote sense
7	STAT_OUTPUT	128	Complementary function output state

Example

`reset()`
`testData = buffer.make(50)`
`smu.source.output = smu.ON`
`trigger.model.load("SimpleLoop", 2, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`printbuffer(1, 2, testData.sourcestatuses)`
	Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.
	Turn on the source output.
	Print the source status for the readings in `testData`.
	Output:
	`128, 128`
	Indicating that the status is `buffer.STAT_OUTPUT`.

Also see

buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
buffer.delete() (on page 8-9)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.sourceunits

This attribute contains the units of measure of the source.

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

Usage

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

readingUnits	The units of measure of the source: • `Volt DC` • `Amp DC`
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

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

Example

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

Also see

Reading buffers (on page 3-10)
bufferVar.sourceunits (on page 8-30)
bufferVar.sourcevalues (on page 8-32)
bufferVar.statuses (on page 8-33)

bufferVar.sourcevalues

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

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

Usage

sourceValue = *bufferVar*.sourcevalues[*N*]

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

Details

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

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

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

Example

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

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

Set the source value to 1e-6 A.

Print the source values being output when readings in the reading buffer were acquired.

Example output:

```
9.9999874692e-07,
    1.0000017028e-06,
    1.0000054544e-06
```

Also see

bufferVar.sourcestatuses (on page 8-29)
bufferVar.sourceunits (on page 8-30)
bufferVar.statuses (on page 8-33)
bufferVar.formattedreadings (on page 8-20)
buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)
smu.source.readback (on page 8-144)

bufferVar.statuses

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

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

Usage

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

statusInformation	The status value when reading *N* of the specified buffer was acquired; measurement status values are: • buffer.STAT_LIMIT1_HIGH • buffer.STAT_LIMIT1_LOW • buffer.STAT_LIMIT2_HIGH • buffer.STAT_LIMIT2_LOW • buffer.STAT_TERMINAL • buffer.STAT_ORIGIN • buffer.STAT_QUESTIONABLE
bufferVar	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

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

Buffer status bits for sense measurements

Bit	Name	Decimal value	Description
1	STAT_QUESTIONABLE	2	Measure status questionable
2	STAT_ORIGIN	4	The A/D converter from which the reading originated; for the Model 2450>, this will always be 0 (Main)
3	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
4	STAT_LIMIT2_LOW	16	Measure status limit 2 low
5	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
6	STAT_LIMIT1_LOW	64	Measure status limit 1 low
7	STAT_LIMIT1_HIGH	128	Measure status limit 1 high

For information about source status values, see bufferVar.sourcestatuses (on page 8-29).

Example

`reset()`
`testData = buffer.make(50)`
`smu.source.output = smu.ON`
`trigger.model.load("SimpleLoop", 2, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`printbuffer(1, 2, testData.statuses)`
	Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.
	Turn on the source output.
	Print the source status for the readings in `testData`.
	
	Output:
	`128, 128`
	Indicating that the status is `buffer.STAT_OUTPUT`.

Also see

buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
buffer.delete() (on page 8-9)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.times

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

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

Usage

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

readingTime	Hours, minutes, and seconds
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

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

Example

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.times[1])`
`printbuffer(1, 3, testData.times)`
	This example creates a reading buffer named `testData` and makes three measurements.
	The `print()` command outputs the time of the first reading.
	Output:
	`23:09:43`
	The `printbuffer()` command outputs the time of readings 1 to 3 in the reading buffer.
	Output:
	`23:09:43, 23:09:43, 23:09:43`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.timestamps

This attribute contains the timestamps of readings stored in the reading buffer.

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

Usage

readingTimestamps = *bufferVar*.timestamps[*N*]

readingTimestamps	The timestamps of the readings in the specified buffer
bufferVar	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
N	The reading number *N*; can be any value between 1 and the number of readings in the buffer; use the *bufferVar*.n command to determine the number of readings in the buffer

Details

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

Example 1

`reset()`
`testData = buffer.make(50)`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
`trigger.model.initiate()`
`waitcomplete()`
`print(testData.timestamps[1])`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	Print the first reading date.
	Output:
	`03/01/2013 14:46:07.714614838`

Example 2

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

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

Output:

```
03/01/2013 14:46:07.714614838
03/01/2013 14:46:08.100468838
03/01/2013 14:46:08.487631838
```

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

bufferVar.units

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

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

Usage

`readingUnits = bufferVar.units[N]`

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

Details

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

Example

`reset()`
`testData = buffer.make(50)`
`testData.fillmode = buffer.FILL_CONTINUOUS`
`trigger.model.load("SimpleLoop", 3, 0, testData)`
	Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
	Set the buffer to fill continuously.

`smu.measure.func = smu.FUNC_DC_CURRENT`
`trigger.model.initiate()`
`waitcomplete()`
`printbuffer(1, testData.n, testData.units)`
	Set the measure function to current.
	Make three readings.
	Print the units for the readings.
	Output:
	`Amp DC, Amp DC, Amp DC`

`trigger.model.load("SimpleLoop", 3, 0, testData)`
`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`trigger.model.initiate()`
`waitcomplete()`
`printbuffer(1, testData.n, testData.units)`
	Set the measure function to voltage.
	Make three readings.
	Output:
	`Volt DC, Volt DC, Volt DC`

Also see

buffer.delete() (on page 8-9)
buffer.make() (on page 8-11)
bufferVar.clear() (on page 8-17)
print() (on page 8-77)
printbuffer() (on page 8-78)
Reading buffers (on page 3-10)
Remote buffer operation (on page 3-27)

createconfigscript()

This function captures most of the present settings of the instrument.

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

Usage

```
createconfigscript(scriptName)
```

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

Details

If *`scriptName`* is set to `autoexec`, the autoexec script in the instrument is replaced by the new configuration script.

If *`scriptName`* is set to the name of an existing script, an error is returned. You must delete the existing script.

Once created, the configuration script can be run and edited like any other script.

Example

`createconfigscript("August2013")`	Captures the present settings of the instrument into a script named `August2013`.

Also see

Saving setups (on page 2-112)
script.delete() (on page 8-83)

dataqueue.add()

This function adds an entry to the data queue.

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

Usage

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

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

Details

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

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

The `dataqueue.add()` function returns `false`:

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

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

Example

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

Clear the data queue.
Each line adds one item to the data queue.
Output:

```
The dataqueue contains:
1.00000e+01
1.10000e+01
1.20000e+01
```

Also see

dataqueue.CAPACITY (on page 8-39)
dataqueue.clear() (on page 8-39)
dataqueue.count (on page 8-40)
dataqueue.next() (on page 8-41)

dataqueue.CAPACITY

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

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

Usage

```
count = dataqueue.CAPACITY
```

count	The variable 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 8-38)
dataqueue.clear() (on page 8-39)
dataqueue.count (on page 8-40)
dataqueue.next() (on page 8-41)

dataqueue.clear()

This function clears the data queue.

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

Usage

```
dataqueue.clear()
```

Details

This function forces all `dataqueue.add()` commands that are in progress to time out 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 8-38)
dataqueue.CAPACITY (on page 8-39)
dataqueue.count (on page 8-40)
dataqueue.next() (on page 8-41)

dataqueue.count

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

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

Usage

count = dataqueue.count

count	The number of items in the data queue

Details

The count gets updated as entries are added 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 8-38)
dataqueue.CAPACITY (on page 8-39)
dataqueue.clear() (on page 8-39)
dataqueue.next() (on page 8-41)

dataqueue.next()

This function removes the next entry from the data queue.

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

Usage

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

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

Details

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

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

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

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

Example

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

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

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

Output:

```
There are 10 items in the data
    queue
1.0000000e+00
2.0000000e+00
3.0000000e+00
4.0000000e+00
5.0000000e+00
6.0000000e+00
7.0000000e+00
8.0000000e+00
9.0000000e+00
1.0000000e+01
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 8-38)
dataqueue.CAPACITY (on page 8-39)
dataqueue.clear() (on page 8-39)
dataqueue.count (on page 8-40)
format.asciiprecision (on page 8-62)

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 100,000 s)

Details

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

Example 1

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

Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.

Output:

```
Delay time was 0.500099
3.500000000e+01
```

Also see

None

digio.line[N].mode

This attribute sets the digital I/O line to be a digital line or trigger model line and sets the line as an input, output, or open-drain. You can also use this attribute to configure synchronous triggering modes.

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

Usage

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

lineState	Digital input: `digio.MODE_DIGITAL_IN` Digital output: `digio.MODE_DIGITAL_OUT` Digital open-drain: `digio.MODE_DIGITAL_OPEN_DRAIN` Trigger model input: `digio.MODE_TRIGGER_IN` Trigger model output: `digio.MODE_TRIGGER_OUT` Trigger model open-drain: `digio.MODE_TRIGGER_OPEN_DRAIN` Synchronous master: `digio.MODE_SYNCHRONOUS_MASTER` Synchronous acceptor: `digio.MODE_SYNCHRONOUS_ACCEPTOR`
N	The digital I/O line (1 to 6)

Details

Use this command to set the line as a digital line or as a trigger line, and configure it as an input, output, or open-drain. A digital line allows direct control of the digital I/O lines; a trigger line controls the digital I/O lines through trigger commands.

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

- `digio.MODE_DIGITAL_IN`
- `digio.MODE_DIGITAL_OUT`
- `digio.MODE_DIGITAL_OPEN_DRAIN`

The following settings of *lineState* set the line as a trigger line:

- `digio.MODE_TRIGGER_IN`
  This line state uses the edge setting specified by `trigger.digin[N].edge` attribute.
- `digio.MODE_TRIGGER_OUT`
- `digio.MODE_TRIGGER_OPEN_DRAIN`

When configured as an open-drain, the line becomes an open-drain signal with a 100 kΩ pull-up resistor. This is true when the line is configured for direct-control or for use as a trigger line. When used as a trigger line, the line uses the edge setting like `digio.MODE_TRIGGER_IN`.

When the line is set as a synchronous acceptor, the line detects the falling-edge input triggers and automatically latches and drives the trigger line low; asserting an output trigger releases the latched line. When set as a synchronous master, the line detects rising-edge triggers as input; for output, the line asserts a TTL-low pulse.

Example

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

Also see

trigger.digin[N].edge (on page 8-171)
Digital I/O port (on page 3-62)

digio.line[N].reset()

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

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

Usage

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

N	Digital I/O trigger line (1 to 6)

Details

This function resets the following attributes to their default values:

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

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

Example

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

Output before reset:

```
digio.MODE_TRIGGER_OUT    5e-05  trigger.EVENT_DIGIO5
```

Output after reset:

```
digio.MODE_TRIGGER_IN     1e-05  trigger.EVENT_NONE
```

Also see

digio.line[N].mode (on page 8-43)
trigger.digin[N].overrun (on page 8-172)
trigger.digout[N].pulsewidth (on page 8-174)
trigger.digout[N].stimulus (on page 8-175)
Digital I/O port (on page 3-62)

digio.line[N].state

This function sets a digital I/O line high or low.

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

Usage

```
digio.line[N].state = state
state = digio.line[N].state
```

N	Digital I/O trigger line (1 to 6)
state	Set the line low: `digio.STATE_LOW` or `0` Set the line high: `digio.STATE_HIGH` or `1`

Details

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

Set the state to zero (`0`) to clear the bit; any non-zero value sets the bit.

Example

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

Also see

digio.line[N].mode (on page 8-43)
digio.readport() (on page 8-45)
digio.writeport() (on page 8-46)
Digital I/O port (on page 3-62)
trigger.digin[N].edge (on page 8-171)

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 digital I/O line 1; bit B6 corresponds to digital I/O line 6.

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

Example

`data = digio.readport()` `print(data)`	Assume lines 2, 4, and 6 are set high when the I/O port is read. Output: `42` This is binary 10101010

Also see

digio.writeport() (on page 8-46)
Digital I/O port (on page 3-62)

digio.writeport()

This function writes to all digital I/O lines.

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

Usage

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

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

Details

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a *data* value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other 3 lines are set low).

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

Example

`digio.writeport(63)`	Sets digital I/O lines 1 through 6 high (binary 111111).

Also see

digio.readport() (on page 8-45)
Digital I/O port (on page 3-62)

display.changescreen()

This function changes which front-panel screen is displayed.

		Affected by	Where saved	Default value
Function	Yes			

Usage

`display.changescreen(screenName)`

screenName	The screen to display: • Home screen: `display.SCREEN_HOME` • Source swipe screen: `display.SCREEN_SOURCE_SWIPE` • Data trend swipe screen: `display.SCREEN_PLOT_SWIPE` • User display swipe screen: `display.SCREEN_USER_SWIPE` • Buffer statistics swipe screen: `display.SCREEN_STATS_SWIPE` • Settings swipe screen: `display.SCREEN_SETTINGS_SWIPE` • Graph screen: `display.SCREEN_GRAPH` • Data sheet screen: `display.SCREEN_DATASHEET`

Example

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

Clear the User Display screen and switch to display the User Display screen.
Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

display.settext() (on page 8-52)

display.clear()

This function clears the front-panel User Display swipe screen.

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

Usage

```
display.clear()
```

Details

This command clears the User Display screen.

If there are active scripts running, there might be a delay before the screen clears. The screen is cleared as soon as processing time becomes available.

Example

`display.clear()`
`display.changescreen(display.SCREEN_USER_SWIPE)`
`display.settext(display.TEXT1, "Serial number:")`
`display.settext(display.TEXT2, localnode.serialno)`
	Clear the User Display screen.
	Set the first line to read "Serial number:" and the second line to display the serial number of the instrument.

Also see

display.settext() (on page 8-52)

display.lightstate

This attribute sets the brightness of the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes			display.STATE_LCD_75

Usage

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

brightness	The brightness of the display: • 100%: `display.STATE_LCD_100` • 75%: `display.STATE_LCD_75` • 50%: `display.STATE_LCD_50` • 25%: `display.STATE_LCD_25` • Display off: `display.STATE_LCD_OFF` • Display and all indicators off: `display.STATE_BLACKOUT`

Details

This command determines the brightness of the front-panel display.

NOTE

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

Example

`display.lightstate = display.STATE_LCD_50`	Set the display brightness to 50%

Also see

None

display.prompt()

This function allows you to create interactive buttons on the front panel display.

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

Usage

promptID = display.prompt(*buttonID*, *promptText*)

promptID	The ID of the prompt
buttonID	The button to display; choose one of the following options: • display.BUTTONS_NONE • display.BUTTONS_OK • display.BUTTONS_CANCEL • display.BUTTONS_OKCANCEL • display.BUTTONS_YESNO • display.BUTTONS_YESNOCANCEL
promptText	A string that contains the text that is displayed above the buttons

Details

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

Example

```
smu.source.sweeplinear("test", 1, 10, 10)

display.prompt(display.BUTTONS_YESNO, "Would you like to start the sweep now?")

promptID, result = display.waitevent()

if result == display.BUTTON_YES then
      trigger.model.initiate()
end
```

Create a linear sweep.
Display the prompt "Would you like to start the sweep now?"
If the user presses Yes, the sweep starts.
If the user presses No, the sweep does not start.
The `promptID` can be used by `display.delete(promptID)` to remove the displayed prompt.

Also see

None

display.readingformat

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

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

Usage

```
format = display.readingformat
display.readingformat = format
```

format	Use exponent format: `display.FORMAT_EXPONENT` Use leading zeros: `display.FORMAT_PREFIX`

Details

this setting persists through reset() and a power cycle

This setting only affects the front-panel display. It does not affect the readings in buffers.

When the prefix option is selected, the display automatically shows in exponent format when the prefix format does not fit.

Example

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

Also see

None

display.settext()

This function defines the text that is displayed on the front-panel User Display screen.

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

Usage

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

userDisplayText1	String that contains the message for the top line of the User Display screen (up to 20 characters)
userDisplayText2	String that contains the message for the bottom line of the User Display screen (up to 32 characters)

Details

These commands define text messages for the User Display swipe screen.

If you enter too many characters, the instrument displays an error message and shortens the message to fit.

Example

`display.clear()`
`display.changescreen(display.SCREEN_USER_SWIPE)`
`display.settext(display.TEXT1, "Batch A122")`
`display.settext(display.TEXT2, "Test running")`
	Clear the User Display screen and switch to display the User Display screen.
	Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

display.clear() (on page 8-48)
display.changescreen (on page 8-47)

display.waitevent()

This function causes the instrument to wait for a user to respond to a prompt or button.

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

Usage

```
promptID, buttonID = display.waitevent()
promptID, buttonID = display.waitevent([timeout])
```

promptID	A number that identifies the object, such as a prompt message, that is displayed on the front panel

buttonID	`display.BUTTON_YES`
	`display.BUTTON_NO`
	`display.BUTTON_OK`
	`display.BUTTON_CANCEL`

timeout	The amount of time to wait before timing out; time is 0 to 300 seconds, where 0 waits indefinitely

Example

```
smu.source.sweeplinear("test", 1, 10, 10)

display.prompt(display.BUTTONS_YESNO, "Would you like to start the sweep now?")

promptID, result = display.waitevent()

if result == display.BUTTON_YES then
      trigger.model.initiate()
end
```

Create a linear sweep.
Display the prompt "Would you like to start the sweep now?"
If the user presses Yes, the sweep starts and displays the message "Sweep Complete!" when done.
If the user presses No, the sweep does not start.

Also see

None

eventlog.clear()

This command clears the event log.

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

Usage

```
eventlog.clear()
```

Details

This function removes all messages from the event log.

Also see

eventlog.next() (on page 8-55)
eventlog.save() (on page 8-57)

eventlog.getcount()

This function returns the number of errors in the event log.

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

Usage

```
eventlog.getcount()
eventlog.getcount(mask)
```

mask	Limits the list of event log entries; set to: • Errors only: `eventlog.SEV_ERROR` • Warnings only: `eventlog.SEV_WARN` • Information only: `eventlog.SEV_INFO` • All events: `eventlog.SEV_ALL`

Details

You can use the mask parameter to limit the event log items that are counted.

This command does not clear the event log.

Example

`print(eventlog.getcount(eventlog.SEV_INFO))`	Displays the present number of information messages in the instrument event log. If there are three information messages in the event log, output is: `3`

Also see

eventlog.clear() (on page 8-54)
eventlog.next() (on page 8-55)

eventlog.next()

This function returns the oldest message from the event log and removes it from the log.

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

Usage

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

eventNumber	The event number
message	A message the describes the event
eventType	The type of event: • Error only: 1 • Warning only: 2 • Information only: 4
nodeID	The TSP-Link node where the error occurred
timeSeconds	The time in seconds
timeNanoSeconds	The fractional seconds
eventTypeMask	Limits the list of event log entries; set to: • Errors only: `eventlog.SEV_ERROR` • Warnings only: `eventlog.SEV_WARN` • Information only: `eventlog.SEV_INFO` • All events: `eventlog.SEV_ALL`

Details

Returns the next entry from the event log.

If there are no entries in the event log, printing this value returns the following:

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

If the event type mask is not defined, the mask type for all events is used.

The event number can be used with the status model to map events to bits in the event registers.

Example

`print(eventlog.next())`
Get the oldest message in the event log. Example output: `-285 TSP Syntax error at line 1: unexpected symbol near `0' 1 0 1367806152 652040060`

Also see

eventlog.clear() (on page 8-54)
eventlog.getcount() (on page 8-54)
eventlog.save() (on page 8-57)
status.operation.setmap() (on page 8-157)
status.questionable.setmap() (on page 8-160)

eventlog.post()

This function allows you to post messages to the event log.

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

Usage

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

message	String that contains the message
eventType	The type of event; if no event is defined, defaults to `eventlog.SEV_INFO`: • `eventlog.SEV_INFO` • `eventlog.SEV_ERROR` • `eventlog.SEV_WARN`

Details

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

You must set the Log Warnings and Log Information options to be reported using the front panel to have the custom warning and information events placed into the event log.

Example

`eventlog.clear()`
`eventlog.post("my error", eventlog.SEV_ERROR)`
`print(eventlog.next())`
	Posts an error named "my error."
	Output:
	`1005 User: my error 1 0 1359414094 769632040`

Also see

None

eventlog.save()

This function saves the event log to a file.

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

Usage

```
eventlog.save(filename)
eventlog.save(filename, eventMask)
```

filename	A string that represents the name of the file to be saved
eventMask	Limits the list of event log entries; set to: • Errors only: `eventlog.SEV_ERROR` • Warnings only: `eventlog.SEV_WARN` • Information only: `eventlog.SEV_INFO` • All events: `eventlog.SEV_ALL`

Details

This command saves all event log entries since the last clear command to a USB flash drive.

You must insert the USB flash drive before sending this command.

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

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

Example

`eventlog.save("WarningsApril", eventlog.SEV_WARN)`	Save warning messages to a `.csv` file on a USB flash drive

Also see

eventlog.next() (on page 8-55)

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 8-252)

file.close()

This function closes a file on the USB flash drive.

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

Usage

`file.close(fileNumber)`

fileNumber	The file number from file open command to close

Details

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

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

`"/usb1/"`

Example

```
file_num = file.open("/usb1/SWEEPTRIGGER", file.MODE_WRITE)
file.close(file_num)
```

Open the file `SWEEPTRIGGER` for writing, then close it.

Also see

file.open() (on page 8-59)

file.flush()

This function writes buffered data to a file.

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

Usage

`file.flush(fileNumber)`

fileNumber	The file number from file open command of the file to flush

Details

The `file.write()` function buffers data, which may not be written immediately to the USB flash drive. Use `file.flush()` to flush this data. Using this function removes the need to close a file after writing to it, which allows 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.

Also see

None

file.mkdir()

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

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

Usage

`file.mkdir(path)`

path	The path of the directory

Details

The directory path must be absolute.

Example

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

Also see

None

file.open()

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

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

Usage

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

fileNumber	A number identifying the open file that you use with other file commands to write, read, flush, or close the file after opening
fileName	The file name to open, including the full path of file
accessType	The type of action to do: • Append the file: `file.MODE_APPEND` • Read the file: `file.MODE_READ` • Write to the file: `file.MODE_WRITE`

Details

The path to the file to open must be absolute.

Example

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

Also see

None

file.read()

This function reads data from a file.

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

Usage

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

fileContents	The contents of the file based on the *readAction* parameter
fileNumber	The file number from file open command of the file to read
readAction	The action to do: • Return the next line; returns `nil` if the present file position is at the end of the file: `file.READ_LINE` • Return 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: `file.READ_NUMBER` • Return the whole file, starting at the present position; returns an empty string if the present file position is at the end of the file: `file.READ_ALL`

Details

This command reads data from a file.

Example

`file_num = file.open("/usb1/testfile.txt", file.MODE_READ)`
`if file_num  != nil then`
`file_contents = file.read(file_num, file.READ_ALL)`
`file.close(file_num)`
`end`
	Open `testfile.txt` on the thumb drive for reading. If it opens successfully, read the entire contents of the file and store it in variable `file_contents`.
	Close the file.

Also see

None

file.usbdriveexists()

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

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

Usage

```
driveInserted = file.usbdriveexists()
```

driveInserted	`0` if no flash drive is detected `1` if a flash drive is detected

Details

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

Example

`print(file.usbdriveexists())`	If the USB drive is not inserted in the USB connector on the front panel, this returns `0`.

Also see

None

file.write()

This function writes data to a file.

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

Usage

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

fileNumber	The file number from file open command of the file to which to write
string	The data to write to the file

Example

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

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

Also see

None

format.asciiprecision

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

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

Usage

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

precision	A number representing the number of digits to be printed for numbers printed with the `print()`, `printbuffer()`, and `printnumber()` functions; must be a number between `1` and `16`; set to `0` to have the instrument select the precision automatically

Details

This attribute specifies the precision (number of digits) for numeric data printed with the `print()`, `printbuffer()`, and `printnumber()` functions. The `format.asciiprecision` attribute is only used with the ASCII format. The precision value must be a number between 1 and 16.

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

Example

`format.asciiprecision = 10`
`x = 2.54`
`printnumber(x)`
`format.asciiprecision = 3`
`printnumber(x)`
	Output:
	`2.540000000e+00`
	
	`2.54e+00`

Also see

format.byteorder (on page 8-63)
format.data (on page 8-64)
print() (on page 8-77)
printbuffer() (on page 8-78)
printnumber() (on page 8-81)

format.byteorder

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

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

Usage

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

order	Byte order value as follows: • Most significant byte first: `format.BIGENDIAN` • Least significant byte first: `format.LITTLEENDIAN`

Details

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

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

Example

`x = 1.23`
`format.data = format.REAL32`
`format.byteorder = format.LITTLEENDIAN`
`printnumber(x)`
`format.byteorder = format.BIGENDIAN`
`printnumber(x)`
	Output depends on the terminal program you use, but will look something like:
	`#0¤p??`
	`#0??p¤`

Also see

format.asciiprecision (on page 8-62)
format.data (on page 8-64)
printbuffer() (on page 8-78)
printnumber() (on page 8-81)

format.data

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

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

Usage

```
value = format.data
format.data = value
```

value	The format to use for data, set to one of the following values: • ASCII format: `format.ASCII` • Single-precision IEEE Std 754 binary format: `format.REAL32` • Double-precision IEEE Std 754 binary format: `format.REAL64`

Details

You can control the precision of numeric values with the `format.asciiprecision` attribute. If `format.REAL32` or `format.REAL64` is selected, you can select the byte order with the `format.byteorder` attribute.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with “#0” and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

Example

`format.asciiprecision = 10`
`x = 3.14159265`
`format.data = format.ASCII`
`printnumber(x)`
`format.data = format.REAL64`
`printnumber(x)`
	Output a number represented by *x* in ASCII using a precision of 10, then output the same number in binary using double precision format.
	Output:
	`3.141592650e+00`
	`#0ñÔÈSû!   @`

Also see

format.asciiprecision (on page 8-62)
format.byteorder (on page 8-63)
printbuffer() (on page 8-78)
printnumber() (on page 8-81)

gpib.address

This attribute contains the GPIB address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Nonvolatile memory	18

Usage

```
address = gpib.address
gpib.address = address
```

address	The GPIB address of the instrument (0 to 30)

Details

The address can be set to any address value between 0 and 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow ample time for the command to be processed before attempting to communicate with the instrument again. After sending this command, make sure to use the new address to communicate with the instrument.

The `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:
	`2.600000e+01`

Also see

GPIB setup (on page 2-46)

lan.ipconfig()

This function specifies the LAN configuration for the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No	Rear panel LAN reset	Nonvolatile memory	lan.MODE_AUTO

Usage

```
method, ipV4Address, subnetMask, gateway = lan.ipconfig()
lan.ipconfig(method)
lan.ipconfig(method, ipV4Address)
lan.ipconfig(method, ipV4Address, subnetMask)
lan.ipconfig(method, ipV4Address, subnetMask, gateway)
```

method	The method for configuring LAN settings; it can be one of the following values: `lan.MODE_AUTO`: The instrument automatically assigns LAN settings `lan.MODE_MANUAL`: You must specify the LAN settings
ipV4Address	LAN IP address; must be a string specifying the IP address in dotted decimal notation
subnetMask	The LAN subnet mask; must be a string in dotted decimal notation
gateway	The LAN default gateway; must be a string in dotted decimal notation

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used.

Example

```
lan.ipconfig(lan.MODE_AUTO)
print(lan.ipconfig())
lan.ipconfig(lan.MODE_MANUAL, "192.168.0.7", "255.255.240.0",
   "192.168.0.3")
print(lan.ipconfig())
```

Set the IP configuration method to automatic. Request the IP configuration. Example output:

```
lan.MODE_AUTO 134.63.78.136   255.255.254.0   134.63.78.1
```

Set the IP configuration method to manual. Request the IP configuration. Output:

```
lan.MODE_MANUAL 192.168.0.7    255.255.240.0   192.168.0.3
```

Also see

None

lan.lxidomain

This attribute contains the LXI domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0

Usage

```
domain = lan.lxidomain
lan.lxidomain = domain
```

domain	The LXI domain number (0 to 255)

Details

This attribute sets the LXI domain number.

All outgoing LXI packets are generated with this domain number. All inbound LXI packets are ignored unless they have this domain number.

Example

`print(lan.lxidomain)`	Displays the LXI domain.

Also see

None

lan.macaddress

This attribute describes the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	Not applicable	Not applicable	Not applicable

Usage

```
MACaddress = lan.macaddress
```

MACaddress	The MAC address of the instrument.

Details

The MAC address is a character string representing the instrument's MAC address in hexadecimal notation. The string includes colons that separate the address octets.

Example

`print(lan.macaddress)`	Returns the MAC address. For example: `00:60:1A:00:00:57`

Also see

lan.ipconfig() (on page 8-66)

localnode.access

This attribute contains the type of access users have to the instrument through different interfaces.

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

Usage

```
accessType = localnode.access
localnode.access = accessType
```

accessType	The type of access: • Full access for all users from all interfaces: `localnode.ACCESS_FULL` • Allows access by one remote interface at a time with logins required from other interfaces: `localnode.ACCESS_EXCLUSIVE` • Allows access by one remote interface at a time with passwords required on all interfaces: `localnode.ACCESS_PROTECTED` • Allows access by one interface (including the front panel) at a time with passwords required on all interfaces: `localnode.ACCESS_LOCKOUT`

Details

When access is set to full, the instrument accepts commands from any interface with no passwords required.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. To use another interface, log out of the present interface before logging into the new interface. You do not need a password with this access.

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

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

Under any access type, if a script is running on one interface when a command comes in from another interface, the command is ignored and the message "Script running from another interface" occurs.

The command `*idn?` is permitted from any interface in all access types.

Example

`localnode.access = localnode.ACCESS_FULL`
`login admin`
`logout`
	Set the instrument access to locked out.
	Log into the interface using the default password.
	Log out of the interface.

Also see

localnode.password (on page 8-70)

localnode.gettime()

This function retrieves the instrument date and time.

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

Usage

```
localnode.gettime()
```

Details

The time is returned in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

`print(os.date('%c', gettime()))`	Example output: `Wed Mar 31 14:25:31 2010`

Also see

localnode.settime() (on page 8-72)

localnode.linefreq

This attribute contains the power line frequency setting that is used for NPLC calculations.

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

Usage

frequency = `localnode.linefreq`

frequency	The detected line frequency: 50 or 60

Details

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

If you are using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].linefreq`.

Example

`frequency = localnode.linefreq` `print(frequency)`	Reads line frequency setting.

Also see

None

localnode.model

This attribute stores the model number.

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

Usage

```
model = localnode.model
```

model	The model number of the instrument

Details

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].model`.

Example

`print(localnode.model)`	Outputs the model number of the local node. For example: `2450`

Also see

localnode.serialno (on page 8-72)

localnode.password

This attribute stores the instrument password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	No	LAN reset	Nonvolatile memory	"admin"

Usage

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

passWord	A string that contains the instrument password (maximum 30 characters)

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

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.

If you forget the password, you can reset the password to the default. On the front panel, press **MENU**. Under System, select **Manage**. Select **LAN and Password Reset**. You can also reset the password from the rear panel by inserting a straightened paper clip into hole below LAN RESET.

If you are using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].password`.

Example

`localnode.password = "N3wpa55w0rd"`	Changes the password to `N3wpa55w0rd`.

Also see

localnode.access (on page 8-68)

localnode.prompts

This attribute determines if the instrument generates prompts in response to command messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.DISABLE

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting	Do not generate prompts: `localnode.DISABLE` Generate prompts: `localnode.ENABLE`

Details

The command messages do not generate prompts. The instrument generates prompts in response to command messages.

There are three prompts that might be generated:

- **TSP>** is the standard prompt. This prompt indicates that everything is normal and the command is done processing.
- **TSP?** is issued if there are entries in the event queue when the prompt is issued. Like the TSP> prompt, it indicates the command is done processing. It does not mean the previous command generated an event, only that there are still events in the queue when the command was done processing.
- **>>>>** is the continuation prompt. This prompt is used when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the current command.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].prompts`.

NOTE

Do not disable prompting when using Test Script Builder. Test Script Builder requires prompts and sets the prompting mode automatically. If you disable prompting, the instrument will stop responding when you communicate with Test Script Builder because it is waiting for a common complete prompt from Test Script Builder.

Example

`localnode.prompts = localnode.ENABLE`	Enable prompting.

Also see

tsplink.initialize() (on page 8-231)

localnode.serialno

This attribute stores the instrument's serial number.

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

Usage

```
serialno = localnode.serialno
```

serialno	The serial number of the instrument

Details

This indicates the instrument serial number.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].serialno`.

Example

```
display.clear()
display.settext(display.TEXT1, localnode.serialno)
```

Clears the instrument display.
Places the serial number of this instrument on the top line of the User Screen display.

Also see

localnode.model (on page 8-70)
localnode.version (on page 8-74)

localnode.settime()

This function sets the date and time of the instrument.

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

Usage

```
localnode.settime(year, month, day, hour, minute, second)
localnode.settime(hour, minute, second)
localnode.settime(os.time({year, month, day}))
localnode.settime(os.time({year = year, month = month, day = day, hour = hour,
   minute = minute, second = second}))
```

year	Year; must be more than 1970
month	Month (1 to 12)
day	Day (1 to 31)
hour	Hour in 24-hour time format (0 to 23)
minute	Minute (0 to 59)
second	Second (0 to 59)

Details

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

If you use `os.time()` but do not specify the time (hour, minute, and second) options, they default to noon for that day. When called without a parameter (the first form), the function returns the current time.

Example 1

`localnode.settime(2010, 3, 31, 14, 25, 0)`	Sets the date and time to Mar 31, 2010 at 2:25 pm.

Example 2

```
systemTime = os.time({year = 2010,
       month = 3,
       day = 31,
       hour = 14,
       min = 25})
localnode.settime(systemTime)
print(os.date('%c', gettime()))
```

Sets the date and time to Mar 31, 2010 at 2:25 pm.

Output:

```
Wed Mar 31 14:25:31 2010
```

Also see

localnode.gettime() (on page 8-69)

localnode.showevents

This attribute sets whether or not the instrument automatically sends generated events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	0 (no events sent)

Usage

```
errorMode = localnode.showevents
localnode.showevents = errorMode
```

errorMode	The errors that are returned: • No events: `0` • Errors only: `1` (`eventlog.SEV_ERR`) • Warnings only: `2` (`eventlog.SEV_WARN`) • Errors and warnings: `3` (`eventlog.SEV_ERR + eventlog.SEV_WARN`) • Information only: `4` (`eventlog.SEV_INFO`) • Information and errors: `5` (`eventlog.SEV_INFO + eventlog.SEV_ERR`) • Warnings and information: `6` (`eventlog.SEV_INFO + eventlog.SEV_WARN`) • All events: `7` (`eventlog.SEV_ALL`)

Details

Enable this attribute to have the instrument automatically send generated events that are stored in the event log. The event log is cleared when the event is sent.

Events are processed after a command message is executed but before prompts are issued (if prompts are enabled with `localnode.prompts`).

If this attribute is disabled, errors are left in the event log and must be explicitly read or cleared.

If you are using this command from a remote node, replace `localnode` with the node reference, such as `node[5].showevents`.

Example

`localnode.showevents = 4`	Send generated warning and information messages, but do not send error messages.

Also see

eventlog.clear() (on page 8-54)
localnode.prompts (on page 8-71)

localnode.version

This attribute stores the instrument version.

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

Usage

`version` `= localnode.version`

`version`	Instrument version level

Details

This attribute indicates the version number of the firmware that is presently running in the instrument.

When using this command from a remote node, `localnode` should be replaced with the node reference. For example, `node[5].version`.

Example

`print(localnode.version)`	Outputs the present version level. Example output: 1.0.0a

Also see

localnode.model (on page 8-70)
localnode.serialno (on page 8-72)

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 64)
scriptCode	A string containing the source code

Details

This command is only applicable to TSP-Link systems. You can use this command to use the remote master node to run a script on the specified node. This function does not run test scripts on the master node; only on the subordinate node when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

Example 1

`node[2].execute(sourcecode)`	Runs script code on node 2. The code is in a string variable called `sourcecode`.

Example 2

`node[3].execute("x = 5")`	Runs script code in string constant (`"x = 5"`) to set `x` equal to 5 on node 3.

Example 3

`node[32].execute(TestDut.source)`	Runs the test script stored in the variable `TestDut` (previously stored on the master node) on node 32.

Also see

tsplink.group (on page 8-230)

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 run-time environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link® network.

Example

`print(node[5].getglobal("test_val"))`	Retrieves and outputs the value of the global variable named test_val from node 5.

Also see

node[N].setglobal() (on page 8-76)

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®.

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 8-75)

opc()

This function sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

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

Usage

```
opc()
```

Details

This function causes the operation complete bit in the Status Event Status Register to be set when all previously started local overlapped commands are complete.

Note that each node independently sets its operation complete bits in its own status model. Any nodes that are not actively performing overlapped commands set their bits immediately. All remaining nodes set their own bits as they complete their own overlapped commands.

Example

`opc()` `waitcomplete() print([[1]])`	Output: `1`

Also see

*OPC (on page B-6)
Status model (on page C-1)
waitcomplete() (on page 8-252)

print()

This function generates a response message.

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

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments are separated with a tab character.

Numbers are printed using the `format.asciiprecision` attribute. If you want use Lua formatting, print the return value from the `tostring()` function.

Example 1

`x = 10` `print(x)`	Example of an output response message: `1.00000e+01` Note that your output might be different if you set your ASCII precision setting to a different value.

Example 2

`x = 10` `print(tostring(x))`	Example of an output response message: `10`

Also see

format.asciiprecision (on page 8-62)

printbuffer()

This function prints data from tables or reading buffer subtables.

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

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

startIndex	Beginning index of the buffer to print; this must be more than one and less than *endIndex*
endIndex	Ending index of the buffer to print; this must be more than startIndex and less than the index of the last entry in the tables
bufferVar	Name of first table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
bufferVar2	Second table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
bufferVarN	The last table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
...	One or more tables or reading buffer subtables separated with commas

Details

If *startIndex* is set to less than 1 or if *endIndex* is more than the size of the index, 9.910000e+37 is returned for each value outside the allows index and an event is generated.

When any given reading buffers are used in overlapped commands that have not yet completed (at least to the desired index), this function outputs data as it becomes available.

When there are outstanding overlapped commands to acquire data, *n* refers to the index that the last entry in the table will have after all the readings have completed.

If you pass a reading buffer instead of a reading buffer subtable, the default subtable for that reading buffer is used.

This command generates a single response message that contains all data.

The `format.data` attribute controls the format of the response message.

You can use the *bufferVar* attributes that are listed in the following table with the print buffer command. Add them after the `bufferVar` parameter.

Attribute	Description
bufferVar.n	The number of readings in the specified buffer. See bufferVar.n (on page 8-23).
bufferVar.readings	The readings stored in a specified reading buffer. See bufferVar.readings (on page 8-24).
bufferVar.dates	The dates of readings stored in the reading buffer. See bufferVar.dates (on page 8-18).
bufferVar.statuses	The status values of readings in the reading buffer. See bufferVar.statuses (on page 8-33).
bufferVar.formattedreadings	The stored readings formatted as they appear on the front-panel display. See bufferVar.formattedreadings (on page 8-20).
bufferVar.sourceformattedvalues	The source levels formatted as they appear on the front-panel display when the readings in the reading buffer were acquired. See bufferVar.sourceformattedvalues (on page 8-28).
bufferVar.sourcevalues	The source levels that were being output when readings in the reading buffer were acquired. See bufferVar.sourcevalues (on page 8-32).
bufferVar.sourcestatuses	The source status conditions of the instrument for the reading point. See bufferVar.sourcestatuses (on page 8-29).
bufferVar.times	The time when the instrument made the readings. See .bufferVar.times (on page 8-34).
bufferVar.timestamps	The timestamps of readings stored in the reading buffer. See bufferVar.timestamps (on page 8-35).
bufferVar.relativetimestamps	The timestamps, in seconds, when each reading occurred relative to the timestamp of reading buffer entry number 1. See bufferVar.relativetimestamps (on page 8-25).
bufferVar.sourceunits	The units of measure of the source. See bufferVar.sourceunits (on page 8-30).
bufferVar.seconds	The nonfractional seconds portion of the timestamp when the reading was stored in UTC format. See bufferVar.seconds (on page 8-26).
bufferVar.fractionalseconds	The fractional portion of the timestamp (in seconds) of when each reading occurred. See bufferVar.fractionalseconds (on page 8-21).
bufferVar.units	The unit of measure that is stored with readings in the reading buffer. See bufferVar.units (on page 8-36).

Example 1

```
reset()
testData = buffer.make(200)
format.data = format.ASCII
format.asciiprecision = 6
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.readings, testData.units,
   testData.relativetimestamps)
```

This assumes that `testData` is a valid reading buffer in the run-time environment. The use of `testData.n` (*`bufferVar`*`.n`) indicates that the instrument should output all readings in the reading buffer. In this example, `testBuffer.n` equals 6.

Example of output data:

```
1.10458e-11, Amp DC, 0.00000e+00, 1.19908e-11, Amp DC, 1.01858e-01, 1.19908e-11, Amp DC,
2.03718e-01, 1.20325e-11, Amp DC, 3.05581e-01, 1.20603e-11, Amp DC, 4.07440e-01, 1.20325e-
11, Amp DC, 5.09299e-01
```

Example 2

```
for x = 1, testData.n do
printbuffer(x,x,testData, testData.units, testData.relativetimestamps)
end
```

Using the same buffer created in Example 1, output readings, units and relative timestamps on a separate line for each reading.

```
1.10458e-11, Amp DC, 0.00000e+00
1.19908e-11, Amp DC, 1.01858e-01
1.19908e-11, Amp DC, 2.03718e-01
1.20325e-11, Amp DC, 3.05581e-01
1.20603e-11, Amp DC, 4.07440e-01
1.20325e-11, Amp DC, 5.09299e-01
```

Also see

bufferVar.n (on page 8-23)
bufferVar.readings (on page 8-24)
format.asciiprecision (on page 8-62)
format.byteorder (on page 8-63)
format.data (on page 8-64)
printnumber() (on page 8-81)

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 8-62)
format.byteorder (on page 8-63)
format.data (on page 8-64)
print() (on page 8-77)
printbuffer() (on page 8-78)

reset()

This function resets commands to their default settings.

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

Usage

```
reset()
reset(system)
```

system	`true`: If the node is the master, the entire system is reset `false`: Only the local group is reset

Details

The `reset()` command in its simplest form resets the entire TSP-enabled system, including the controlling node and all subordinate nodes.

If you want to reset a specific instrument, use the `node[X].reset()` command. Also use the `node[X].reset()` command to reset an instrument on a subordinate node.

When no value is specified for *system*, the default value is `true`.

You can only reset the entire system using `reset(true)` if the node is the master. If the node is not the master node, executing this command generates an error.

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

None

script.delete()

This function deletes a script from the run-time memory and nonvolatile memory.

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

Usage

```
script.delete(scriptName)
```

scriptName	The string that represents the name of the script

Details

When a script is deleted, the global variable referring to this script is also deleted.

Example

`script.delete("test8")`	Deletes a user script named `test8` from nonvolatile memory and the global variable named `test8`.

Also see

Deleting a user script using a remote interface (on page 7-9)
scriptVar.save() (on page 8-84)

script.load()

This function creates a script from a specified file.

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

Usage

```
script.load(file)
scriptVar = script.load(file)
```

file	The path and file name of the script file to load; if *scriptVar* is not defined, this name is used as the global variable name for this script
scriptVar	The created script; a global variable with this name is used to reference the script

Details

The named that is used for *scriptVar* must not already exist as a global variable. In addition, the *scriptVar* name must be a global reference and not a local variable, table, or array.

For external scripts, the root folder of the USB flash drive has the absolute path `/usb1/`.

Example

`test8 = script.load("/usb1/testSetup.tsp")`	Loads the script with the file name `testSetup.tsp` that is on the USB flash drive and names it `test8`.

Also see

None

scriptVar.run()

This function runs a script.

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

Usage

scriptVar.run()
scriptVar()

scriptVar	The name of the variable that references the script

Details

The *scriptVar*.run() function runs the script referenced by *scriptVar*. You can also run the script by using *scriptVar*().

Example

test8.run()	Runs the script referenced by the variable test8.

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory or to a USB flash drive.

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

Usage

scriptVar.save()
scriptVar.save(*filename*)

scriptVar	The name of variable that references the script
filename	The file name to use when saving the script to a USB flash drive

Details

The *scriptVar*.save() function saves a script to nonvolatile memory or a USB flash drive. The root folder of the USB flash drive has the absolute path /usb1/.

If no *filename* is specified, the script is saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

If no *filename* is specified (the filename 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 flash drive.

Also see

Working with scripts (on page 7-5)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	Not applicable	Not saved	Not applicable

Usage

```
code = scriptVar.source
```

scriptVar	The name of the variable that references the script that contains the source code
code	The body of the script

Details

The body of the script is a single string with lines separated by the new line character.

Example

```
print(test7.source)
```

Assuming a script named `test7` was created on the instrument, this example retrieves the source code.
Output:

```
reset()
display.settext(display.TEXT1, "Text on User Display 1")
display.settext(display.TEXT2, "Text on line 2")
```

Also see

scriptVar.save() (on page 8-84)

smu.interlock.tripped

This attribute indicates that the interlock has been tripped.

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

Usage

```
interlockStatus = smu.interlock.tripped
```

interlockStatus	The status of the interlock: • `smu.OFF`: The interlock is not asserted and the 200 V range is disabled; lower voltage ranges are available • `smu.ON`: The interlock signal is asserted and all voltage ranges are available

Details

This command gives you the status of the interlock. When the safety interlock signal is asserted, all voltage ranges of the instrument are available. However, when the safety interlock signal is not asserted, the 200 V range is disabled, limiting the nominal output to ±37 V.

When the interlock is not asserted:

- The front-panel INTERLOCK indicator is on.
- High voltage ranges are disabled.
- An event message is generated when you attempt to turn on the source with a voltage higher than ±20 V.

Example

`print(smu.interlock.tripped)`	If the interlock is not asserted, returns `smu.OFF`. If the interlock is asserted, returns `smu.ON`.

Also see

None

smu.measure.autorange

This attribute determines if the measurement range is set manually or automatically for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
autoRange = smu.measure.autorange
smu.measure.autorange = autoRange
```

autoRange	Set the measurement range manually: `smu.OFF` Set the measurement range automatically: `smu.ON`

Details

This command determines how the measurement range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was selected automatically.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement. The instrument sets the range when a measurement is requested.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT` `smu.measure.autorange = smu.ON`	Set the measurement function to current. Set the range to be set automatically.

Also see

Ranges (on page 2-101)
smu.measure.range (on page 8-117)

smu.measure.autorangehigh

This attribute reads the highest measurement range that is used when the instrument selects the measurement range automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Resistance: 2e5

Usage

```
highRange = smu.measure.autorangehigh
smu.measure.autorangehigh = highRange
```

highRange	The highest voltage or resistance measurement range that is used when the range is set automatically: • Voltage: 2e-2 to 200 volts • Resistance: 20 to 2e8 ohms

Details

This command is only available for voltage and resistance measurements. It can be written to and read for resistance measurements. For voltage measurements, it can only be read.

For voltage measurements, the upper limit is controlled by the voltage limit.

Resistance measurements only: You can use this command when automatic range selection is enabled to put an upper bound on the range that is used for resistance measurements.

The upper limit must be more than the lower limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.autorange = smu.ON`
`print(smu.measure.autorangehigh)`
	Sets the measurement function to voltage and turn autorange on. Check the high range for voltage measurements.

Also see

Ranges (on page 2-101)
reset() (on page 8-82)
smu.measure.autorange (on page 8-86)
smu.reset() (on page 8-126)

smu.measure.autorangelow

This attribute sets the lowest measurement range that is used when the instrument selects the measurement range automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: 1e-8 Voltage: 20 Resistance: 20

Usage

```
lowRange = smu.measure.autorangerangelow
smu.measure.autorangerangelow = lowRange
```

lowRange	The lower limit: • Current: 1e-8 to 1 amps • Voltage: 0.02 to 200 volts • Resistance: 2 to 2.0e8 ohms

Details

You can use this command when automatic range selection is enabled. It prevents the instrument from selecting a range that is below this limit. Because the lowest ranges generally require longer settling times, setting the low limit that is appropriate for your application but above the lowest possible range can make measurements require less settling time.

The lower limit must be less than the upper limit.

While you can send any value when you send this command, the instrument select the next highest range value. For example, if you send 15 for the lowest volt range, the instrument will be set to the 20 V range as the low limit.

If the lower limit is equal to the upper limit, automatic range setting is effectively disabled.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.autorange = smu.ON`
`smu.measure.autorangelow = 2`
	Sets the low range for voltage measurements to 2 V.

Also see

Ranges (on page 2-101)
smu.measure.autorange (on page 8-86)

smu.measure.autozero.enable

This attribute enables or disables of the internal reference measurements (autozero) of the source-measure unit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
azMode = smu.measure.autozero.enable
smu.measure.autozero.enable = azMode
```

azMode	The status of autozero; set to one of the following values: • Disable autozero: `smu.OFF` • Enable autozero: `smu.ON`

Details

The analog-to-digital converter (ADC) uses a ratiometric A/D conversion technique. To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The Model 2450 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

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 disable autozero. If autozero is disabled, to prevent inaccurate readings, you can use `smu.measure.autozero.once()` before a test sequence to force a one-time refresh of the reference measurements.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.autozero.enable = smu.OFF`	Set autozero off for voltage measurements.

Also see

smu.measure.autozero.once() (on page 8-90)
smu.measure.nplc (on page 8-116)

smu.measure.autozero.once()

This function causes the instrument to autozero once.

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

Usage

```
smu.measure.autozero.once()
```

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

`smu.measure.autozero.once()`	Forces a refresh of the reference and zero measurements.

Also see

smu.measure.autozero.enable (on page 8-89)

smu.measure.configlist.catalog()

This function returns the name of one measure configuration list stored on the instrument.

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

Usage

```
smu.measure.configlist.catalog()
```

Details

You can use this command to see the names of measure configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no measure configuration lists have been created for the instrument.

Commands are stored in the instrument are stored in runtime memory.

Example

`print(smu.measure.configlist.catalog())`	Request the name of one measure configuration list that is stored in the instrument. Send the command again until it returns `nil` to get all stored lists.
`print(smu.measure.configlist.catalog())`	If there are two configuration lists on the instrument. Example output: `testMeasList`
`print(smu.measure.configlist.catalog())`	`myMeasList`
`print(smu.measure.configlist.catalog())`	`nil`

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.create() (on page 8-92)

smu.measure.configlist.create()

This command creates an empty measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script	

Usage

```
smu.measure.configlist.create(listName)
```

listName	The name of the configuration list

Details

This command creates an empty configuration list. To add configuration points to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Example

```
smu.measure.configlist.create("MyMeasList")
```

Create a measure configuration list named `MyMeasList`.

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.catalog() (on page 8-91)
smu.measure.configlist.delete() (on page 8-93)
smu.measure.configlist.query() (on page 8-94)
smu.measure.configlist.recall() (on page 8-95)
smu.measure.configlist.size() (on page 8-96)
smu.measure.configlist.store() (on page 8-97)

smu.measure.configlist.delete()

This command deletes a measure configuration list.

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

Usage

```
smu.measure.configlist.delete(listName)
smu.measure.configlist.delete(listName, point)
```

listName	The name of the configuration list
point	The number that identifies a specific configuration point on the configuration list to delete

Details

Deletes a configuration list. If the point parameter is not specified, the entire configuration list is deleted. If the point parameter is specified, only the specified configuration point in the list is deleted.

Example

`smu.measure.configlist.delete("myMeasList")`	Delete a measure configuration list named `myMeasList`.
`smu.measure.configlist.delete("myMeasList", 2)`	Delete configuration point 2 from the measure configuration list named `myMeasList`.

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.create() (on page 8-92)

smu.measure.configlist.query()

This function returns a list of TSP commands that represent the parameters that are stored in the specified configuration point.

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

Usage

```
smu.measure.configlist.query(listName, point)
```

listName	A string that represents the name of a measure configuration list
point	A specific configuration point in the configuration list query

Details

This command can only return data for one configuration point. To get data for additional configuration points, send the command specifying different configuration points each time.

For additional information about the attributes this command returns, see Instrument settings stored in a measure configuration list (on page 3-36).

Example

```
print(smu.measure.configlist.query("testMeasList", 2))
```

Returns the TSP commands that represent the settings in configuration point 2.

Example output:

```
smu.source.func = smu.FUNC_DC_VOLTAGE, smu.source.autorange = smu.OFF,
smu.source.range = 2.000000e+01, smu.source.offmode = smu.OFFMODE_NORMAL,
smu.source.level = 4.000000e+00, smu.source.autodelay = smu.ON,
smu.source.readback = smu.ON, smu.source.highc = smu.OFF, smu.source.ilimit.level
= 1.050000e-04, smu.source.protect.level = smu.PROTECT_NONE,
smu.source.userdelay[1] = 0.000000e+00, smu.source.userdelay[2] = 0.000000e+00,
smu.source.userdelay[3] = 0.000000e+00, smu.source.userdelay[4] = 0.000000e+00,
smu.source.userdelay[5] = 0.000000e+00, smu.source.output = smu.OFF,
```

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.create() (on page 8-92)

smu.measure.configlist.recall()

This command recalls a specific configuration point in a specific measure configuration list.

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

Usage

```
smu.measure.configlist.recall(listName, point)
```

listName	A string that represents the name of a measure configuration list
point	A specific configuration point in the configuration list

Details

Use this command to recall the settings stored in a specific configuration point on a specific configuration list. If you do not specify a point when you send the command, it recalls the settings stored in the first configuration point on the specified configuration list.

This command can only return data for one configuration point. To get data for additional configuration points, send the command specifying different configuration points each time.

For additional information about the attributes this command returns, see Instrument settings stored in a measure configuration list (on page 3-36).

Example

`smu.measure.configlist.recall("MyMeasList")`	Since a point was not specified, this command recalls configuration point 1 from a configuration list named `MyMeasList`.
`smu.measure.configlist.recall("MyMeasList", 5)`	Recalls configuration point 5 in a configuration list named MyMeasList.

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.create() (on page 8-92)

smu.measure.configlist.size()

This function returns the size (number of configuration points) of a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script	

Usage

```
smu.measure.configlist.size(listName)
```

listName	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration points) of a measure configuration list.

The size of the list is equal to the number of configuration points in a configuration list.

Example

`print(smu.measure.configlist.size("testMeasList"))`	Print the size of a measure configuration list named `testMeasList`. Example output: `1`

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.create() (on page 8-92)

smu.measure.configlist.store()

This function stores the active measure settings into the named configuration list for measuring.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script	

Usage

```
smu.measure.configlist.store(listName)
smu.measure.configlist.store(listName, point)
```

listName	The name of the measure configuration list
point	A specific configuration point in the configuration list

Details

Use this command to store the active source settings to a configuration point in a configuration list. If the `point` parameter is not provided, the configuration point will append to the end of the list.

Refer to Instrument settings stored in a measure configuration list (on page 3-36) for a complete list of measure settings that the instrument stores in a measure configuration list.

Example

`smu.measure.configlist.store("MyConfigList")`	To store the instrument's active settings to the measure configuration list `MyConfigList`. Settings are saved at the end of the list since no point parameter is specified.
`smu.measure.configlist.store("MyConfigList", 5)`	To store the instrument's active settings to configuration point 5 on the measure configuration list `MyConfigList`.

Also see

Configuration lists (on page 3-33)
smu.measure.configlist.create() (on page 8-92)

smu.measure.count

This attribute sets the number of measurements to be performed when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	1

Usage

```
count = smu.measure.count
smu.measure.count = count
```

count	Number of readings (1 to 300,000)

Details

This command sets the number of measurements that are taken when a measurement is requested. This command does not affect the trigger model.

NOTE

To get better feedback from the instrument, use the Simple Loop trigger model template instead of using the count command.

Example 1

```
reset()

--Set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1
smu.measure.count = 200

--Set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

--Turn on output and initiate readings
smu.source.output = smu.ON
smu.measure.read(defbuffer1)

--Parse index and data into three columns
print("Rdg #", "Times", "Current (A)")
for i=1, defbuffer1.n do
      print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

--Discharge the capacitor to 0 V and turn off the output
smu.source.level=0
delay(2)
smu.source.output=smu.OFF
```

This example uses `smu.measure.count` to do a capacitor test. This outputs 200 readings that are similar to the following output:

```
Rdg # Times  Current (A)
1   0  8.5718931952528e-11
2   0.151875  1.6215984111057e-10
3   0.303727  1.5521139928865e-10
. . .
198 29.91579194     1.5521250951167e-10
199 30.067648716    1.4131290582142e-10
200 30.219497716    1.5521067764368e-10
```

Example 2

```
reset()

--set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1

--set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

--turn on output and initiate readings
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

--Parse index and data into three columns
print("Rdg #", "Times", "Current (A)")
for i=1, defbuffer1.n do
      print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

--Discharge the capacitor to 0 V and turn off the output
smu.source.level=0
delay(2)
smu.source.output=smu.OFF
```

This example uses the Simple Loop trigger model template to do a capacitor test. This also outputs 200 readings that are similar to the following output:

```
Rdg # Time (s)      Current (A)
1   0  8.5718931952528e-11
2   0.151875  1.6215984111057e-10
3   0.303727  1.5521139928865e-10
. . .
198 29.91579194     1.5521250951167e-10
199 30.067648716    1.4131290582142e-10
200 30.219497716    1.5521067764368e-10
```

Also see

smu.measure.read() (on page 8-119)
trigger.model.load() — Simple Loop (on page 8-191)

smu.measure.displaydigits

This attribute determines the number of digits that are displayed for measurements on the front panel for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.DIGITS_5_5

Usage

```
digits = smu.measure.displaydigits
smu.measure.displaydigits = digits
```

digits	6½ display digits: `smu.DIGITS_6_5` 5½ display digits: `smu.DIGITS_5_5` 4½ display digits: `smu.DIGITS_4_5` 3½ display digits: `smu.DIGITS_3_5`

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

To change the number of digits returned in a remote command reading, use `format.asciiprecision`.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.displaydigits = smu.DIGITS_6_5`	Set the measurement function to voltage with a front-panel display resolution of 6.5.

Also see

format.asciiprecision (on page 8-62)

smu.measure.filter.count

This attribute sets the number of measurements that are averaged when filtering is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	10

Usage

```
filterCount = smu.measure.filter.count
smu.measure.filter.count = filterCount
```

filterCount	The number of readings required for each filtered measurement (1 to 100)

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. The larger the filter count, the more filtering that is performed.

This command is set for the selected function.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.filter.count = 10`
`smu.measure.filter.type = smu.FILTER_MOVING_AVG`
`smu.measure.filter.enable = smu.ON`
	Set the measurement function to current.
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 4-22)
smu.measure.filter.enable (on page 8-101)
smu.measure.filter.type (on page 8-102)

smu.measure.filter.enable

This attribute enables or disables the averaging filter for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
filterState = smu.measure.filter.enable
smu.measure.filter.enable = filterState
```

filterState	The filter status: • Disable the filter: `smu.OFF` or `0` • Enable the filter: `smu.ON` or `1`

Details

This command enables or disables the averaging filter. When this is enabled, the measurements for the selected measurement function are averaged as set by the filter count and filter type.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.filter.count = 10`
`smu.measure.filter.type = smu.FILTER_MOVING_AVG`
`smu.measure.filter.enable = smu.ON`
	Set the measurement function to current.
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 4-22)
smu.measure.filter.count (on page 8-100)
smu.measure.filter.type (on page 8-102)

smu.measure.filter.type

This attribute sets the type of averaging filter that is used for the selected measurement function when the measurement filter is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.FILTER_REPEAT_AVG

Usage

```
filterType = smu.measure.filter.type
smu.measure.filter.type = filterType
```

filterType	The filter type to use when filtering is enabled; set to one of the following values: • Moving average filter: `smu.FILTER_MOVING_AVG` • Repeat filter: `smu.FILTER_REPEAT_AVG`

Details

You can select one of two types of averaging filters: repeating average or moving average.

When the repeating average filter is selected, a set of measurements are made, which are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack has to be completely filled before an averaged sample can be produced.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

Note that when the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results, but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.filter.count = 10`
`smu.measure.filter.type = smu.FILTER_MOVING_AVG`
`smu.measure.filter.enable = smu.ON`
	Set the measurement function to current.
	Set the averaging filter type to moving average, with a filter count of 10.
	Enable the averaging filter.

Also see

Filtering measurement data (on page 4-22)
smu.measure.filter.count (on page 8-100)
smu.measure.filter.enable (on page 8-101)

smu.measure.func

This attribute selects which type of measurement is active: current, voltage, or resistance.

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

Usage

```
mFunction = smu.measure.func
smu.measure.func = mFunction
```

mFunction	The measurement function: • Voltage measurement: `smu.FUNC_DC_VOLTAGE` • Current measurement: `smu.FUNC_DC_CURRENT` • Ohms measurement: `smu.FUNC_RESISTANCE`

Details

Set this command to the type of measurement you want to make.

Reading this attribute returns the function that is presently active.

When you select a function, settings for other commands that are related to the function become active. For example, assume that:

- You had selected resistance previously and set the math function set to reciprocal.
- You changed to the voltage function and set the math function to percent.

If you return to the resistance function, the math function returns to reciprocal. If you then switch from the resistance function to the voltage function, the math function returns to percent. All attributes that begin with `smu.measure.` will change settings based on the selected function unless otherwise indicated in the command description.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format = smu.MATH_PERCENT`
`smu.measure.math.enable = smu.ON`
`smu.measure.func = smu.FUNC_RESISTANCE`
`smu.measure.math.format = smu.MATH_RECIPROCAL`
`smu.measure.math.enable = smu.ON`
`print(smu.measure.math.format)`
`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`print(smu.measure.math.format)`
	Sets the instrument to measure voltage and set the math format to percent and enable the math functions.
	Set the instrument to measure resistance and set the math format to reciprocal and enable the math functions.
	Print the math format while the resistance measurement function is selected. The output is:
	`smu.MATH_RECIPROCAL`
	Change the function to voltage. Print the math format. The output is:
	`smu.MATH_PERCENT`

Also see

smu.source.level (on page 8-135)
smu.source.output (on page 8-140)

smu.measure.limit[Y].autoclear

This command indicates if limit *Y* should be cleared automatically or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.ON

Usage

```
value = smu.measure.limit[Y].autoclear
smu.measure.limit[Y].autoclear = value
```

value	The auto clear setting: • Enable: `smu.ON` or `1` • Disable: `smu.OFF` or `0`
Y	Limit number: `1` or `2`

Details

When this command sets autoclear to on for a measurement function, if a measurement fails limit, but the next measurement passes limit, the failed limit condition is cleared. Therefore, if you are making a series of measurements, the instrument uses last measurement limit for the fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the auto clear setting to off. When this set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The auto clear setting affects both the high and low limits of *Y*.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT` `smu.measure.limit[1].autoclear = smu.ON`	Turns on autoclear for limit 1 when measuring DC current.

Also see

smu.measure.limit[Y].clear() (on page 8-105)

smu.measure.limit[Y].clear()

This function clears the results of the limit test for the selected measurement function.

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

Usage

```
smu.measure.limit[Y].clear()
```

Y	1 or 2 for limit number

Details

Use this command to clear the test results of limit *Y* when the limit auto clear command is disabled. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, enable the auto clear command.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT` `smu.measure.limit[2].clear()`	Clears the test result for the high and low limit 2 for current measurements.

Also see

smu.measure.limit[Y].autoclear (on page 8-104)

smu.measure.limit[Y].enable

This attribute enables or disables a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
state = smu.measure.limit[Y].enable
smu.measure.limit[Y].enable = state
```

state	Disable the test: `smu.OFF` or `0` Enable the test: `smu.ON` or `1`
Y	The number of the test: `1` or `2`

Details

This command enables or disables a limit test for the selected measurement function.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.limit[1].enable = smu.ON`	Enable testing for limit 1 when measuring voltage.

Also see

smu.measure.limit[Y].autoclear (on page 8-104)
smu.measure.limit[Y].clear() (on page 8-105)
smu.measure.limit[Y].fail (on page 8-107)
smu.measure.limit[Y].high.value (on page 8-109)
smu.measure.limit[Y].low.value (on page 8-110)

smu.measure.limit[Y].fail

This attribute queries the results of a limit test.

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

Usage

result = smu.measure.limit[*Y*].fail

result	The results of the limit test for limit *Y*: • smu.FAIL_NONE: Test passed; measurement under or equal to the high limit • smu.FAIL_HIGH: Test failed; measurement exceeded high limit • smu.FAIL_LOW: Test failed; measurement exceeded low limit • smu.FAIL_BOTH: Test failed; measurement exceeded both limits
Y	Limit number: 1 or 2

Details

These commands query the result of a limit test for the selected measurement function.

The response message indicates if the limit test has passed or how it failed.

Reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command.

Note that if you are making a series of measurements and auto clear enabled for a limit, the last measurement limit dictates the fail indication for the limit. If autoclear is disabled, you can take a series of readings and read fails to see if any of one of the readings failed.

To use this attribute, you must set the limit state to enable.

If the readings are stored in a reading buffer, the values are associated with *bufferVar*.statuses for the readings.

Example

This example enables limits 1 and 2 for voltage, measurements. Limit 1 is checking for readings to be between 3 and 5 volts, while limit 2 is checking for the readings to be between 1 and 7 volts. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
reset()
smu.source.func = smu.FUNC_DC_CURRENT  -- set the instrument source current
smu.measure.func = smu.FUNC_DC_VOLTAGE -- set the instrument to measure voltage
smu.measure.range = 10                   -- set the range to 10 volts
smu.measure.nplc = 0.1                   -- set the nplc to 0.1
smu.measure.limit[1].autoclear = smu.OFF   -- disable auto clearing for limit 1
smu.measure.limit[1].high.value = 5      -- set high limit on 1 to fail if
                                            -- reading exceeds 5 volts
smu.measure.limit[1].low.value = 3     -- set low limit on 1 to fail if reading
                                            -- is less than 3 volts
smu.measure.limit[1].enable = smu.ON   -- enable limit 1 checking for voltage
                                       -- measurements
smu.measure.limit[2].autoclear = smu.OFF   -- disable auto clearing for limit 2
smu.measure.limit[2].high.value = 7      -- set high limit on 2 to fail if
                                          -- reading exceeds 7 volts
smu.measure.limit[2].low.value = 1   -- set low limit on 2 to fail if reading
                                        -- is less than 1 volts
smu.measure.limit[2].enable = smu.ON     -- enable limit 2 checking for voltage
                                            -- measurements
smu.measure.count = 50                -- set the measure count to 50
LimitBuffer = buffer.make(100)        -- create a reading buffer that can store
                                         -- 100 readings
smu.measure.read(LimitBuffer)         -- make 50 readings and store them in
                                         -- LimitBuffer
                                      -- then check if any of the 50 readings
                                         -- were outside of the limits
print("limit 1 results = " .. smu.measure.limit[1].fail)
print("limit 2 results = " .. smu.measure.limit[2].fail)

smu.measure.limit[1].clear()                 -- clear limit 1 conditions
smu.measure.limit[2].clear()                 -- clear limit 2 conditions
```

Sample output that shows all readings are within limit values (all readings between 3 and 5 volts):

```
limit 1 results = smu.FAIL_NONE
limit 2 results = smu.FAIL_NONE
```

Sample output showing at least one reading failed limit 1 high values (a 6 volt reading would cause this condition or a reading greater than 5 but less than 7.):

```
limit 1 results = smu.FAIL_HIGH
limit 2 results = smu.FAIL_NONE
```

Sample output showing at least one reading failed limit 1 and 2 low values (a 0.5 volt reading would cause this condition or a reading less than 1):

```
limit 1 results = smu.FAIL_LOW
limit 2 results = smu.FAIL_LOW
```

Also see

Limit testing and binning (on page 3-68)
smu.measure.limit[Y].enable (on page 8-106)

smu.measure.limit[Y].high.value

This command specifies the upper limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1.000000E+00

Usage

```
highLimit = smu.measure.limit[Y].high.value
smu.measure.limit[Y].high.value = highLimit
```

highLimit	The high value; range is –999999000000 to +999999000000
Y	Limit number: 1 or 2

Details

This command sets the high limits for the limit tests for the selected measurement function. When limit testing is enabled for this limit, the instrument generates a fail indication when the measurement value is more than this value.

Example

See the example in smu.measure.limit[Y].fail (on page 8-107).

Also see

smu.measure.limit[Y].enable (on page 8-106)

smu.measure.limit[Y].low.value

This command specifies the lower limit for limit tests.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	–1.000000E+00

Usage

```
value = smu.measure.limit[Y].low.value
smu.measure.limit[Y].low.value = value
```

value	The high value; range is –999999900000 to +999999900000
Y	Limit number: `1` or `2`

Details

This command sets the lower limits for the limit tests for the selected measurement function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

Example

See the example in smu.measure.limit[Y].fail (on page 8-107).

Also see

smu.measure.limit[Y].enable (on page 8-106)

smu.measure.math.enable

This command enables or disables math operation on measurements for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
value = smu.measure.math.enable
smu.measure.math.enable = value
```

value	The math enable setting: • Enable: `smu.ON` or `1` • Disable: `smu.OFF` or `0`

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format =`
`   smu.MATH_PERCENT`
`smu.measure.math.enable = smu.ON`
	When voltage measurements are made, the math format is enabled and set to percent.

Also see

Calculations that you can apply to measurements (on page 3-6)
smu.measure.math.format (on page 8-112)

smu.measure.math.format

This attribute specifies which math operation is performed on measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.MATH_PERCENT

Usage

```
operation = smu.math.format
smu.math.format = operation
```

operation	Math operation to be performed on measurements: • y = mx+b: `smu.MATH_MXB` • Percent: `smu.MATH_PERCENT` • Reciprocal: `smu.MATH_RECIPROCAL`

Details

This specifies which math operation is performed on measurements for the selected measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Specify a constant that is applied to the measurement and display measurements as percentages.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is 1/X, where `X` is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.math.format = smu.MATH_RECIPROCAL`
`smu.measure.math.enable = smu.ON`
	Enables the reciprocal math operation on voltage measurements.

Also see

Calculations that you can apply to measurements (on page 3-6)
smu.measure.math.enable (on page 8-111)
smu.measure.math.mxb.bfactor (on page 8-113)
smu.measure.math.mxb.mfactor (on page 8-114)
smu.measure.math.percent (on page 8-115)

smu.measure.math.mxb.bfactor

This attribute specifies the offset for the y = mx + b operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
value = smu.measure.math.mxb.bfactor
smu.measure.math.mxb.bfactor = value
```

value	The offset for the y = mx + b operation; the valid range is −1e12 to +1e12

Details

This attribute specifies the offset (b) for an mx + b operation.

The mx+b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$y = mx + b$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`	Set the measurement function to voltage.
`smu.measure.math.format = smu.MATH_MXB` `smu.measure.math.mxb.mfactor = 0.80`	Set the scale factor for the mx +b operation to 0.80.
`smu.measure.math.mxb.bfactor = 50`	Set the offset factor to 50.
`smu.measure.math.enable = smu.ON`	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 3-6)
smu.measure.math.enable (on page 8-111)
smu.measure.math.mxb.mfactor (on page 8-114)

smu.measure.math.mxb.mfactor

This attribute specifies the scale factor for the y = mx + b math operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
value = smu.measure.math.mxb.mfactor
smu.measure.math.mxb.mfactor = value
```

value	The scale factor; the valid range is –4294967295 to +4294967295

Details

This command sets the scale factor (m) for an mx + b operation for the selected measurement function.

The mx+b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$y = mx + b$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`	Set the measurement function to voltage.
`smu.measure.math.format = smu.MATH_MXB` `smu.measure.math.mxb.mfactor = 0.80`	Set the scale factor for the mx +b operation to 0.80.
`smu.measure.math.mxb.bfactor = 50`	Set the offset factor to 50.
`smu.measure.math.enable = smu.ON`	Enable the math function.

Also see

Calculations that you can apply to measurements (on page 3-6)
smu.measure.math.enable (on page 8-111)
smu.measure.math.mxb.bfactor (on page 8-113)

smu.measure.math.percent

This attribute specifies the constant to use when math operation is set to percent.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1

Usage

```
value = smu.measure.math.percent
smu.measure.math.percent = value
```

value	The constant when the math operation is set to percent; the range is –4294967295 to +4294967295

Details

This is the constant that is used when the math operation is set to percent for the selected measurement function.

The percent math function displays measurements as percent deviation from a specified constant. The percent calculation is:

$$Percent = \left(\frac{input - reference}{reference}\right) \times 100\%$$

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`	Set the measurement function to voltage.
`smu.measure.math.format = smu.MATH_PERCENT`	Set the math operations to percent.
`smu.measure.math.percent = 50`	Set the percentage value to 50 for voltage measurements.
`smu.measure.math.enable = smu.ON`	Enable math operations.

Also see

Calculations that you can apply to measurements (on page 3-6)
smu.measure.math.enable (on page 8-111)
smu.measure.math.format (on page 8-112)

smu.measure.nplc

This command sets the time that the input signal is measured for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	1.0

Usage

```
nplc = smu.measure.nplc
smu.measure.nplc = nplc
```

nplc	The number of power line cycles; set from 0.01 to 10

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified in parameters that are based on the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50).

This command is set for the measurement of specific functions (current, resistance, or voltage).

The shortest amount of time (0.01 PLC) results in the fastest reading rate, but increases the reading noise and decreases the number of usable digits.

The longest amount of time (10 PLC) provides the lowest reading noise and more usable digits, but has the slowest reading rate.

Settings between the fastest and slowest number of PLCs are a compromise between speed and noise.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.nplc = 0.5`	Sets the measurement time to 0.0083 (0.5/60) seconds.

Also see

Using NPLCs to adjust speed and accuracy (on page 4-8)

smu.measure.offsetcompensation

This command enables or disables offset compensation for resistance measurements.

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

Usage

```
state = smu.measure.offsetcompensation
smu.measure.offsetcompensation = state
```

state	Enable offset compensation: `smu.ON` Disable offset compensation: `smu.OFF`

Details

The voltage offsets because of the presence of thermal EMFs (V_{EMF}) can adversely affect resistance measurement accuracy. To overcome these offset voltages, you can use offset-compensated ohms.

This feature is only available for resistance measurements.

Example

```
smu.measure.func = smu.FUNC_RESISTANCE
smu.measure.sense = smu.SENSE_4WIRE
smu.measure.offsetcompensation = smu.ON
smu.source.output = smu.ON
print(smu.measure.read())
smu.source.output = smu.OFF
```

Sets the measurement function to resistance. Set the instrument for 4-wire measurements and turn offset compensation on. Turn on the source, make a measurement, and turn the source off. An example output is:

```
81592000
```

Also see

None

smu.measure.range

This attribute contains the positive full-scale value of the measurement range for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: 1e-06 Resistance: 200000 Voltage: 20

Usage

```
rangeValue = smu.measure.range
smu.measure.range = rangeValue
```

rangeValue	Set to the maximum expected value to be measured

Details

When you assign a range value, the instrument is set on a fixed range that is large enough to measure the assigned value. The instrument selects the best range for measuring the maximum expected value.

This command is primarily intended to eliminate the time that is required by the instrument to select an automatic range.

Note that when you select a fixed range, an overrange condition can occur.

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is the same as the source range, regardless of measurement range setting. However, the setting for the measure range is retained, and when the source function is changed (for example, from sourcing voltage to sourcing current), the retained measurement range is used.

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the measurement function.

When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using. If you change the range while the output is off, the instrument does not update the hardware settings, but if you read the range setting, the return is the setting that will be used when the output is turned on. If you set a range while the output is on, the new setting takes effect immediately.

Example

`smu.source.func = smu.FUNC_DC_CURRENT`
`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.range = 0.5`
	Select the measurement function to be voltage.
	Instrument selects the 2 V measurement range.

Also see

Ranges (on page 2-101)
smu.measure.autorange (on page 8-86)

smu.measure.read()

This function makes a measurement and returns the reading.

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

Usage

```
reading = smu.measure.read()
reading = smu.measure.read(bufferName)
```

reading	The last reading of the measurement process
bufferName	The name of the buffer where the reading is stored; if nothing is specified, the reading is stored in `defbuffer1`

Details

This makes a measurement using the present function setting, stores the reading in a reading buffer, and returns the last reading.

The `smu.measure.count` attribute determines how many measurements are performed. You can also use the trigger model Simple Loop.

When a reading buffer is used with a command or action that involves taking multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

NOTE

To make a power reading, use the `smu.measure.unit` command and set the units to `smu.UNIT_WATT` for the voltage or current measurement function.

Example

`voltMeasBuffer = buffer.make(10000)`
`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`print(smu.measure.read(voltMeasBuffer))`
	Create a buffer named `voltMeasBuffer`. Set the instrument to measure voltage.
	Make reading that is stored in the `voltMeasBuffer` and return a measurement.

Also see

buffer.make() (on page 8-11)
Reading buffers (on page 3-10)
smu.measure.count (on page 8-97)
smu.measure.unit (on page 8-125)
trigger.model.load() — Simple Loop (on page 8-191)

smu.measure.readwithtime()

This function returns the last actual measurement and time information in UTC format without using the trigger model.

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

Usage

```
reading, seconds, fractional = smu.measure.readwithtime()
smu.measure.readwithtime(bufferName)
```

reading	The last reading of the measurement process
seconds	Seconds in UTC format
fractional	Fractional seconds
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

When a reading buffer is used with a command or action that involves taking multiple readings, all readings are available in the reading buffer. However, this command only returns the last reading and time information.

The `smu.measure.count` attribute or trigger model Simple Loop determines how many measurements are performed. When you use a buffer, it also determines if the reading buffer has enough room to store the requested readings.

Example

`print(smu.measure.readwithtime())`	Print the last measurement and time information in UTC format, which will look similar to: `-1.405293589829e-11  1400904629  0.1950935`

Also see

smu.measure.count (on page 8-97)
trigger.model.load() — Simple Loop (on page 8-191)

smu.measure.rel.acquire()

This function acquires an internal measurement to store as the relative offset value.

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

Usage

```
relativeValue = smu.measure.rel.acquire()
```

relativeValue	The internal measurement acquired for the relative offset value

Details

This command triggers the Model 2450 to make a new measurement for the selected function. This measurement is then stored as the new relative offset level setting.

When you send this command, the measurement is made without applying any math, limit test, or filter settings, even if they are set. It is a reading as if these settings are disabled.

After executing this command, you can use the `smu.measure.rel.level` attribute to see the last relative level value that was acquired or that was set.

If an error occurs during the measurement, `nil` is returned and the relative offset level remains at the last valid setting.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`rel_value = smu.measure.rel.acquire()`
`smu.measure.rel.enable = smu.ON`
	Acquires a relative offset level value for voltage measurements and turns the relative offset feature on.

Also see

smu.measure.rel.enable (on page 8-121)
smu.measure.rel.level (on page 8-122)

smu.measure.rel.enable

This attribute enables or disables the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.OFF

Usage

```
relEnable = smu.measure.rel.enable
smu.measure.rel.enable = relEnable
```

relEnable	Relative measurement control: • Disable relative measurements: `smu.OFF` or 0 • Enable relative measurements: `smu.ON` or `1`

Details

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value calculated when you acquire the relative offset value.

Each returned measured relative reading is the result of the following calculation:

Display value = Actual measured value - Relative offset value

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`rel_value = smu.measure.rel.acquire()`
`smu.measure.rel.enable = smu.ON`
	Acquires a relative offset level value for voltage measurements and turns the relative offset feature on.

Also see

Relative offset (on page 3-1)
smu.measure.rel.acquire() (on page 8-121)
smu.measure.rel.level (on page 8-122)

smu.measure.rel.level

This attribute contains the relative offset value for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
relValue = smu.measure.rel.level
smu.measure.rel.level = relValue
```

relValue	Relative offset value for measurements: • Current: –1 to 1 • Resistance: –200000000 to 200000000 • Voltage: –200 to 200

Details

This command specifies the relative offset value that is used for measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

Example

`smu.measure.func = smu.FUNC_DC_CURRENT`
`smu.measure.rel.level = smu.measure.read()`
`smu.measure.rel.enable = smu.ON`
	Sets the measurement function to current, performs a current measurement, uses it as the relative offset value, and enables the relative offset for current measurements.

Also see

Relative offset (on page 3-1)
smu.measure.rel.acquire() (on page 8-121)
smu.measure.rel.enable (on page 8-121)

smu.measure.sense

This command selects local (2-wire) or remote (4-wire) sensing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	smu.SENSE_2WIRE

Usage

```
sensing = smu.measure.sense
smu.measure.sense = sensing
```

sensing	Two-wire sensing: `smu.SENSE_2WIRE` Four-wire sensing: `smu.SENSE_4WIRE`

Details

This command determines if 2-wire (local) or 4-wire (remote) sensing is used.

When you use 4-wire sensing, voltages are measured at the device under test (DUT). For the source voltage, if the sensed voltage is lower than the programmed amplitude, the voltage source increases the voltage until the sensed voltage is the same as the programmed amplitude. This compensates for IR drop in the output test leads.

Using 4-wire sensing with voltage measurements eliminates any voltage drops that may be in the test leads between the Model 2450 and the DUT.

When you are using 2-wire sensing, voltage is measured at the output connectors.

When you are measuring resistance, you can enable 4-wire sensing to make 4-wire resistance measurements.

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

Example

`smu.measure.func = smu.FUNC_RESISTANCE` `smu.measure.sense = smu.SENSE_4WIRE`	Set the measurement function to resistance. Set the sense to 4-wire remote.

Also see

Two-wire local sense connections (on page 2-75)
Four-wire remote sense connections (on page 2-76)

smu.measure.terminals

This command determines which set of input and output terminals the instrument is using.

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

Usage

```
terminals = smu.measure.terminals
smu.measure.terminals = terminals
```

terminals	Use the front-panel input and output terminals: `smu.TERMINALS_FRONT` Use the rear-panel input and output terminals: `smu.TERMINALS_REAR`

Details

This command selects which set of input and output terminals the instrument uses. You can select front panel or rear panel terminals.

If the output is turned on when you change from one set of terminals to the other, the output is turned off.

Example

`smu.measure.terminals = smu.TERMINALS_FRONT`	Use the front-panel terminals for measurements.

Also see

None

smu.measure.unit

This attribute describes the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	Current: smu.UNIT_AMP Voltage: smu.UNIT_VOLT Resistance: smu.UNIT_OHM

Usage

```
unitOfMeasure = smu.measure.unit
smu.measure.unit = unitOfMeasure
```

unitOfMeasure	The units of measure to be displayed for the measurement: • Volts: `smu.UNIT_VOLT` (only available for voltage measurements) • Resistance: `smu.UNIT_OHM` (only available for resistance measurements) • Current: `smu.UNIT_AMP` (only available for current measurements) • Power: `smu.UNIT_WATT` (only available for voltage or current measurements)

Details

The change in measurement units is displayed when the next measurement occurs.

Example

`smu.measure.func = smu.FUNC_DC_VOLTAGE` `smu.measure.unit = smu.UNIT_WATT`	Changes the front-panel display and buffer readings for voltage measurements to be displayed as power readings in watts.

Also see

smu.measure.func (on page 8-103)

smu.measure.userdelay[N]

This attribute sets a user-defined delay that can be used in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
delayTime = smu.measure.userdelay[N]
smu.measure.userdelay[N] = delayTime
```

delayTime	The delay in seconds (0 to 10000)
N	The user delay to which this time applies (1 to 5)

Details

To use this commands in a trigger model, assign the delay to the dynamic delay block.

Example

```
smu.measure.userdelay[2] = .5
trigger.model.setblock(6, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M2)
```

Set user delay 2 to be 0.5 seconds. Sets trigger model block 6 to use the delay.

Also see

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 8-206)

smu.reset()

This function turns off the output and resets the commands that begin with `smu.` to their default settings.

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

Usage

```
smu.reset()
```

Details

This function turns off the output and resets the commands that begin with `smu.` to their default settings.

Example

`smu.reset()`	Turns off the output and resets the SMU commands to their default settings.

Also see

reset() (on page 8-82)

smu.source.autorange

This attribute determines if the range is selected manually or automatically for the selected source function or voltage source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
sourceAutorange = smu.source.autorange
smu.source.autorange = sourceAutorange
```

sourceAutorange	Disable automatic source range: `smu.OFF` or `0` Enables automatic source range: `smu.ON` or `1`

Details

This command indicates the state of the range for the selected source. When automatic source range is disabled, the source range is set manually.

When automatic source range is enabled, the instrument selects the range that is most appropriate for the value that is being sourced. The output level controls the range. If you read the range after the output level is set, the instrument returns the range that the instrument chose as appropriate for that source level.

If the source range is set to a specific value from the front panel or a remote command, the setting for automatic range is set to disabled.

Only available for current and voltage functions.

Example

`smu.source.func = smu.FUNC_DC_CURRENT` `smu.source.autorange = smu.ON`	Set the source function to current. Set the instrument to select the source range automatically.

Also see

smu.source.range (on page 8-143)

smu.source.autodelay

This attribute enables or disables the autodelay that occurs when the source is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Restore configuration Instrument reset Power cycle Source configuration list	smu.ON

Usage

```
state = smu.source.autodelay
smu.source.autodelay = state
```

state	Autodelay on: `smu.ON` Autodelay off: `smu.OFF`

Details

When auto delay is turned on, the actual delay that is set depends on the range.

When source autodelay is on, if you set a source delay, the autodelay is turned off.

Example

`smu.source.autodelay = smu.OFF`	Turn off auto delay when current is being sourced.

Also see

smu.source.delay (on page 8-133)

smu.source.configlist.catalog()

This function returns the name of one source configuration list.

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

Usage

```
smu.source.configlist.catalog()
```

Details

You can use this command to retrieve the names of source configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no source configuration lists have been created for the instrument.

Example

`print(smu.source.configlist.catalog())`	Request the name of one measure configuration list that is stored in the instrument. Send the command again until it returns `nil` to get all stored lists.

Also see

Configuration lists (on page 3-33)
smu.source.configlist.create() (on page 8-129)

smu.source.configlist.create()

This command creates an empty source configuration list for sourcing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list	Configuration script	

Usage

`smu.source.configlist.create(listName)`

listName	A string that represents the name of a source configuration list

Details

This command creates an empty configuration list. To add configuration points to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. If you want to save a configuration list through a power cycle, create a configuration script to save instrument settings, including any defined configuration lists.

Example

`reset()` `smu.source.configlist.create("MyScrList")`	Create a source configuration list named `MyScrLst`.
`print(smu.source.configlist.catalog())`	Print the name of one configuration list stored in volatile memory. Output: `MyScrList`
`print(smu.source.configlist.catalog())`	Print the name of one configuration list. Output: `nil` Nil indicates that no more configuration lists are stored.
`smu.source.configlist.store("MyScrList")`	Store a configuration point in `MyScrList`.
`smu.source.configlist.store("MyScrList")`	Store a configuration point in `MyScrList`.
`print(smu.source.configlist.size("MyScrList"))`	Print the number of configuration points in `MyScrList`. Output: `2`

Also see

Configuration lists (on page 3-33)
smu.source.configlist.store() (on page 8-132)

smu.source.configlist.delete()

This command deletes a source configuration list.

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

Usage

```
smu.source.configlist.delete(listName)
smu.source.configlist.delete(listName, point)
```

listName	A string that represents the name of a source configuration list
point	The number identifying a specific configuration point on the configuration list to delete

Details

Deletes a configuration list. If the point parameter is not specified, the entire configuration list is deleted. If the point parameter is specified, only the specified configuration point in the list is deleted.

Example

`smu.source.configlist.delete("mySourceList")`	Deletes a configuration list named `mySourceList`.
`smu.source.configlist.delete("mySourceList", 14)`	Deletes delete configuration point 14 in the source configuration list named `mySourceList`

Also see

Configuration lists (on page 3-33)
smu.source.configlist.create() (on page 8-129)

smu.source.configlist.query()

This function returns a list of TSP commands that represent the parameters that are stored in the specified configuration point.

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

Usage

```
smu.source.configlist.query(listName)
smu.source.configlist.query(listName, point)
smu.source.configlist.query(listName, point, fieldSeparator)
```

listName	A string that represents the name of a source configuration list
point	A specific configuration point in the configuration list; the default is the first point in the configuration list
fieldSeparator	String that represents the separator for the data; use one of the following: • Comma (default): `,` • Semicolon: `;` • New line: `\n`

Details

This command can only return data for one configuration point. To get data for additional configuration points, resend the command and specify different configuration points.

Refer to Instrument settings stored in a source configuration list (on page 3-38) for a complete list of source settings that the instrument stores in a source configuration list.

Example

`print(smu.source.configlist.query("MyScrList", 2))`	
Returns the TSP commands that represent the settings in configuration point 2.	

Also see

Configuration lists (on page 3-33)
smu.source.configlist.create() (on page 8-129)
Instrument settings stored in a source configuration list (on page 3-38)

smu.source.configlist.recall()

This command recalls a specific configuration point in a source configuration list.

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

Usage

```
smu.source.configlist.recall(name, point)
```

name	A string that represents the name of a source configuration list
point	A specific configuration point in the configuration list

Details

Use this command to recall the settings stored in a specific configuration point on a specific configuration list. If you do not specify a point when you send the command, it recalls the settings stored in the first configuration point on the specified configuration list.

Example

`smu.source.configlist.recall("MySourceList")`	Since a point was not specified, this command recalls configuration point 1 from a configuration list named `MySourceList`.
`smu.source.configlist.recall("MySourceList", 5)`	Recalls configuration point 5 in a configuration list named `MySourceList`.

Also see

Configuration lists (on page 3-33)
smu.source.configlist.create() (on page 8-129)

smu.source.configlist.size()

This command returns the number of configuration points in a source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list	Configuration script	

Usage

```
smu.source.configlist.size(listName)
smu.source.configlist.size(listName, point)
```

listName	A string that represents the name of a source configuration list
point	A specific configuration point in the configuration list

Details

The size of the list is equal to the number of configuration points in a configuration list.

Example

`print(smu.source.configlist.size("MyScrList"))`	Determine the number of configuration points in a source configuration list named `MyScrList`. Example output: `2`

Also see

Configuration lists (on page 3-33)
smu.source.configlist.create() (on page 8-129)

smu.source.configlist.store()

This function stores the active source settings into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Source configuration list		

Usage

```
smu.source.configlist.store(listName)
smu.source.configlist.store(listName, point)
```

listName	A string that represents the name of a source configuration list
point	A specific configuration point in the configuration list

Details

Use this command to store the active source settings to a configuration point in a configuration list. The `point` parameter indicates a specific configuration point in the list in which to store the active settings. If the point parameter is not provided, the configuration point is appended to the end of the list. If a configuration point already exists for the specified point, the new configuration overwrites the existing configuration point.

Refer to Instrument settings stored in a source configuration list (on page 3-38) for information about the settings this command stores.

Example

`smu.source.configlist.store("MyConfigList")`	Store the active settings of the instrument to the source configuration list `MyConfigList`. Settings are saved at the end of the list since no point parameter is specified.
`smu.source.configlist.store("MyConfigList", 5)`	Store the active settings of the instrument to configuration point 5 on the source configuration list `MyConfigList`.

Also see

None

smu.source.delay

This attribute contains the source delay.

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

Usage

```
sDelay = smu.source.delay
smu.source.delay = sDelay
```

sDelay	The length of the delay (0 to 4 s)

Details

This command sets a delay for the selected source function. This delay is in addition to normal settling times.

After the programmed source is turned on, this delay allows the source level to settle before a measurement is taken.

If source autodelay is on, if you set a specific delay, it is turned off.

If source autodelay is on, the manual source delay setting is not saved in the source configuration list.

Example

`smu.source.func = smu.FUNC_DC_VOLTAGE` `smu.source.delay = 3`	Set the function to voltage. Set a 3-second delay after the source is turned on before a measurement is taken.

Also see

Source delay (on page 2-111)
smu.source.autodelay (on page 8-128)

smu.source.func

This attribute contains the source function, which can be voltage or current.

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

Usage

```
sFunction = smu.source.func
smu.source.func = sFunction
```

sFunction	The source function; set to one of the following values: • Current source: `smu.FUNC_DC_CURRENT` • Voltage source: `smu.FUNC_DC_VOLTAGE`

Details

Setting this command configures the instrument as either a voltage source or a current source.
Reading this attribute returns the output setting of the source.

Example

`smu.source.func = smu.FUNC_DC_CURRENT`	Sets the source function of the instrument to be a current source.

Also see

smu.source.level (on page 8-135)
smu.source.output (on page 8-140)

smu.source.highc

This attribute enables or disables high capacitance feature.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFF

Usage

```
state = smu.source.highc
smu.source.highc = state
```

state	Turn high capacitance off: `smu.OFF` or `0` Turn high capacitance on: `smu.ON` or `1`

Details

When the instrument is measuring low current and is driving a capacitive load, you may see overshoot, ringing, and instability. You can enable the high capacitance feature to minimize these problems.

High-capacitance settings apply when operating using the 10 nA through the 100 mA current ranges. When operating using the 1 A range, the high-capacitance setting will not affect the instrument rise time or current measure settling time.

Example

`smu.source.highc = smu.ON`	Turn the high capacitance feature on.

Also see

High capacitance operation (on page 4-21)

smu.source.level

This attribute immediately selects a fixed amplitude for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

Usage

```
sourceLevel = smu.source.level
smu.source.level = sourceLevel
```

sourceLevel	Current: -1.05 A to 1.05 A Voltage: -210 V to 210 V

Details

This command sets the output level of the voltage or current source. If the output is on, the new level is sourced immediately.

The sign of the source level dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

If a manual source range is selected, the level cannot exceed the specified range. For example, if the voltage source is on the 2 V range (auto range is disabled), you cannot set the voltage source amplitude to 3 V. When auto range is selected, the amplitude can be set to any level.

Example

`smu.source.func = smu.FUNC_DC_VOLTAGE` `smu.source.level = 1`	Set the instrument to source voltage and set it to source 1 V.

Also see

smu.source.func (on page 8-134)
smu.source.output (on page 8-140)
smu.source.protect.level (on page 8-141)

smu.source.xlimit.level

This attribute selects the source limit for measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Restore configuration Instrument reset Power cycle Source configuration list	Current: 1.05E-04 Voltage: 21

Usage

```
value = smu.source.xlimit.level
smu.source.xlimit.level = value
```

value	The limit: • Current: -1 to 1 amps • Voltage: -200 to 200 volts
x	The function for which to set the limit: • Voltage: `v` • Current: `i`

Details

This command sets the source limit for measurements. The Model 2450 cannot source levels that exceed this limit.

The values that can be set for this command are limited by the setting for the overvoltage protection limit.

This value can also be limited by the measurement range. If a specific measurement range is set, the limit must be more than 0.1% of the measurement range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

If you change the source range to a level that is not appropriate for this limit, the source limit is changed to a limit that is appropriate to the range and a warning is generated.

Example

`smu.source.func = smu.FUNC_DC_VOLTAGE` `smu.source.ilimit.level = 1`	Set the source function to voltage with the current limit set to 1 A.

Also see

smu.source.protect.level (on page 8-141)
smu.source.xlimit.tripped (on page 8-138)

smu.source.xlimit.tripped

This attribute indicates if the source exceeded the limits that were set for the selected measurements.

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

Usage

```
state = smu.source.xlimit.tripped
```

state	Indicates if limit has been tripped: • Not tripped: `smu.OFF` • Tripped: `smu.ON`
x	The function whose limit was tripped: • `v`: voltage • `i`: current

Details

You can use this command check the limit state of the source.

If the limits were exceeded, the instrument clamps the source to keep the source within the set limits.

If you check the limit for the source that is not presently selected, `nil` is returned.

Example

`print(smu.source.vlimit.tripped)`	Check the state of the source limit for voltage. If the limit was exceeded, the output is: `smu.ON`

Also see

smu.source.xlimit.level (on page 8-137)

smu.source.offmode

This attribute defines the state of the source when the output is turned off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFFMODE_NORMAL

Usage

```
sourceOffMode = smu.source.offmode
smu.source.offmode = sourceOffMode
```

sourceOffMode	The output-off setting; set to one of the following values (see the **Details** below for specifics regarding each option): • `smu.OFFMODE_NORMAL` • `smu.OFFMODE_ZERO` • `smu.OFFMODE_HIGHZ` • `smu.OFFMODE_GUARD`

Details

Reading this attribute returns the output-off state of the source. Setting this attribute configures the output-off state.

When the Model 2450 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10 percent of the full scale of the present current range
- OUTPUT OFF is displayed in the Home page Source area
- The Source button on the Home page shows the output that will be sourced when the output is turned on again

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

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

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

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

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

- The measurement sense is changed to 2-wire sense
- The source function is set to voltage
- The source voltage is set to 0
- Set the range to the presently selected range (turn off autorange)
- Program the voltage DAC to zero, and the current DAC to full scale of the present current range.

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

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

- The measurement sense is changed to 2-wire sense
- The current source is selected and set to 0 A
- The voltage limit is set to 10% full scale of the present voltage range

Example

`smu.source.offmode = smu.OFFMODE_HIGHZ`	Sets the output-off state so that the instrument opens the output relay when the output is turned off.

Also see

Output-off state (on page 2-82)
smu.source.output (on page 8-140)

smu.source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.OFF

Usage

```
sourceOutput = smu.source.output
smu.source.output = sourceOutput
```

sourceOutput	Switch the source output off: `smu.OFF` or `0` Switch the source output on: `smu.ON` or `1`

Details

When the output is switched on, the instrument sources either voltage or current, as set by `smu.source.func`.

Example

`smu.source.output = smu.ON`	Switch the source output of the instrument to on.

Also see

Turning the Model 2450 output off (on page 2-6)
smu.source.offmode (on page 8-139)

smu.source.protect.level

This attribute sets the overvoltage protection setting of the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.PROTECT_NONE

Usage

```
limit = smu.source.protect.level
smu.source.protect.level = limit
```

limit	The limit value; set as `smu.PROTECT_x` where *x* is `2V`, `5V`, `10V`, `20V`, `40V`, `60V`, `80V`, `100V`, `120V`, `140V`, `160V`, `180V`, or `NONE`

Details

Overvoltage protection restricts the maximum voltage level that the instrument can source. It is in effect when either current or voltage is sourced.

This protection is in effect for both positive and negative output voltages.

When this attribute is used in a test sequence, it should be set before the turning the source on.

WARNING

Even with the overvoltage protection set to the lowest value, never touch anything connected to the terminals of the Model 2450 when the output is on. Always assume that a hazardous voltage (>30 V_{rms}) is present when the output is on. To prevent damage to the device under test (DUT) and external circuitry, do not set the voltage source to levels that exceed the overvoltage protection value.

Example

`smu.source.func = smu.FUNC_DC_VOLTAGE` `smu.source.protect.level = smu.PROTECT_40V`	Sets the maximum voltage limit of the instrument to 40 V.

Also see

Overvoltage protection (on page 2-98)

smu.source.protect.tripped

This attribute indicates if the overvoltage source protection feature is active.

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

Usage

```
value = smu.source.protect.tripped
```

value	Overvoltage protection not activated: `smu.OFF` Overvoltage protection activated: `smu.ON`

Details

When overvoltage protection is active, the instrument restricts the maximum voltage level that the instrument can source.

Example

`print(smu.source.protect.tripped)`	If overvoltage protection is active, the output is: `smu.ON`

Also see

smu.source.protect.level (on page 8-141)

smu.source.range

This attribute selects the range for the source for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	Current: 1e-08 Voltage: 20 mV

Usage

```
rangeValue = smu.source.range
smu.source.range = rangeValue
```

rangeValue	Set to the maximum expected voltage or current to be sourced; see **Details** for values; the ranges are: • Current: −1 A to 1 A • Voltage: −200 V to 200 V

Details

This command manually selects the measurement range for the specified source.

If you select a specific source range, the range must be large enough to source the value. If not, an overrange condition can occur.

If an overrange condition occurs, an event is displayed and the change to the setting is ignored.

The fixed current source ranges are 10 nA, 100 nA, 1 μA, 10 μA, 100 μA, 1 mA, 10 mA, 100 mA, and 1 A.

The fixed voltage source ranges are 20 mV, 200 mV, 2 V, 20 V, and 200 V.

When you read this value, the instrument returns the positive full-scale value that the instrument is presently using.

This command is intended to eliminate the time required by the automatic range selection.

To select the range, you can specify the approximate source value that you will use. The instrument selects the lowest range that can accommodate that level. For example, if you expect to source levels around 50 mV, send 0.05 (or 50e-3) to select the 200 mV range.

NOTE

If automatic range selection is set to on, when you select a specific range, automatic is set to off. To set the range to automatic selection, use the source autorange command.

Example

`smu.source.func = smu.FUNC_DC_CURRENT`
`smu.source.autorange = smu.OFF`
`smu.source.range = 1`
	Set the instrument to source current.
	Turn autorange off.
	Set the source range to 1 A.

Also see

Ranges (on page 2-101)
smu.source.autorange (on page 8-127)

smu.source.readback

This attribute determines if the instrument records the measured source value or the configured source value when making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	smu.ON

Usage

```
state = smu.source.readback
smu.source.readback = state
```

state	Disable read back: `smu.OFF` Enable read back: `smu.ON`

Details

When you use the configured source value (source readback off), the instrument records and displays the value that was configured. When you use the actual source value (source readback on), the instrument measures the actual source value immediately before making the device under test measurement.

Using source readback results in more accurate measurements, but also a reduction in measurement speed.

When source readback is on, the front-panel display shows the measured source value and the buffer records the measured source value immediately before the device-under-test measurement. When source readback is off, the front-panel display shows the configured source value and the buffer records the configured source value immediately before the device-under-test measurement.

Example

```
reset()
testDataBuffer = buffer.make(100)
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.measure.func = smu.FUNC_DC_CURRENT
smu.source.readback = smu.ON
smu.source.level = 10
smu.measure.count = 100
smu.source.output = smu.ON
smu.measure.read(testDataBuffer)
smu.source.output = smu.OFF
printbuffer(1, 100, testDataBuffer.sourcevalues,
   testDataBuffer)
```

Reset the instrument to default settings.
Make a buffer named `testDataBuffer` that can hold 100 readings.
Set source function to voltage.
Set the measurement function to current.
Set read back on.
Set the instrument to take 100 readings.
Turn the output on.
Take the measurements.
Turn the output off.
Get the source values and measurements from the buffer.

Also see

smu.measure.func (on page 8-103)
smu.source.func (on page 8-134)

smu.source.sweeplinear()

This function creates a linear source sweep configuration list and trigger model.

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

Usage

```
smu.source.sweeplinear(configListName, start, stop, points)
smu.source.sweeplinear(configListName, start, stop, points, delay)
smu.source.sweeplinear(configListName, start, stop, points, delay, count)
smu.source.sweeplinear(configListName, start, stop, points, delay, count,
   rangeType)
smu.source.sweeplinear(configListName, start, stop, points, delay, count,
   rangeType, failAbort)
smu.source.sweeplinear(configListName, start, stop, points, delay, count,
   rangeType, failAbort, dual)
smu.source.sweeplinear(configListName, start, stop, points, delay, count,
   rangeType, failAbort, dual, bufferName)
```

configListName	A string that contains the name of the configuration list that the instrument will create for this sweep
start	The voltage or current source level at which the sweep starts: • Current: -1.05 to 1.05 • Voltage: -210 to 210
stop	The voltage or current at which the sweep stops: • Current: -1.05 to 1.05 • Voltage: -210 to 210
points	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
delay	The delay between measurement points; default is `smu.DELAY_AUTO`, which enables autodelay, or a specific delay value from 50 μs to 10,000 seconds, or 0 for no delay
count	The number of times to run the sweep; default is `1`: • Infinite loop: `smu.INFINITE` • Finite loop: `1` to `268435455`
rangeType	The source range that is used for the sweep: • Most sensitive source range for each source level in the sweep: `smu.RANGE_AUTO` or `0` • Best fixed range: `smu.RANGE_BEST` or `2` (default) • Present source range for the entire sweep: `smu.RANGE_FIXED` or `1`
failAbort	Abort the sweep if the source limit is exceeded: `smu.ON` (default) Complete the sweep if the source limit is exceeded: `smu.OFF`
dual	Determines if the sweep runs from start to stop and then from stop to start: • Sweep from start to stop only: `smu.OFF` (default) • Sweep from start to stop, then stop to start: `smu.ON`
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

Sweeps are only available when the source function is set to voltage or current.

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it creates a source configuration list and populates the trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

Example

`reset()`	Reset the instrument to its defaults.
`smu.source.func = smu.FUNC_DC_VOLTAGE`	Set the source function to voltage.
`smu.source.range = 20`	Set the source range to 20 V.
`smu.source.sweeplinear("VoltLinSweep", 0, 10, 20, 1e-3, 1, smu.RANGE_FIXED)`	Set up a linear sweep that sweeps from 0 to 10 volts in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep `VoltLinSweep`.
`smu.measure.func = smu.FUNC_DC_CURRENT`	Set the measure function to current.
`smu.measure.range = 100e-6`	Set the current range to 100 μA.
`trigger.model.initiate()`	Start the sweep.

Also see

Sweep operation (on page 3-51)
trigger.model.initiate() (on page 8-187)

smu.source.sweeplinearstep()

This function sets up a linear source sweep configuration list and trigger model with a fixed number of steps.

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

Usage

```
smu.source.sweeplinearstep(configListName, start, stop, step)
smu.source.sweeplinearstep(configListName, start, stop, step, delay)
smu.source.sweeplinearstep(configListName, start, stop, step, delay, count)
smu.source.sweeplinearstep(configListName, start, stop, step, delay, count,
    rangeType)
smu.source.sweeplinearstep(configListName, start, stop, step, delay, count,
    rangeType, failAbort)
smu.source.sweeplinearstep(configListName, start, stop, step, delay, count,
    rangeType, failAbort, dual)
smu.source.sweeplinearstep(configListName, start, stop, step, delay, count,
    rangeType, failAbort, dual, bufferName)
```

configListName	A string that contains the name of the configuration list that the instrument will create for this sweep
start	The voltage or current source level at which the sweep starts: • Current: -1.05 to 1.05 • Voltage: -210 to 210
stop	The voltage or current at which the sweep stops: • Current: -1.05 to 1.05 • Voltage: -210 to 210
step	The step size at which the source level will change; must be more than 0
delay	The delay between measurement points; default is `smu.DELAY_AUTO`, which enables autodelay, a specific delay value from 50 µs to 10,000 seconds, or 0 for no delay
count	The number of times to run the sweep; default is `1`: • Infinite loop: `smu.INFINITE` • Finite loop: `1` to `268435455`
rangeType	The source range that is used for the sweep: • Most sensitive source range for each source level in the sweep: `smu.RANGE_AUTO` or `0` • Best fixed range: `smu.RANGE_BEST` or `2` (default) Present source range for the entire sweep: `smu.RANGE_FIXED` or `1`
failAbort	Abort the sweep if the source limit is exceeded: `smu.ON` (default) Complete the sweep if the source limit is exceeded: `smu.OFF`
dual	Determines if the sweep runs from start to stop and then from stop to start: • Sweep from start to stop only: `smu.OFF` (default) • Sweep from start to stop, then stop to start: `smu.ON`
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Detail

Sweeps are only available when the source function is set to voltage or current.

When the sweep is started, the instrument sources a specific voltage or current voltage to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it creates a trigger model with a uniform series of ascending or descending voltage or current changes, called steps. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the stop level, which is calculated from the number of steps. A measurement is performed at each source step (including the start and stop levels). At this level, the instrument performs another measurement and then stops the sweep.

The instrument uses the step size parameter to determine the number of source level changes. The source level changes in equal steps from the start level to the stop level. To avoid a setting conflicts error, make sure the step size is greater than the start value and less than the stop value. To calculate the number of source-measure points in a sweep, use the following formula:

$$step = \frac{stop - start}{points - 1}$$

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

Example

`reset()`	Reset the instrument to its defaults.
`smu.source.func = smu.FUNC_DC_CURRENT`	Set the source function to current.

`smu.source.range = 1`
`smu.measure.func = smu.FUNC_DC_VOLTAGE`
`smu.measure.range = 20`
	Set the source range to 1 A. Set the measure function to voltage with a range of 20 V.

`smu.source.sweeplinearstep("CurrLogSweep", -1.05,` `   1.05, .25, 10e-3, 1, smu.RANGE_FIXED)`	Set up a linear step sweep that sweeps from -1.05 A to 1.05 A in 0.25 A increments with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep `CurrLogSweep`.
`trigger.model.initiate()`	Start the sweep.

Also see

Sweep operation (on page 3-51)

smu.source.sweeplist()

This function sets up a sweep based on a configuration list, which allows you to customize the sweep.

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

Usage

```
smu.source.sweeplist(configListName)
smu.source.sweeplist(configListName, index)
smu.source.sweeplist(configListName, index, delay)
smu.source.sweeplist(configListName, index, delay, count)
smu.source.sweeplist(configListName, index, delay, count, failAbort)
smu.source.sweeplist(configListName, index, delay, count, failAbort, bufferName)
```

configListName	The name of the configuration list that the sweep uses; this must be defined before sending this command
index	The index in the configuration list where the sweep starts; default is `1`
delay	The delay between measurement points; default is 0 for no delay or you can set a specific delay value from 50 μs to 10,000 seconds
count	The number of times to run the sweep; default is `1`: • Infinite loop: `smu.INFINITE` • Finite loop: `1` to `268435455`
failAbort	Abort the sweep if the source limit is exceeded: `smu.ON` (default) Complete the sweep if the source limit is exceeded: `smu.OFF`
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

Sweeps are only available when the source function is set to voltage or current.

This command allows you to set up a custom sweep, using a configuration list to specify the source levels.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

To run the sweep, initiate the trigger model.

Example

```
reset()
smu.source.configlist.create("CurrListSweep")
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 100e-3
smu.source.level = 1e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 10e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 5e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 7e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 11e-3
smu.source.configlist.store("CurrListSweep")
smu.source.level = 9e-3
smu.source.configlist.store("CurrListSweep")
smu.source.sweeplist("CurrListSweep",  1, 0.001)
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.range = 20
trigger.model.initiate()
```

Reset the instrument to its defaults
Create a source configuration list called `CurrListSweep`.
Set the source function to current.
Set the source current range to 100 mA.
Set the source current level to 1 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 1 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 10 μA.
Save the source settings to `CurrListSweep`.
Set the source current level to 7 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 11 mA.
Save the source settings to `CurrListSweep`.
Set the source current level to 9 mA.
Save the source settings to `CurrListSweep`.
Set up a list sweep that uses the entries from the `CurrListSweep` configuration list and starts at index 1 of the list.
Set a source delay of 1 ms.
Start the sweep.

Also see

Configuration lists (on page 3-33)
Sweep operation (on page 3-51)
trigger.model.initiate() (on page 8-187)

smu.source.sweeplog()

This function creates a logarithmic source sweep configuration list and trigger model.

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

Usage

```
smu.source.sweeplog(configListName, start, stop, points)
smu.source.sweeplog(configListName, start, stop, points, delay)
smu.source.sweeplog(configListName, start, stop, points, delay, count)
smu.source.sweeplog(configListName, start, stop, points, delay, count, rangeType)
smu.source.sweeplog(configListName, start, stop, points, delay, count, rangeType,
    failAbort)
smu.source.sweeplog(configListName, start, stop, points, delay, count, rangeType,
    failAbort, dual)
smu.source.sweeplog(configListName, start, stop, points, delay, count, rangeType,
    failAbort, dual, bufferName)
smu.source.sweeplog(configListName, start, stop, points, delay, count, rangeType,
    failAbort, dual, bufferName, asymptote)
```

configListName	A string that contains the name of the configuration list that the instrument will create for this sweep
start	The voltage or current source level at which the sweep starts: • Current: 1 pA to 1.05 • Voltage: 1 pV to 210
stop	The voltage or current at which the sweep stops: • Current: 1 pA to 1.05 • Voltage: 1 pV to 210
points	The number of source-measure points between the start and stop values of the sweep (2 to 1e6); to calculate the number of source-measure points in a sweep, use the following formula: Points = [(Stop - Start) / Step] + 1
delay	The delay between measurement points; default is `smu.DELAY_AUTO`, which enables autodelay, or a specific delay value from 50 μs to 10,000 seconds, or 0 for no delay
count	The number of times to run the sweep; default is `1`: • Infinite loop: `smu.INFINITE` • Finite loop: `1` to `268435455`
rangeType	The source range that is used for the sweep: • Most sensitive source range for each source level in the sweep: `smu.RANGE_AUTO` or `0` • Best fixed range: `smu.RANGE_BEST` or `2` (default) Present source range for the entire sweep: `smu.RANGE_FIXED` or `1`
failAbort	Abort the sweep if the source limit is exceeded: `smu.ON` (default) Complete the sweep if the source limit is exceeded: `smu.OFF`
dual	Determines if the sweep runs from start to stop and then from stop to start: • Sweep from start to stop only: `smu.OFF` (default) • Sweep from start to stop, then stop to start: `smu.ON`
bufferName	The name of a reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

asymptote	Default is 0

Details

Sweeps are only available when the source function is set to voltage or current.

When the sweep is started, the instrument sources a specific voltage or current value to the device under test (DUT). A measurement is made for each point of the sweep.

When the sweep command is sent, it creates a trigger model. To run the sweep, initiate the trigger model.

The sweep continues until the source outputs the specified stop level. At this level, the instrument performs another measurement and then stops the sweep.

When you specify a delay, a delay block is added to the sweep trigger model. This delay is added to any source delay you may have set. For example, if you set 10 ms for the source delay and 25 ms for the delay in the for the log sweep command, the actual delay is 35 ms.

The range type specifies the source range that is used for the sweep. You can select the following options:

- Best fixed: The instrument selects a single fixed source range that accommodates all the source levels in the sweep. This avoids overshoots during sweeps.
- Auto: The instrument automatically goes to the most sensitive source range for each source level in the sweep.
- Fixed: The source remains on the range that is set when the sweep is started. If a sweep point that exceeds the capability of the source range, the source outputs the maximum level for that range.

The asymptote changes the inflection of the sweep curve and allows it to sweep through zero. You can use the asymptote parameter to customize the inflection and offset of the source value curve. Setting this parameter to zero provides a conventional logarithmic sweep. The asymptote value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep. The asymptote value must not be equal to or between the starting and ending values. It must be outside the range defined by the starting and ending values.

Example

```
reset()
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.range = 20
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.range = 100e-6
smu.source.sweeplog("VoltLogSweep", 1, 10, 20, 1e-3,
   1, smu.RANGE_FIXED)
trigger.model.initiate()
```

Reset the instrument to its defaults.
Set the source function to voltage.
Set the source range to 20 V.
Set the measure function to current.
Set the current range to 100 µA.
Set up a log sweep that sweeps from 1 to 10 volts in 20 steps with a source delay of 1 ms, a sweep count of 1, and a fixed source range. Name the configuration list that is created for this sweep `VoltLogSweep`.
Start the sweep.

Also see

Sweep operation (on page 3-51)
trigger.model.initiate() (on page 8-187)

smu.source.userdelay[N]

This attribute sets a user-defined delay that can be used in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list Function change	Configuration script Measure configuration list	0

Usage

```
delayTime = smu.source.userdelay[N]
smu.source.userdelay[N] = delayTime
```

delayTime	The delay in seconds (0 to 10000)
N	The user delay to which this time applies (1 to 5)

Details

To use this commands in a trigger model, assign the delay to the dynamic delay block.

Example

`smu.source.userdelay[1] = 3`	Set user delay 1 for the source to be 3 seconds.

Also see

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 8-206)

status.clear()

This function clears event registers and the event log.

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

Usage

```
status.clear()
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It does not affect the Questionable Event Enable or Operation Event Enable registers.

Example

`status.clear()`	Clear the bits in the registers

Also see

*CLS (on page B-2)

status.condition

This attribute stores the status byte condition register.

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

Usage

```
statusByte = status.condition
```

statusByte	The status byte

Details

You can use this command to read the status byte, which is returned as a numeric value.

When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence. The returned value can indicate that one or more status events occurred. If more than one bit of the register is set, *statusByte* equals the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128). See Understanding bit settings (on page C-15) for additional information about reading bit values.

NOTE

If you are using the GPIB, USB, or VXI-11 serial poll sequence of the Model 2450 to get the status byte (also called a serial poll byte), B6 is the Request for Service (RQS) bit. If the bit is set, it indicates that a serial poll (SRQ) has occurred. For additional detail, see Serial polling and SRQ (on page C-13)

The meanings of the individual bits of this register are shown in the following table.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.MSB`	An enabled measurement event
1	2	Not used	
2	4	`status.EAV`	An error or status message is present in the Error Queue
3	8	`status.QSB`	An enabled questionable event
4	16	`status.MAV`	A response message is present in the Output Queue
5	32	`status.ESB`	An enabled standard event
6	64	`status.MSS`	An enabled summary bit of the status byte register is set
7	128	`status.OSB`	An enabled operation event

Example

```
statusByte = status.condition
print(statusByte)
```

Returns `statusByte`.

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

None

status.operation.condition

This attribute reads the Operation Condition Register of the status model.

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

Usage

`operationRegister = status.operation.condition`

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

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page C-15).

Example

```
print(status.operation.condition)
```

Also see

Operation Event Register (on page C-8)

status.operation.enable

This attribute sets or reads the contents of the Operation Event Enable Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
.enable (RW)	Yes	status.preset()	Nonvolatile memory	0

Usage

```
operationRegister = status.operation.enable
status.operation.enable = operationRegister
```

operationRegister	The status of the operation status register

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

Example

`-- decimal 20480 = binary 0101 0000 0000 0000` `status.operation.enable = operationRegister`	Sets the 12 and 14 bits of the operation status enable register using a decimal value.

Also see

Operation Event Register (on page C-8)
Understanding bit settings (on page C-15)

status.operation.event

This attribute reads the Operation Event Register of the status model.

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

Usage

operationRegister = status.operation.event

operationRegister	The status of the operation status register

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example 1

`-- decimal 20480 = binary 0101 0000 0000 0000`
`operationRegister = 20480`
`status.operation.enable = operationRegister`
	Sets the 12 and 14 bits of the operation status enable register using a decimal value.

Also see

Operation Event Register (on page C-8)

status.operation.getmap()

This attribute requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers.

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

Usage

setEvent, *clearEvent* = status.operation.getmap(*bitNumber*)

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`print(status.operation.getmap(9))`	Query bit 9 of the Operation Event Register.

Also see

Operation Event Register (on page C-8)
status.operation.setmap() (on page 8-157)

status.operation.setmap()

This attribute maps events to bits in the Operation Event Register.

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

Usage

```
status.operation.setmap(bitNumber, setEvent)
status.operation.setmap(bitNumber, setEvent, clearEvent)
```

bitNumber	The bit number that is being mapped to an event
setEvent	The event of the event that sets the bits in the condition and event registers (–440 to 5800)
clearEvent	The event of the event that clears the bit in the condition register (–440 to 5800)

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

Example

`status.operation.setmap(0, 5000, 5020)`	When event 5000 occurs, bit 0 in the condition and event registers of the Operation Event Register are set. When event 5020 occurs, bit 0 in the condition register is cleared.

Also see

Operation Event Register (on page C-8)
Programmable status register sets (on page C-5)
status.operation.getmap() (on page 8-156)

status.questionable.condition

This attribute reads the Questionable Condition Register of the status model.

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

Usage

```
questionableRegister = status.questionable.condition
```

questionableRegister	The value of the register (0 to 65535)

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see Understanding bit settings (on page C-15).

Example

`print(status.questionable.condition)`	Reads the Questionable Condition Register.

Also see

Questionable Event Register (on page C-7)
Understanding bit settings (on page C-15)

status.questionable.enable

This attribute sets or reads the contents of the questionable event enable register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Nonvolatile memory	0

Usage

```
questionableRegister = status.questionable.enable
status.questionable.enable = questionableRegister
```

questionableRegister	The value of the register (0 to 65535)

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register is set.

For detail on interpreting the value of a register, see Understanding bit settings (on page C-15).

Example

`status.questionable.enable = 17` `print(status.questionable.enable)`	Set bits 0 and 4 of the Questionable Event Enable Register. Returns 17, which indicates the register was set correctly.

Also see

Questionable Event Register (on page C-7)

status.questionable.event

This attribute reads the questionable event register.

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

Usage

```
questionableRegister = status.questionable.event
```

questionableRegister	The value of the questionable status register (0 to 65535)

Example 1

`-- decimal 66 = binary 0100 0010`
`questionableRegister = 66`
`status.questionable.enable = questionableRegister`
	Uses a decimal value to set bits B1 and B6 of the status questionable enable register.

Example 2

`-- decimal 2560 = binary 00001010 0000 0000`
`questionableRegister = 2560`
`status.questionable.enable = questionableRegister`
	Uses a decimal value to set bits B9 and B11 of the status questionable enable register.

Also see

Questionable Event Register (on page C-7)

status.questionable.getmap()

This attribute requests the mapped set event and mapped clear event status for a bit in the Questionable Event Registers.

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

Usage

```
setEvent, clearEvent = status.questionable.getmap(bitNumber)
```

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

`print(status.questionable.getmap(9))`	Returns the events that were mapped to set and clear bit 9.

Also see

status.questionable.setmap() (on page 8-160)
Questionable Event Register (on page C-7)

status.questionable.setmap()

This attribute maps events to bits in the questionable event registers.

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

Usage

```
status.questionable.setmap(bitNumber, setEvent)
status.questionable.setmap(bitNumber, setEvent, clearEvent)
```

bitNumber	The bit number that is being mapped to an event
setEvent	The event that will set the bit
clearEvent	The event that will clear the bit

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

Also see

status.questionable.getmap() (on page 8-159)

status.request_enable

This attribute stores the settings of the Service Request (SRQ) Enable Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
SRQEnableRegister = status.request_enable
status.request_enable = SRQEnableRegister
```

SRQEnableRegister	The status of the service request (SRQ) enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings (0 to 255)

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	`status.MSB`	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	`status.EAV`	An error or status message is present in the Error Queue.
3	8	`status.QSB`	An enabled event in the Questionable Status Register has occurred.
4	16	`status.MAV`	A response message is present in the Output Queue.
5	32	`status.ESB`	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	`status.OSB`	An enabled event in the Operation Status Register has occurred.

Example 1

```
requestSRQEnableRegister = status.MSB +
      status.OSB
status.request_enable = requestSRQEnableRegister
```

Uses constants to set the MSB and OSB bits of the service request (SRQ) enable register and clear all other bits.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the MSB and OSB bits and clear all other bits of the service request (SRQ) enable register.

Example 3

```
status.request_enable = 0
```

Clear the register.

Also see

Status model (on page C-1)
Understanding bit settings (on page C-15)

status.preset()

This function resets all bits in the status model.

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

Usage

```
status.preset()
```

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the enable status request enable (`*SRE`) to standard enable (`*ESE`).

Preset does not affect the event queue.

The Status Event Status Register is not affected by this command.

Example

`status.preset()`	Resets the instrument status model.

Also see

Status model (on page C-1)

status.standard.enable

This attribute reads or sets the bits in the Status Enable register of the Standard Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
standardRegister = status.standard.enable
status.standard.enable = standardRegister
```

standardRegister	The value of the Status Enable register of the Standard Event Register (0 to 255)

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or value of the bit as the *standardRegister* parameter.

You can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B4, set *standardRegister* to 17 (which is the sum of 1 + 16). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.EXE
```

When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page B-6) command or TSP opc() (on page 8-77) function.
1	2	Not used	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	`status.standard.DDE`	An instrument operation did not execute properly due to an internal condition.
4	16	`status.standard.EXE`	The instrument detected an error while trying to execute a command.
5	32	`status.standard.CME`	A command error has occurred. See information following this table for descriptions of command errors.
6	64	`status.standard.URQ`	The instrument transitioned from remote control to local control.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example 1

```
standardRegister = status.standard.OPC + status.standard.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 a decimal value to set the OPC and EXE bits of the standard event status enable register.

Also see

Standard Event Register (on page C-3)
Understanding bit settings (on page C-15)

status.standard.event

This attribute returns the contents of the Standard Event Status Register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Status preset()	Not saved	0

Usage

`standardRegister = status.standard.event`

`standardRegister`	The status of the standard event status register

Details

When this command returns zero (0), no bits are set. You can send 0 to clear all bits.
The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page B-6) command or TSP opc() (on page 8-77) function.
1	2	`Not used`	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	`status.standard.DDE`	An instrument operation did not execute properly due to an internal condition.
4	16	`status.standard.EXE`	The instrument detected an error while trying to execute a command.
5	32	`status.standard.CME`	A command error has occurred. See information following this table for descriptions of command errors.
6	64	`status.standard.URQ`	The instrument transitioned from remote control to local control.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

`print(status.standard.event)`	May return the value `149`, showing that the Standard Event Status Register contains binary `10010101`

Also see

Standard Event Register (on page C-3)
Understanding bit settings (on page C-15)

timer.cleartime()

This function resets the timer to zero (0) seconds.

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

Usage

```
timer.cleartime()
```

Example

`timer.cleartime()`
`delay(3)`
`print(timer.gettime())`	

Also see

timer.gettime() (on page 8-165)

timer.gettime()

This function measures the elapsed time since the timer was last cleared.

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

Usage

```
time = timer.gettime()
```

time	The elapsed time in seconds (1 µs resolution)

Example 1

`timer.cleartime()`
`delay(3)`
`print(timer.gettime())`	

Also see

timer.cleartime() (on page 8-165)

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 (1 or 2)

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

`trigger.blender[2].clear()`	Clears the event detector for blender 2.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender performs OR operations or AND operations.

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

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The type of operation: • `true`: OR operation • `false`: AND operation
N	The trigger blender (1 or 2)

Details

This command selects whether the blender waits for any one event (`OR`) or waits for all selected events (`AND`) before signaling an output event.

Example

`trigger.blender[1].orenable = true`
`trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3`
`trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5`
	Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 8-167)

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 (1 or 2)

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 8-167)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

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

Usage

```
trigger.blender[N].reset()
```

N	The trigger event blender (1 or 2)

Details

The `trigger.blender[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.blender[N].orenable`
- `trigger.blender[N].stimulus[M]`

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

Example

`trigger.blender[1].reset()`	Resets the trigger blender 1 settings to factory defaults.

Also see

trigger.blender[N].orenable (on page 8-166)
trigger.blender[N].overrun (on page 8-167)
trigger.blender[N].stimulus[M] (on page 8-168)

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	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = event
```

event	The event that triggers the blender action; see **Details**
N	An integer that represents the trigger event blender (1 or 2)
M	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event. The *event* parameter can be any trigger event.

Use zero to disable the blender input.

The *event* parameter may be one of the existing trigger events shown in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: `*TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 8-167)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

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

Usage

```
triggered = trigger.blender[N].wait(timeout)
```

triggered	Trigger detection indication for blender
N	The trigger blender (1 or 2) on which to wait
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

`digio.line[3].mode = digio.MODE_TRIGGER_IN`
`digio.line[5].mode = digio.MODE_TRIGGER_IN`
`trigger.digin[3].edge = trigger.EDGE_FALLING`
`trigger.digin[5].edge = trigger.EDGE_FALLING`
`trigger.blender[1].orenable = true`
`trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3`
`trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5`
`print(trigger.blender[1].wait(3))`
	Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.
	Wait three seconds while checking if trigger blender 1 event has occurred.

Also see

trigger.blender[N].clear() (on page 8-166)

trigger.digin[N].clear()

This function clears the trigger event on a digital input line.

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

Usage

`trigger.digin[N].clear()`

N	Digital I/O trigger line (1 to 6)

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to `false`).

For this command to function as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

`trigger.digin[2].clear()`	Clears the trigger event detector on I/O line 2.

Also see

digio.line[N].mode (on page 8-43)
Digital I/O port (on page 3-62)
trigger.digin[N].overrun (on page 8-172)
trigger.digin[N].wait() (on page 8-172)

trigger.digin[N].edge

This attribute sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

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

Usage

```
detectedEdge = trigger.digin[N].edge
trigger.digin[N].edge = detectedEdge
```

detectedEdge	The trigger logic value; see **Details** for values
N	Digital I/O trigger line (1 to 6)

Details

Set *detectedEdge* to one of the following values:

Trigger mode values

detectedEdge	**Description**
trigger.EDGE_FALLING	Detects falling-edge triggers as input.
trigger.EDGE_RISING	Detects rising-edge triggers as input.
trigger.EDGE_EITHER	Detects rising- or falling-edge triggers as input.

When the line is programmed to be used as a trigger line (see the mode command), the output state of the I/O line is controlled through the trigger logic specified by this command.

To directly control the line state, set the mode of the line to digital and use the write command. When in digital mode with the line configured for open drain, the edge setting asserts a TTL low-pulse for output. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse for output.

Example

`digio.line[4].mode = digio.MODE_TRIGGER_IN` `trigger.digin[4].edge = trigger.EDGE_RISING`	Sets the trigger mode for I/O line 4 so it detects a rising-edge trigger as an input.

Also see

digio.line[N].mode (on page 8-43)
digio.line[N].reset() (on page 8-44)
digio.writeport() (on page 8-46)
trigger.digin[N].clear() (on page 8-170)
Digital I/O port (on page 3-62)

trigger.digin[N].overrun

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Digital I/O trigger *N* clear Digital I/O trigger *N* reset	Not saved	Not applicable

Usage

`overrun = trigger.digin[N].overrun`

overrun	Trigger overrun state (`true` or `false`)
N	Digital input line (1 to 6)

Details

If this is true, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

`overrun = trigger.digin[1].overrun` `print(overrun)`	If there is no trigger overrun on digital input 1, the output is: `false`

Also see

digio.line[N].mode (on page 8-43)
digio.line[N].reset() (on page 8-44)
Digital I/O port (on page 3-62)
trigger.digin[N].clear() (on page 8-170)

trigger.digin[N].wait()

This function waits for a trigger.

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

Usage

`triggered = trigger.digin[N].wait(timeout)`

triggered	The value `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 6)
timeout	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `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

`digio.line[4].mode = digio.MODE_TRIGGER_OUT`
`triggered = trigger.digin[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.line[N].mode (on page 8-43)
Digital I/O port (on page 3-62)
trigger.digin[N].clear() (on page 8-170)

trigger.digout[N].assert()

This function asserts a trigger on one of the digital I/O lines.

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

Usage

```
trigger.digout[N].assert()
```

N	Digital I/O trigger line (1 to 6)

Details

Initiates a trigger event and does not wait for completion. The set pulse width determines how long the trigger is asserted.

Example

`digio.line[2].mode = digio.MODE_TRIGGER_OUT` `trigger.digout[2].assert()`	Asserts a trigger on digital I/O line 2.

Also see

digio.line[N].mode (on page 8-43)
Digital I/O port (on page 3-62)
trigger.digout[N].pulsewidth (on page 8-174)

trigger.digout[N].logic

This attribute sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger *N* reset	Configuration script	trigger.LOGIC_POSITIVE

Usage

```
logicType = trigger.digout[N].logic
trigger.digout[N].logic = logicType
```

logicType	The trigger mode: `trigger.LOGIC_POSITIVE` or `trigger.LOGIC_NEGATIVE`
N	Digital I/O trigger line (1 to 6)

Details

This attribute configures the trigger event generator to assert a TTL pulse for output logic; positive is a high pulse, negative is a low pulse.

Example

`digio.line[4].mode = digio.MODE_TRIGGER_OUT` `trigger.digout[4].logic = trigger.LOGIC_NEGATIVE`	Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

digio.line[N].mode (on page 8-43)
digio.line[N].reset() (on page 8-44)
Digital I/O port (on page 3-62)

trigger.digout[N].pulsewidth

This attribute describes the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
width = trigger.digout[N].pulsewidth
trigger.digout[N].pulsewidth = width
```

width	The pulse width (0 to 100,000 s)
N	Digital I/O trigger line (1 to 6)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `trigger.digout[N].release()`.

Example

`digio.line[4].mode = digio.MODE_TRIGGER_OUT` `trigger.digout[4].pulsewidth = 20e-6`	Sets the pulse width for trigger line 4 to 20 µs.

Also see

digio.line[N].mode (on page 8-43)
digio.line[N].reset() (on page 8-44)
Digital I/O port (on page 3-62)
trigger.digout[N].assert() (on page 8-173)
trigger.digout[N].release() (on page 8-175)

trigger.digout[N].release()

This function releases an indefinite length or latched trigger.

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

Usage

```
trigger.digout[N].release()
```

N	Digital I/O trigger line (1 to 6)

Details

Releases a trigger that was asserted with an indefinite pulse width time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Example

`digio.line[4].mode = digio.MODE_TRIGGER_OUT` `trigger.digout[4].release()`	Releases digital I/O trigger line 4.

Also see

digio.line[N].mode (on page 8-43)
Digital I/O port (on page 3-62)
trigger.digout[N].assert() (on page 8-173)
trigger.digout[N].pulsewidth (on page 8-174)

trigger.digout[N].stimulus

This attribute selects the event that causes a trigger to be asserted on the digital output line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger *N* reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.digout[N].stimulus
trigger.digout[N].stimulus = event
```

event	The triggering event
N	Digital I/O trigger line (1 to 6)

Details

The digital trigger pulsewidth command determines how long the trigger is asserted. The trigger stimulus for a digital I/O line may be set to one of the existing trigger events, which are described in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *TRG • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

```
digio.line[2].mode = digio.MODE_TRIGGER_OUT
trigger.digout[2].stimulus = trigger.EVENT_TIMER3
```

Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.

Also see

digio.line[N].mode (on page 8-43)
digio.line[N].reset() (on page 8-44)
trigger.digin[N].clear() (on page 8-170)
trigger.digout[N].assert() (on page 8-173)
Digital I/O port (on page 3-62)

trigger.lanin[N].clear()

This function clears the event detector for a trigger.

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

Usage

```
trigger.lanin[N].clear()
```

N	The LAN event number to clear (1 to 8)

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the previous history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

`trigger.lanin[5].clear()`	Clears the event detector with LAN packet 5.

Also see

trigger.lanin[N].overrun (on page 8-178)

trigger.lanin[N].edge

This attribute sets the trigger operation and detection mode of the specified LAN event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle LAN trigger *N* reset	Configuration script	trigger.EDGE_EITHER

Usage

```
edgeState = trigger.lanin[N].edge
trigger.lanin[N].edge = edgeState
```

edgeState	The trigger mode; see the **Details** for more information
N	The LAN event number (1 to 8)

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values			
Mode		**Trigger packets detected as input**	**LAN trigger packet generated for output with a…**
`trigger.EDGE_EITHER`		Rising or falling edge (positive or negative state)	negative state
`trigger.EDGE_FALLING`		Falling edge (negative state)	negative state
`trigger.EDGE_RISING`		Rising edge (positive state)	positive state

Example

`trigger.lanin[1].edge = trigger.EDGE_FALLING`	Set the edge state of LAN event 1 to falling.

Also see

Digital I/O (on page 3-62)
TSP-Link system expansion interface (on page 3-118)

trigger.lanin[N].overrun

This attribute contains the overrun status of the event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger *N* clear LAN trigger *N* reset	Not applicable	Not applicable

Usage

```
overrun = trigger.lanin[N].overrun
```

overrun	The trigger overrun state for the specified LAN packet (`true` or `false`)
N	The LAN event number (1 to 8)

Details

This attribute indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

Example

`overrun = trigger.lanin[5].overrun` `print(overrun)`	Checks the overrun status of a trigger on LAN5 and outputs the value, such as: `false`

Also see

trigger.lanin[N].clear() (on page 8-177)
trigger.lanin[N].wait() (on page 8-179)
trigger.lanout[N].assert() (on page 8-179)
trigger.lanout[N].stimulus (on page 8-184)

trigger.lanin[N].wait()

This function waits for an input trigger.

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

Usage

`triggered = trigger.lanin[N].wait(timeout)`

triggered	Trigger detection indication (`true` or `false`)
N	The trigger packet over LAN to wait for (1 to 8)
timeout	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `trigger.lanin[N].wait()` or `trigger.lanin[N].clear()` was called, this function returns immediately.

After waiting for a LAN trigger event with this function, the event detector is automatically reset and rearmed regardless of the number of events detected.

Example

`triggered = trigger.lanin[5].wait(3)`	Wait for a trigger with LAN packet 5 with a timeout of 3 seconds.

Also see

trigger.lanin[N].clear() (on page 8-177)
trigger.lanin[N].overrun (on page 8-178)
trigger.lanout[N].assert() (on page 8-179)
trigger.lanout[N].stimulus (on page 8-184)

trigger.lanout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event.

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

Usage

`trigger.lanout[N].assert()`

N	The LAN event number (1 to 8)

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudo line state to the appropriate state.

The following indexes provide the listed LXI events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- …
- 8:LAN7

Example

`trigger.lanout[5].assert()`	Creates a trigger with LAN packet 5.

Also see

lan.lxidomain (on page 8-67)
trigger.lanin[N].clear() (on page 8-177)
trigger.lanin[N].overrun (on page 8-178)
trigger.lanin[N].wait() (on page 8-179)
trigger.lanout[N].assert() (on page 8-179)
trigger.lanout[N].ipaddress (on page 8-182)
trigger.lanout[N].protocol (on page 8-183)
trigger.lanout[N].stimulus (on page 8-184)

trigger.lanout[N].connect()

This function prepares the event generator for outgoing trigger events.

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

Usage

```
trigger.lanout[N].connect()
```

N	The LAN event number (1 to 8)

Details

This command prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event are changed.

Example

`trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST`
`trigger.lanout[1].connect()`
`trigger.lanout[1].assert()`
	Set the protocol for LAN trigger 1 to be multicast when sending LAN triggers. Then, after connecting the LAN trigger, send a message on LAN trigger 1 by asserting it.

Also see

trigger.lanin[N].overrun (on page 8-178)
trigger.lanin[N].wait() (on page 8-179)
trigger.lanout[N].assert() (on page 8-179)
trigger.lanout[N].ipaddress (on page 8-182)
trigger.lanout[N].protocol (on page 8-183)
trigger.lanout[N].stimulus (on page 8-184)

trigger.lanout[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 = trigger.lanout[N].connected`

connected	The LAN event connection state: • `true`: Connected • `false`: Not connected
N	The LAN event number (1 to 8)

Details

This is set to true when the LAN trigger is connected and ready to send trigger events following a successful `trigger.lanout[N].connect()` command. If the LAN trigger is not ready to send trigger events, this value is false.

This attribute is also false when the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attribute is changed or when the remote connection closes the connection.

Example

```
trigger.lanout[1].protocol =
   lan.PROTOCOL_MULTICAST
print(trigger.lanout[1].connected)
```

Outputs `true` if connected, or `false` if not connected.

Example output:

`false`

Also see

trigger.lanout[N].connect() (on page 8-180)
trigger.lanout[N].ipaddress (on page 8-182)
trigger.lanout[N].protocol (on page 8-183)

trigger.lanout[N].disconnect()

This function disconnects the LAN trigger event generator.

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

Usage

```
trigger.lanout[N].disconnect()
```

N	The LAN event number (1 to 8)

Details

When this command is set for TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attributes for this event are changed.

Also see

trigger.lanout[N].ipaddress (on page 8-182)
trigger.lanout[N].protocol (on page 8-183)

trigger.lanout[N].ipaddress

This attribute specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle LAN trigger *N* reset	Configuration script	"0.0.0.0"

Usage

```
ipAddress = trigger.lanout[N].ipaddress
trigger.lanout[N].ipaddress = ipAddress
```

ipAddress	The LAN address for this attribute as a string in dotted decimal notation
N	The LAN event number (1 to 8)

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

`trigger.lanout[3].protocol = lan.PROTOCOL_TCP`
`trigger.lanout[3].ipaddress = "192.0.32.10"`
`trigger.lanout[3].connect()`
	Set the protocol for LAN trigger 3 to be TCP when sending LAN triggers.
	Use IP address `"192.0.32.10"` to connect the LAN trigger.

Also see

trigger.lanout[N].connect() (on page 8-180)

trigger.lanout[N].logic

This attribute sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

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

Usage

```
logicType = trigger.lanout[N].logic
trigger.lanout[N].logic = logicType
```

logicType	`trigger.LOGIC_POSITIVE` `trigger.LOGIC_NEGATIVE`
N	The LAN event number (1 to 8)

Example

`trigger.lanout[2].logic = trigger.LOGIC_POSITIVE`	Set the logic for LAN trigger line 2 to positive.

Also see

None

trigger.lanout[N].protocol

This attribute sets the LAN protocol to use for sending trigger messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle LAN trigger *N* reset	Configuration script	lan.PROTOCOL_TCP

Usage

```
protocol = trigger.lanout[N].protocol
trigger.lanout[N].protocol = protocol
```

protocol	The protocol to use for messages from the trigger: • `lan.PROTOCOL_TCP` • `lan.PROTOCOL_UDP` • `lan.PROTOCOL_MULTICAST`
N	The LAN event number (1 to 8)

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

`print(trigger.lanout[1].protocol)`	Get LAN protocol that is being used for sending trigger messages for LAN event 1.

Also see

trigger.lanout[N].connect() (on page 8-180)
trigger.lanout[N].ipaddress (on page 8-182)

trigger.lanout[N].stimulus

This attribute specifies events that cause this trigger to assert.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle LAN trigger *N* reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.lanout[N].stimulus
trigger.lanout[N].stimulus = event
```

event	The LAN event that causes this trigger to assert
N	A number specifying the trigger packet over the LAN for which to set or query the trigger source (1 to 8)

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored and the action overrun is set.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

`trigger.lanout[5].stimulus = trigger.EVENT_TIMER1`	Use timer 1 trigger event as the source for LAN packet 5 trigger stimulus.

Also see

trigger.lanin[N].clear() (on page 8-177)
trigger.lanin[N].wait() (on page 8-179)
trigger.lanout[N].assert() (on page 8-179)
trigger.lanout[N].connect() (on page 8-180)

trigger.model.abort()

This function stops all trigger model commands on the instrument.

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

Usage

```
trigger.model.abort()
```

Details

When this command is received, the Model 2450 stops the trigger model.

Example

`trigger.model.abort()`	Terminates all commands related to the trigger model on the instrument.

Also see

Effect of GPIB line events on Model 2450 (on page 2-48)
Aborting the trigger model (on page 3-108)

trigger.model.getblocklist()

This attribute returns the settings for all trigger model building blocks.

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

Usage

```
trigger.model.getblocklist()
```

Details

This returns the settings for the trigger model.

Example

```
print(trigger.model.getblocklist())
```

Returns the settings for the trigger model. A typical output is:

```
1) SOURCE_OUTPUT        OUTPUT: ON
2) CONFIG_RECALL        CONFIG_LIST: ampLevel   INDEX: 1
3) DELAY_CONSTANT       DELAY: 0.100000
4) MEASURE              BUFFER: defbuffer1
5) CONFIG_RECALL        CONFIG_LIST: biasLevel   INDEX: 1
6) DELAY_CONSTANT       DELAY: 0.200000
7) BRANCH_COUNTER       VALUE: 19  BRANCH_BLOCK: 2
```

Also see

trigger.model.getbranchcount() (on page 8-187)

trigger.model.getbranchcount()

This attribute returns the count value of the trigger model counter block.

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

Usage

```
trigger.model.getbranchcount(blockNumber)
```

blockNumber	The sequence of the block in the trigger model

Details

This command returns the counter value. When the counter is active, this returns the count. If the trigger model has started or is running but has not yet reached the counter block, this value is 0.

Example

`print(trigger.model.getbranchcount(4))`	Returns the value of the counter for building block 4.

Also see

trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 8-193)

trigger.model.initiate()

This function starts the trigger model.

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

Usage

```
trigger.model.initiate()
```

Also see

Trigger model (on page 3-95)
trigger.model.abort() (on page 8-186)

trigger.model.load() — Config List

This function loads a predefined trigger model configuration that uses source and measure configuration lists.

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

Usage

```
trigger.model.load("ConfigList", measureConfigList, sourceConfigList)
trigger.model.load("ConfigList", measureConfigList, sourceConfigList, delay)
trigger.model.load("ConfigList", measureConfigList, sourceConfigList, delay,
    readingBuffer)
```

measureConfigList	A string that contains the name of the measurement configuration list to use
sourceConfigList	A string the contains the name of the source configuration list to use
delay	The delay that will occur before the measurement is made; default is 0
readingBuffer	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; default is `defbuffer1`

Details

This trigger model template incorporates a source configuration list and measure configuration list.

You must set up the configuration lists before loading the trigger model.

You can also set a delay and reading buffer.

Also see

None

trigger.model.load() — Duration Loop

This function loads a predefined trigger model configuration that makes continuous measurements for a specified amount of time.

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

Usage

```
trigger.model.load("DurationLoop", duration)
trigger.model.load("DurationLoop", duration, delay)
trigger.model.load("DurationLoop", duration, delay, readingBuffer)
```

duration	The amount of time for which to take measurements (0 to 100,000 seconds)
delay	The delay time before the measurement (seconds); defaults to 0
readingBuffer	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`

Details

When you load this predefined trigger model, you can specify amount of time to make a measurement and the length of the delay before the measurement.

Example

```
reset()

--set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT

--set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.level = 5

--turn on output and initiate readings
trigger.model.load("DurationLoop", 10, 0.01)
trigger.model.initiate()
```

Reset the instrument. Set the instrument to source voltage at 5 V. Set to measure current.
Load the duration loop trigger model to take measurements for 10 seconds with a 10 ms delay before each measurement.
Start the trigger model.

Also see

None

trigger.model.load() — Empty

This function resets the trigger model.

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

Usage

```
trigger.model.load("Empty")
```

Details

When you load this predefined trigger model, any existing trigger model settings are reset.
Any existing trigger blocks are deleted when you execute this command.

Example

`trigger.model.load("Empty")`
Reset trigger model settings.

Also see

None

trigger.model.load() — External Trigger

This function loads a predefined trigger model configuration that sets up an external trigger through the digital I/O.

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

Usage

```
trigger.model.load("ExternalTrigger", digInLine, digOutLine, count)
trigger.model.load("ExternalTrigger", digInLine, digOutLine, count, delay)
trigger.model.load("ExternalTrigger", digInLine, digOutLine, count, delay,
    readingBuffer)
```

digInLine	The digital input line (1 to 6); also the event that the trigger model will wait on in block 1
digOutLine	The digital output line (1 to 6)
count	Number of times to wait for the trigger and send out the trigger
delay	The time in seconds before the measurement is made; default is 0
readingBuffer	The name of the reading buffer, which may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer; defaults to `defbuffer1`

Details

This trigger model waits for a digital I/O event to occur, makes a measurement, and issues a notify event.

Also see

None

trigger.model.load() — Simple Loop

This function loads a predefined trigger model configuration.

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

Usage

```
trigger.model.load("SimpleLoop", count)
trigger.model.load("SimpleLoop", count, delay)
trigger.model.load("SimpleLoop", count, delay, readingBuffer)
```

count	The number of measurements to make
delay	The time before the measurement in seconds; default is 0
readingBuffer	The reading buffer; the default buffers (`defbuffer1` or `defbuffer2`) or the name of a user-defined buffer; if no buffer is specified, this parameter defaults to `defbuffer1`

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you defined in the count parameter.

Example

```
reset()

--set up measure function
smu.measure.func = smu.FUNC_DC_CURRENT
smu.measure.terminals = smu.TERMINALS_REAR
smu.measure.autorange = smu.ON
smu.measure.nplc = 1

--set up source function
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.ilimit.level = 0.1
smu.source.level = 20
smu.source.delay = 0.1
smu.source.highc = smu.OFF

--turn on output and initiate readings
smu.source.output = smu.ON
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

--Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i=1, defbuffer1.n do
      print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

--Discharge the capacitor to 0 V and turn off the output
smu.source.level=0
delay(2)
smu.source.output=smu.OFF
```

This example uses the Simple Loop trigger model template to do a capacitor test. This example outputs 200 readings that have output similar to the following example output:

```
Rdg # Time (s)      Current (A)
1   0             8.5718931952528e-11
2   0.151875      1.6215984111057e-10
3   0.303727      1.5521139928865e-10
. . .
198 29.91579194     1.5521250951167e-10
199 30.067648716    1.4131290582142e-10
200 30.219497716    1.5521067764368e-10
```

Also see

None

trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS

This function defines a trigger model block that always goes to a specific block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

`trigger.model.setblock(`*`blockNumber`*`, trigger.BLOCK_BRANCH_ALWAYS, `*`branchToBlock`*`)`

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger model block to execute when the trigger model reaches this block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by *branchToBlock*.

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ALWAYS, 20)
```

When the trigger model reaches block 6, always branch to block 20.

Also see

None

trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER

This function defines a trigger model block that branches to a specified block a specified number of times.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_COUNTER, targetCount,
    branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
targetCount	The number of times to repeat
branchToBlock	The trigger model block to execute when the counter is less than the *targetCount* value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value.

Example

```
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
print(trigger.model.getbranchcount(4))
```

When the trigger model reaches this block, the trigger model returns to block 2. This repeats 10 times.

An example of the return if the trigger model has reached this block 5 times is:

```
5
```

Also see

trigger.model.getbranchcount() (on page 8-187)

trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA

This command defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
    branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
    branchToBlock, measureBlock)
```

blockNumber	The sequence of the block in the trigger model
targetDifference	The value against which the block compares the difference between the measurements
branchToBlock	The block number of the trigger model block to execute when the difference between the measurements is less than the *targetDifference*
measureBlock	The *blockNumber* of the measurement block that makes the measurements to be compared

Details

This block calculates the difference between the last two measurements from a measure block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measurement block, it will compare measurements of a measure block that precedes the branch delta block. For example, if you have a measure block, a wait block, another measure block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure block.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_DELTA, 0.35, 8, 3)
```

Configure trigger block 5 to branch to block 8 when the measurement difference from block 3 is less than 0.35.

Also see

Delta (on page 3-100)

trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT

This command defines a trigger model block that branches to a block outside the normal trigger model flow if a measurement meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
    limitA, limitB, branchToBlock, measureBlock)
```

blockNumber	The sequence of the block in the trigger model
limitType	The type of limit (`trigger.LIMIT_ABOVE`, `trigger.LIMIT_BELOW`, `trigger.LIMIT_INSIDE`, or `trigger.LIMIT_OUTSIDE`)
limitA	The lower limit that the measurement is tested against; if *limitType* is set to: • `trigger.LIMIT_ABOVE`: This value is ignored • `trigger.LIMIT_BELOW`: The measurement must be below this value • `trigger.LIMIT_INSIDE`: This is the low limit that the measurement is compared against • `trigger.LIMIT_OUTSIDE`: This is the low limit that the measurement is compared against
limitB	The upper limit that the measurement is tested against; if *limitType* is set to: • `trigger.LIMIT_ABOVE`: The measurement must be above this value • `trigger.LIMIT_BELOW`: This value is ignored • `trigger.LIMIT_INSIDE`: This is the high limit that the measurement is compared against • `trigger.LIMIT_OUTSIDE`: This is the high limit that the measurement is compared against
branchToBlock	The block number of the trigger model block to execute when the measurement meets the defined criteria
measureBlock	The *blockNumber* of the measurement block that makes the measurement to be compared

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B. Limit A must be set, but is ignored when this type is selected.
- Below: The measurement is below the value set by limit A. Limit B must be set, but is ignored when this type is selected.
- Inside: The measurement is inside the values set by limits A and B. Limit A must be the low value and Limit B must be the high value.
- Outside: The measurement is outside the values set by limits A and B. Limit A must be the low value and Limit B must be the high value.

The measurement block must be a measurement building block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from the measurement building block is used.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, trigger.LIMIT_ABOVE,
    .1, 1, 2)
```

Also see

Constant limits (on page 3-99)

trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC

This command defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    branchToBlock, measureBlock)
```

blockNumber	The sequence of the block in the trigger model
limitType	The type of limit, which can be one of the following types: • `trigger.LIMIT_ABOVE` • `trigger.LIMIT_BELOW` • `trigger.LIMIT_INSIDE` • `trigger.LIMIT_OUTSIDE`
branchToBlock	The block number of the trigger model block to execute when the measurement meets the criteria set in the configuration list
measureBlock	The block number of the measurement block that makes the measurement to be compared

Details

The branch-on-user-limits block goes to a specified building block if a measurement meets the criteria set by this command.

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values. You set these limit threshold values as separate settings. Limit 1 and limit 2 are stored in the measurement configuration list. You can set them to different values in different indices of the measurement configuration list to allow you to step through different values. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

The type of limit can be:

- Above: The measurement is above the value set by the limit low value. The high value is not used when this type is selected.
- Below: The measurement is below the value set by the limit high value. The low value is not used when this type is selected.
- Inside: The measurement is inside the low and high values set for the limit.
- Outside: The measurement is outside the low and high values set for the limit.

The measurement block must be a measurement building block that occurs in the trigger model before the branch-on-constant-limits block.

Example

```
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC,
    trigger.LIMIT_OUTSIDE, 2, 10, 5)
```

Configure block 7 to check if limit 2 is outside its limit values, based on the measurements made in block 5. If values are outside the measurements, branch to block 10. If the values are not outside the measurements, trigger model execution continues to block 8.

Also see

smu.measure.limit[Y].low.value (on page 8-110)
smu.measure.limit[Y].high.value (on page 8-109)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT

This command branches to a specified block when a specified trigger event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ON_EVENT, event,
    branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
event	The event that must occur before the trigger block will act
branchToBlock	The block number of the trigger model block to execute when the specified event occurs

Details

The branch-on-event building block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

The event can be one of the events shown in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ON_EVENT, trigger.EVENT_DISPLAY,
    2)
```

When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the next block in the trigger model).

Also see

On event (on page 3-99)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE

This attribute causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE, branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time the trigger model encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when the trigger model reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model encounters this block.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE, 4)
```

When the trigger model reaches block 2, the trigger model goes to block 4 instead of going in the default sequence of block 3.

Also see

Once (on page 3-100)

trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED

This command defines a trigger model block that causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE_EXCLUDED,
   branchToBlock)
```

blockNumber	The sequence of the block in the trigger model
branchToBlock	The block number of the trigger model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded building block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts.

Example

`trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE_EXCLUDED, 4)`
When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

Once excluded (on page 3-101)

trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR

This command defines a trigger model block that clears the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR, bufferName)
```

blockNumber	The sequence of the block in the trigger model
bufferName	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used

Details

When the trigger model reaches the buffer clear trigger block, the instrument empties the buffer that is specified by the command. The specified buffer can be the default buffer or a buffer that you defined. Assigning the name in the buffer clear trigger block does not create a buffer; it only references an existing buffer.

Readings that are made after the buffer is cleared are added to the beginning of the buffer.

You must create the buffer before you define this block.

If no buffer name is assigned, the instrument clears default buffer 1.

Example

```
trigger.model.setblock(3, trigger.BLOCK_BUFFER_CLEAR, capTest2)
```

Assign trigger block 3 to buffer clear; when the trigger model reaches block 3, it clears the reading buffer named `capTest2`.

Also see

buffer.make() (on page 8-11)
Reading-buffer clear building block (on page 3-95)

trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT

This attribute recalls the settings at the next index point of a source or measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

`trigger.model.setblock(`*`blockNumber`*`, trigger.BLOCK_CONFIG_NEXT, `*`configurationList`*`)`

blockNumber	The sequence of the block in the trigger model
configurationList	The configuration list from which to recall settings

Details

When the trigger model reaches a configuration recall next building block, the settings at the next index point in a configuration list are restored.

Each time this block is encountered, the settings at the next index point in the configuration list are recalled and take effect before the next step executes. When the last index point in the list is reached, it returns to the first point.

Example

```
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "measTrigList")
```

Configure trigger block 5 to load the next index point in the configuration list named `measTrigList`.

Also see

Configuration lists (on page 3-33)

trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV

This command defines a trigger model block that recalls the settings stored at the previous index point in a measure or source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_PREV, configurationList)
```

blockNumber	The sequence of the block in the trigger model
configurationList	The configuration list from which to recall settings

Details

The configuration list previous index trigger block type recalls the previous index point in a configuration list. It configures the source or measure settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

Each time the trigger model reaches a configuration list previous block, it goes backward one index point. When the first point in the list is reached, it goes to the last index point in the configuration list.

Example

```
trigger.model.setblock(8, trigger.BLOCK_CONFIG_PREV, "measTrigList")
```

Configure trigger block 8 to load the previous index point in the configuration list named `measTrigList`.

Also see

Configuration lists (on page 3-33)

trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL

This attribute recalls the system settings that are stored in a measure or source configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL, configurationList)
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,
   configurationList, index)
```

blockNumber	The sequence of the block in the trigger model
configurationList	The configuration list to recall
index	The point in the configuration list to recall; default is 1

Details

When the trigger model reaches a configuration recall building block, the settings in the specified configuration list are recalled.

You can restore a specific set of configuration settings in the configuration list by defining the index.

Example

```
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 5)
```

Configure trigger block 3 to load index point 5 from the configuration list named `measTrigList`.

Also see

Configuration lists (on page 3-33)

trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT

This command adds a constant delay to the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DELAY_CONSTANT, time)
```

blockNumber	The sequence of the block in the trigger model
time	The amount of time to delay in seconds

Details

When the trigger model reaches a delay building block, it stops operation for the amount of time set by the delay. This delay is a fixed amount of time. If other delays have been set, this delay is in addition to the other delays.

Example

```
trigger.model.setblock(7, trigger.BLOCK_DELAY_CONSTANT, 30e-3)
```

Configure trigger block 7 to delay the trigger model before the next block until a delay of 30 ms elapses.

Also see

None

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC

This command adds a delay to the execution of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

`trigger.model.setblock(`*`blockNumber`*`, trigger.BLOCK_DELAY_DYNAMIC, `*`userDelay`*`)`

blockNumber	The sequence of the block in the trigger model
userDelay	The number of the user delay to recall: • `trigger.USER_DELAY_Sn`, where *n* is the number of the user delay (1 to 5) set by `smu.measure.userdelay[N]` • `trigger.USER_DELAY_Sn`, where *n* is the number of the user delay (1 to 5) set by `smu.source.userdelay[N]`

Details

When the trigger model reaches a delay building block, it stops the trigger model for the amount of time set by the delay.

The delay time is set by the user delay command.

Example

```
smu.source.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_SOURCE_OUTPUT, smu.ON)
trigger.model.setblock(2, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_S1)
trigger.model.setblock(3, trigger.BLOCK_MEASURE)
trigger.model.setblock(4, trigger.BLOCK_SOURCE_OUTPUT, smu.OFF)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay for source 1 to 5 seconds.
Set trigger block 1 to turn the source output on.
Set trigger block 2 to a dynamic delay that calls source user delay 1.
Set trigger block 3 to make a measurement.
Set trigger block 4 to turn the source output off.
Set trigger block 5 to branch to block 1 ten times.
Start the trigger model.

Also see

smu.measure.userdelay[N] (on page 8-126)
smu.source.userdelay[N] (on page 8-153)

trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO

This function defines a trigger model block that sets the lines on the digital I/O port high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

`trigger.model.setblock(`*`blockNumber`*`, trigger.BLOCK_DIGITAL_IO, `*`bitPattern`*`, `*`bitMask`*`)`

blockNumber	The sequence of the block in the trigger model
bitPattern	Sets the value that specifies the bit pattern
bitMask	Specifies the bit mask; if omitted, all lines are driven

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern. The pattern can be specified as a six-bit binary, hexadecimal, or integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The optional bit mask defines the bits in the pattern that are driven high or low. If the bit for a line is set to 1, the line is driven high. If the bit is set to 0, the line is driven low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63, 0x3F, 0b111111) or omit this parameter.

For this command to function as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

```
for x = 3,6 do digio.line[x].mode = digio.MODE_DIGITAL_OUT end
trigger.model.setblock(4, trigger.BLOCK_DIGITAL_IO, 20, 60)
```

The for loop configures digital I./O lines 3 through 6 as digital outputs. Trigger block `4` is then configured with a bit pattern of `20` (digital I/O lines 3 and 5 high). The optional bit mask is specified as `60` (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

digio.line[N].mode (on page 8-43)

trigger.model.setblock() — trigger.BLOCK_LOG_EVENT

This function allows you to log an event in the event log when the trigger model is running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_LOG_EVENT, eventNumber, message)
```

blockNumber	The sequence of the block in the trigger model
eventNumber	The event number: • `trigger.LOG_INFON` • `trigger.LOG_WARNN` • `trigger.LOG_ERRORN` Where *N* is 1 to 4; you can define up to four of each type
message	A string up to 31 characters

Details

This block allows you to log an event in the event log when the trigger model is running. Insert the block into the trigger model. When the trigger model executes the block, the event is logged.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

Example

```
trigger.model.setblock(1, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "info
   message")
```

Set trigger model block 1 to log an event.

Also see

None

trigger.model.setblock() — trigger.BLOCK_MEASURE

This function defines a trigger block that makes a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE, bufferName)
```

blockNumber	The sequence of the block in the trigger model
bufferName	The name of the buffer, which must be an existing buffer; if no buffer is defined, `defbuffer1` is used

Details

When the trigger model reaches the measurement block:

1. The instrument makes a reading.
2. The trigger model waits for the measurement to complete.
3. The instrument places the measurement into the specified reading buffer. If no buffer is specified, the reading is placed into the default buffer (`defbuffer1`).

If you are defining a specific reading buffer, you must create it before you define this block.

Example

`trigger.model.setblock(4, trigger.BLOCK_MEASURE)`	Make a measurement when the trigger model reaches block 4 and stores the setting in `defbuffer1`

Also see

buffer.make() (on page 8-11)
Measure building block (on page 3-96)

trigger.model.setblock() — trigger.BLOCK_NOP

This function creates a placeholder that performs no action in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

`trigger.model.setblock(blockNumber, trigger.BLOCK_NOP)`

blockNumber	The sequence of the block in the trigger model

Details

If you remove a trigger model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

`trigger.model.setblock(4, trigger.BLOCK_NOP)`	Set block number 4 to be a no operation block.

Also see

None

trigger.model.setblock() — trigger.BLOCK_NOTIFY

This function defines a trigger model block that generates a trigger event and immediately continues to the next block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFYN)
```

blockNumber	The sequence of the block in the trigger model
N	The notify event that is generated (1 to 8)

Details

When the trigger model reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

You can define up to eight notify blocks in a trigger model. You can reference the event that the notify block generates by other commands to assign a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

Define trigger model block 5 to be the notify 2 event. Assign the notify 2 event to be the stimulus for digital output line 3.

Also see

Notify building block (on page 3-97)

trigger.model.setblock() — trigger.BLOCK_SOURCE_OUTPUT

This attribute defines a trigger block that turns the output source on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_SOURCE_OUTPUT, state)
```

blockNumber	The sequence of the block in the trigger model
state	Turn the source off (`smu.OFF` or `0`) Turn the source on (`smu.ON` or `1`)
time	The number of the user delay to recall

Details

The source building block determines if the output source is turned on or off when the trigger model reaches this block.

This block does not determine the settings of the output source (such as the output voltage level and source delay). The source settings are determined by either the present settings of the instrument or by a source configuration list.

When you list trigger blocks, this block is listed as `SOURCE_OUTPUT`.

Example

`trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, 1)`	
Set trigger model to turn the source on when it reaches block 2.	

Also see

Wait building block (on page 3-96)

trigger.model.setblock() — trigger.BLOCK_WAIT

This command defines a trigger model block that waits for an event before allowing the trigger model to continue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, logic, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, logic, event, event)
```

blockNumber	The sequence of the block in the trigger model
event	The event that must occur before the trigger block will act
logic	If each event must occur before the trigger model continues: `trigger.WAIT_AND` If at least one of the events must occur before the trigger model continues: `trigger.WAIT_OR`

Details

You can use the wait block to synchronize measurements with other instruments and devices.

Events that you can set the instrument to wait for include:

- Digital input/output signals, such as DB-9 and TSP-Link
- LAN events
- Blenders

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. When the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again.

The instrument clears the memory of the recorded event when the trigger model is at the start block and when the trigger model exits the wait block.

You can have up to eight wait blocks in a trigger model.

All items in the list are subject to the same action — you cannot combine `AND` and `OR` logic in a single command.

The events can be one of the events shown in the following table.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

```
trigger.model.setblock(9, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
```

Set trigger model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing.

Also see

Wait building block (on page 3-96)

trigger.model.state()

This function returns the present state of the trigger model.

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

Usage

```
status = trigger.model.state()
```

status	The status of the trigger model: • `trigger.STATE_IDLE` • `trigger.STATE_RUNNING` • `trigger.STATE_WAITING` • `trigger.STATE_EMPTY` • `trigger.STATE_BUILDING` • `trigger.STATE_FAILED` • `trigger.STATE_ABORTING` • `trigger.STATE_ABORTED`

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model is presently executing.

The trigger model states are:

- Idle: The trigger model is stopped
- Running: The trigger model is running
- Waiting: The trigger model has been in the same wait block for more than 100 ms
- Empty: The trigger model is selected, but no blocks are defined
- Building: Blocks have been added.
- Failed: The trigger model is stopped because of an error.
- Aborting: The trigger model is stopping because of a user request.
- Aborted: The trigger model is stopped because of a user request.

Example

`print(trigger.model.state())`	An example output if the trigger model is waiting and is at block 9 would be: `trigger.STATE_WAITING  trigger.STATE_WAITING  9`

Also see

None

trigger.clear()

This function clears the trigger event detector.

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

Usage

```
trigger.clear()
```

Details

The trigger event detector indicates if an event has been detected since the last `trigger.wait()` command was sent. `trigger.clear()` clears the trigger event detector and discards the history of trigger events.

Also see

trigger.wait() (on page 8-215)

trigger.wait()

This function waits for a trigger event.

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

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 8-215)

trigger.timer[N].clear()

This function clears the timer event detector and overrun indicator for the specified trigger timer number.

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

Usage

```
trigger.timer[N].clear()
```

N	Trigger timer number to clear (1 to 4)

Details

This 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 8-216)

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	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

count	Number of times to repeat the trigger (0 to 1,048,575)
N	A trigger timer number (1 to 4)

Details

If *count* is set to a number greater than 1, the timer automatically starts the next delay at the expiration of the previous delay.

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

This command should not be used with the trigger model.

Example

`print(trigger.timer[1].count)`	Read trigger count for timer number 1.

Also see

trigger.timer[N].clear() (on page 8-216)
trigger.timer[N].reset() (on page 8-220)

trigger.timer[N].delay

This attribute sets and reads the timer delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

interval	Delay interval in seconds (5.00e-07 to 100,000)
N	Trigger timer number (1 to 4)

Details

Once the timer is enabled, each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

`trigger.timer[N].delaylist = {interval}`

This creates a delay list of one value.

Reading this attribute returns the delay interval that will be used the next time the timer is triggered.

This command should not be used with the trigger model.

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 8-220)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	{10e-6}

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

intervals	Table of delay intervals in seconds
N	Trigger timer number (1 to 4)

Details

Each time the timer is triggered after it is enabled, it uses the next delay period from the array. The default value is an array with one value of 10 µs.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be ≥ 50 µs.

This command should not be used with the trigger model.

Example

`trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}`	Set a delay list on trigger timer 3 with three delays (50 µs, 100 µs, and 150 µs).

`DelayList = trigger.timer[3].delaylist`
`for x = 1, table.getn(DelayList) do`
`   print(DelayList[x])`
`end`
	Read the delay list on trigger timer 3.
	
	Output (assuming the delay list was set to 50 µs, 100 µs, and 150 µs):
	`5.000000000e-05`
	`1.000000000e-04`
	`1.500000000e-04`

Also see

trigger.timer[N].reset() (on page 8-220)

trigger.timer[N].enable

This attribute enables the trigger timer.

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

Usage

```
state = trigger.timer[N].enable
trigger.timer[N].enable = state
```

state	Disable the trigger timer: `trigger.OFF` or `0` Enable the trigger timer: `trigger.ON` or `1`
N	Trigger timer number (1 to 4)

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

Example

`trigger.timer[3].enable = trigger.ON`	Enable the trigger timer for timer 3.

Also see

None

trigger.timer[N].reset()

This function resets trigger timer settings to their default values.

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

Usage

```
trigger.timer[N].reset()
```

N	Trigger timer number (1 to 4)

Details

The `trigger.timer[N].reset()` function resets the following attributes to their default values:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].enable`
- `trigger.timer[N].start.generate`
- `trigger.timer[N].start.fractionalseconds`
- `trigger.timer[N].start.seconds`
- `trigger.timer[N].stimulus`

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

Example

`trigger.timer[1].reset()`	Resets the attributes associated with timer 1 to their default values.

Also see

trigger.timer[N].count (on page 8-216)
trigger.timer[N].delay (on page 8-217)
trigger.timer[N].delaylist (on page 8-217)
trigger.timer[N].enable (on page 8-219)
trigger.timer[N].start.fractionalseconds (on page 8-221)
trigger.timer[N].start.generate (on page 8-221)
trigger.timer[N].start.overrun (on page 8-222)
trigger.timer[N].start.seconds (on page 8-222)
trigger.timer[N].start.stimulus (on page 8-223)

trigger.timer[N].start.fractionalseconds

This attribute configures an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes			0

Usage

```
time = trigger.timer[N].start.fractionalseconds
trigger.timer[N].start.fractionalseconds = time
```

time	The time in fractional seconds (0 to 2147483647)
N	Trigger timer number (1 to 4)

Example

`trigger.timer[1].start.fractionalseconds = 40`	Set the trigger timer to start in 40 nanoseconds.

Also see

trigger.timer[N].start.generate (on page 8-221)

trigger.timer[N].start.generate

This attribute specifies when timer events are generated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer *N* reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].start.generate
trigger.timer[N].start.generate = state
```

state	Generate a timer event when the timer delay elapses: `trigger.OFF` or `0` Generate a timer event when the timer starts and when the delay elapses: `trigger.ON` or `1`
N	Trigger timer number (1 to 4)

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event `trigger.EVENT_TIMERN`.

Example

`print(trigger.timer[1].start.generate)`	The setting of this command.

Also see

trigger.timer[N].reset() (on page 8-220)

trigger.timer[N].start.overrun

This attribute indicates if an event was ignored because of the event detector state.

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

Usage

```
state = trigger.timer[N].start.overrun
trigger.timer[N].start.overrun = state
```

state	The trigger overrun state (`true` or `false`)
N	Trigger timer number (1 to 4)

Details

This attribute indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

`print(trigger.timer[1].start.overrun)`	If an event was ignored, the output is `true`. If the event was not ignored, the output is `false`.

Also see

trigger.timer[N].reset() (on page 8-220)

trigger.timer[N].start.seconds

This attribute configures an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	0

Usage

```
time = trigger.timer[N].start.seconds
trigger.timer[N].start.seconds = time
```

time	The time in seconds (0 to 2147483647)
N	Trigger timer number (1 to 4)

Example

```
trigger.timer[1].start.seconds = localnode.gettime() + 30
trigger.timer[1].enable = trigger.ON
```

Set the trigger timer to start 30 seconds from the time when the timer is enabled.

Also see

None

trigger.timer[N].start.stimulus

This attribute describes the event that starts the trigger timer.

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

Usage

```
event = trigger.timer[N].start.stimulus
trigger.timer[N].start.stimulus = event
```

event	The event that starts the trigger timer
N	Trigger timer number (1 to 4)

Details

Set this attribute any trigger event to start the timer when that event occurs.

Set this attribute to zero (0) to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *`TRG` • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

`digio.line[3].mode = digio.MODE_TRIGGER_IN`
`trigger.timer[1].delay = 3e-3`
`trigger.timer[1].start.stimulus = trigger.EVENT_DIGIO3`
	Set digital I/O line 3 to be a trigger input.
	Set timer 1 to delay for 3 ms.
	Set timer 1 to start the timer when an event is detected on digital I/O line 3

Also see

None

trigger.timer[N].wait()

This function waits for a trigger.

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

Usage

```
triggered = trigger.timer[N].wait(timeout)
```

triggered	Trigger detection indication
N	Trigger timer number (1 to 4)
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

`triggered = trigger.timer[3].wait(10)` `print(triggered)`	Waits up to 10 seconds for a trigger on timer 3. If `false` is returned, no trigger was detected during the 10-second timeout. If `true` is returned, a trigger was detected.

Also see

trigger.timer[N].clear() (on page 8-216)

trigger.tsplinkin[N].clear()

This function clears the event detector for a trigger.

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

Usage

```
trigger.tsplinkin[N].clear()
```

N	The trigger line (1 to 3)

Details

The trigger event detector enters the detected state when an event is detected. When this command is sent, the instrument does the following actions:

- Clears the trigger event detector
- Discards the history of the trigger line
- Clears the `trigger.tsplinkin[N].overrun` attribute

Example

`tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN` `trigger.tsplinkin[2].clear()`	Clears the trigger event on TSP-Link line 2.

Also see

trigger.tsplinkin[N].overrun (on page 8-226)
tsplink.line[N].mode (on page 8-232)

trigger.tsplinkin[N].edge

This attribute indicates which trigger edge controls the trigger event detector for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.tsplinkin[N].edge
trigger.tsplinkin[N].edge = detectedEdge
```

detectedEdge	The trigger mode: • Detect falling-edge triggers as inputs: `trigger.EDGE_FALLING` • Detect rising-edge triggers as inputs: `trigger.EDGE_RISING` • Detect either falling or rising-edge triggers as inputs: `trigger.EDGE_EITHER`
N	The trigger line (1 to 3)

Details

When the edge is detected, the instrument asserts a TTL-low pulse for the output.

The output state of the I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

`tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN` `trigger.tsplinkin[3].edge = trigger.EDGE_RISING`	Sets synchronization line 3 to detect rising edge triggers as input.

Also see

digio.writeport() (on page 8-46)
tsplink.line[N].mode (on page 8-232)

trigger.tsplinkin[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup TSP-Link line *N* clear TSP-Link line *N* reset	Not applicable	Not applicable

Usage

overrun = trigger.tsplinkin[*N*].overrun

overrun	Trigger overrun state
N	The trigger line (1 to 3)

Details

Indicates that an event was ignored because the event detector had already detected an event when another event was detected.

Indicates the overrun state of the event detector built into the line.

It does not indicate whether an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

It does not indicate output trigger overrun.

Example

`print(trigger.tsplinkin[1].overrun)`	If an event on line 1 was ignored, displays `true`; if no additional event occurred, displays `false`.

Also see

None

trigger.tsplinkin[N].wait()

This function waits for a trigger.

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

Usage

triggered = trigger.tsplinkin[*N*].wait(*timeout*)

triggered	Trigger detection indication; set to one of the following values: `true`: A trigger is detected during the timeout period `false`: A trigger is not detected during the timeout period
N	The trigger line (1 to 3)
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events are detected since the last time this command or `trigger.tsplinkin[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

`tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN`
`triggered = trigger.tsplinkin[3].wait(10)`
`print(triggered)`
	Waits up to 10 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

trigger.tsplinkin[N].clear() (on page 8-224)
tsplink.line[N].mode (on page 8-232)

trigger.tsplinkout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding trigger event.

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

Usage

`trigger.tsplinkout[N].assert()`

N	The trigger line (1 to 3)

Details

Initiates a trigger event and does not wait for completion. The set pulse width determines how long the trigger is asserted.

Example

`tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN` `trigger.tsplinkout[2].assert()`	Asserts trigger on trigger line 2.

Also see

tsplink.line[N].mode (on page 8-232)

trigger.tsplinkout[N].logic

This attribute defines the trigger output with output logic for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	trigger.LOGIC_POSITIVE

Usage

```
logicType = trigger.tsplinkout[N].logic
trigger.tsplinkout[N].logic = logicType
```

logicType	The trigger logic: • `trigger.LOGIC_POSITIVE` • `trigger.LOGIC_NEGATIVE`
N	The trigger line (1 to 3)

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

If the output logic is set to `trigger.LOGIC_POSITIVE`, the trigger generator asserts a TTL-high pulse for output.

If the output logic is set to `trigger.LOGIC_NEGATIVE`, the trigger generator asserts a TTL-low pulse for output.

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.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].logic = trigger.LOGIC_POSITIVE
```

Sets the trigger logic for synchronization line 3 to output a positive pulse.

Also see

trigger.tsplinkout[N].assert() (on page 8-227)
tsplink.line[N].mode (on page 8-232)

trigger.tsplinkout[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	10e-6 (10 µs)

Usage

```
width = trigger.tsplinkout[N].pulsewidth
trigger.tsplinkout[N].pulsewidth = width
```

width	The pulse width (0.0 to 100,000 seconds)
N	The trigger line (1 to 3)

Details

Setting the pulse width to 0 asserts the trigger indefinitely.

Example

`tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN` `trigger.tsplinkout[3].pulsewidth = 20e-6`	Sets pulse width for trigger line 3 to 20 µs.

Also see

trigger.tsplinkout[N].assert() (on page 8-227)
trigger.tsplinkout[N].release() (on page 8-229)
tsplink.line[N].mode (on page 8-232)

trigger.tsplinkout[N].release()

This function releases a latched trigger on the given TSP-Link trigger line.

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

Usage

```
trigger.tsplinkout[N].release()
```

N	The trigger line (1 to 3)

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

`tsplink.line[3].mode =`
`   tsplink.MODE_TRIGGER_OPEN_DRAIN`
`trigger.tsplinkout[3].release()`
	Releases trigger line 3.

Also see

trigger.tsplinkout[N].assert() (on page 8-227)
tsplink.line[N].mode (on page 8-232)

trigger.tsplinkout[N].stimulus

This attribute specifies the event that causes the synchronization line to assert a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.tsplinkout[N].stimulus
trigger.tsplinkout[N].stimulus = event
```

event	The event identifier for the triggering event (see **Details**)
N	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to `trigger.EVENT_NONE`.

Do not use this attribute when triggering under script control. Use `trigger.tsplinkout[N].assert()` instead.

The *`event`* parameters that you can use are described in the table below.

Trigger events	
Event description	**Event constant**
No trigger event	`trigger.EVENT_NONE`
Front-panel TRIGGER key press	`trigger.EVENT_DISPLAY`
Notify trigger block *N* (1 to 8) generates a trigger event when the trigger model executes it	`trigger.EVENT_NOTIFYN`
A command interface trigger occurred: • Any remote interface: *TRG • GPIB only: GET bus command • VXI-11: VXI-11 command `device_trigger`	`trigger.EVENT_COMMAND`
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line *N* (1 to 6)	`trigger.EVENT_DIGION`
Line edge detected on TSP-Link synchronization line *N* (1 to 3)	`trigger.EVENT_TSPLINKN`
Appropriate LXI trigger packet is received on LAN trigger object *N* (1 to 8)	`trigger.EVENT_LANN`
Trigger event blender *N* (1 to 2), which combines trigger events	`trigger.EVENT_BLENDERN`
Trigger timer *N* (1 to 4) expired	`trigger.EVENT_TIMERN`
Source limit condition occurs	`trigger.EVENT_SOURCE_LIMIT`

Example

`print(trigger.tsplinkout[3].stimulus)`	Outputs the event that will start action on TSP-Link trigger line 3.

Also see

trigger.tsplinkout[N].assert() (on page 8-227)
tsplink.line[N].reset() (on page 8-233)

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	Not applicable	Nonvolatile memory	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 3-124)

tsplink.initialize()

This function initializes all instruments and enclosures in the TSP-Link system.

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

Usage

```
nodesFound = tsplink.initialize()
tsplink.initialize()
tsplink.initialize(expectedNodes)
```

nodesFound	The number of nodes actually found on the system, including the node on which the command is running
expectedNodes	The number of nodes expected on the system (1 to 32)

Details

This function regenerates the system configuration information regarding the nodes connected to the TSP-Link system. You must initialize the system after making configuration changes. Changes that require you to initialize the system include:

- Turning off power or rebooting any instrument in the system
- Changing node numbers on any instrument in the system
- Rearranging or disconnecting the TSP-Link cable connections between instruments

If the only node on the TSP-Link network is the one running the command and *expectedNodes* is not provided, this function generates an error. If you set *expectedNodes* to 1, the node is initialized.

If you include *expectedNodes*, if *nodesFound* is less than *expectedNodes*, an error is generated.

Example

`nodesFound = tsplink.initialize(2)`
`print("Nodes found = " .. nodesFound)`
	Perform a TSP-Link initialization 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.showevents = 7`:
	`1219, TSP-Link found fewer nodes than expected`
	`Nodes found = 1`

Also see

localnode.showevents (on page 8-73)
tsplink.node (on page 8-234)
tsplink.state (on page 8-235)

tsplink.line[N].mode

This attribute defines the trigger operation of a TSP-Link line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line *N* reset	Configuration script	tsplink.MODE_DIGITAL_OPEN_DRAIN

Usage

```
mode = tsplink.line[N].mode
tsplink.line[N].mode = mode
```

mode	The trigger mode; see **Details**
N	The trigger line (1 to 3)

Details

This command defines whether or not the line is used by the trigger model and if it is an input or output.

The line mode can be set to the following options:

- TSP-Link digital open drain line: `tsplink.MODE_DIGITAL_OPEN_DRAIN`
- TSP-Link trigger open drain line: `tsplink.MODE_TRIGGER_OPEN_DRAIN`
- TSP-Link trigger synchronous master: `tsplink.MODE_SYNCHRONOUS_MASTER`
- TSP-Link trigger synchronous acceptor: `tsplink.MODE_SYNCHRONOUS_ACCEPTOR`

Example

`tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN`	Sets the trigger mode for synchronization line 3 as a trigger open drain line.

Also see

trigger.tsplinkin[N].edge (on page 8-225)
trigger.tsplinkout[N].logic (on page 8-228)

tsplink.line[N].reset()

This function resets some of the TSP-Link trigger attributes to their defaults.

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

Usage

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

N	The trigger line (1 to 3)

Details

The `tsplink.line[N].reset()` function resets the following attributes to their default values:

- `trigger.tsplinkin[N].edge`
- `trigger.tsplinkout[N].logic`
- `tsplink.line[N].mode`
- `trigger.tsplinkout[N].stimulus`
- `trigger.tsplinkout[N].pulsewidth`

This also clears `trigger.tsplinkin[N].overrun`.

Example

`tsplink.line[3].reset()`	Resets TSP-Link trigger line 3 attributes to default values.

Also see

trigger.tsplinkin[N].edge (on page 8-225)
trigger.tsplinkin[N].overrun (on page 8-226)
trigger.tsplinkout[N].logic (on page 8-228)
trigger.tsplinkout[N].pulsewidth (on page 8-228)
trigger.tsplinkout[N].stimulus (on page 8-229)
tsplink.line[N].mode (on page 8-232)

tsplink.line[N].state

This attribute reads or writes the digital state of a TSP-Link synchronization line.

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

Usage

```
lineState = tsplink.line[N].state
tsplink.line[N].state = lineState
```

lineState	The state of the synchronization line: • Low: `tsplink.STATE_LOW` or `0` • High: `tsplink.STATE_HIGH` or `1`
N	The trigger line (1 to 3)

Example

`lineState = tsplink.line[3].state` `print(lineState)`	Assume line 3 is set high, and then the state is read. Output: `tsplink.STATE_HIGH`

Also see

tsplink.line[N].mode (on page 8-232)

tsplink.master

This attribute reads the node number assigned to the master node.

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

Usage

```
masterNodeNumber = tsplink.master
```

masterNodeNumber	The node number of the master node (1 to 64)

Details

This attribute returns the node number of the master in a set of instruments connected using TSP-Link.

Example

`LinkMaster = tsplink.master`	Store the TSP-Link master node number in a variable called LinkMaster.

Also see

tsplink.initialize() (on page 8-231)

tsplink.node

This attribute defines the node number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	2

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

nodeNumber	The node number of the instrument or enclosure (1 to 64)

Details

This attribute sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.reset()` from an earlier TSP-Link instrument or `tsplink.initialize()` is executed on any node in the system.

Each node connected to the TSP-Link system must be assigned a different node number.

Example

`tsplink.node = 3`	Sets the TSP-Link node for this instrument to number 3.

Also see

tsplink.initialize() (on page 8-231)
tsplink.state (on page 8-235)

tsplink.readport()

This function reads the TSP-Link synchronization lines as a digital I/O port.

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

Usage

data = tsplink.readport()

data	Numeric value that indicates which lines are set

Details

The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and the value of bit 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. This indicates that line 2 is high (1), and that the other two lines are low (0).

Example

`data = tsplink.readport()` `print(data)`	Reads state of all three TSP-Link lines. Assuming line 2 is set high, the output is: `2.000000e+00` (binary 010) The format of the output may vary depending on the ASCII precision setting.

Also see

Triggering using TSP-Link synchronization lines (on page 3-113)
tsplink.line[N].state (on page 8-233)
tsplink.writeport() (on page 8-236)

tsplink.state

This attribute describes the TSP-Link online state.

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

Usage

state = tsplink.state

state	TSP-Link state (`online` or `offline`)

Details

When the instrument power is first turned on, the state is `offline`. After `tsplink.initialize()` or `tsplink.reset()` is successful, the state is `online`.

Example

`state = tsplink.state` `print(state)`	Read the state of the TSP-Link system. If it is online, the output is: `online`

Also see

tsplink.initialize() (on page 8-231)
tsplink.node (on page 8-234)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines as a digital I/O port.

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

Usage

```
tsplink.writeport(data)
```

data	Value to write to the port (0 to 7)

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other two lines are set low (0).

The `reset()` function does not affect the present states of the trigger lines.

Example

`tsplink.writeport(3)`	Sets the synchronization lines 1 and 2 high (binary 011).

Also see

tsplink.line[N].state (on page 8-233)
tsplink.readport() (on page 8-235)

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))`	Write data to a device, then print how much is available. Output: `6.00000e+00`
`tspnet.clear(testdevice)` `print(tspnet.readavailable(testdevice))`	Clear data and print how much data is available again. Output: `0.00000e+00`

Also see

tspnet.connect() (on page 8-237)
tspnet.readavailable() (on page 8-242)
tspnet.write() (on page 8-248)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

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

Usage

```
connectionID = tspnet.connect(ipAddress)
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

connectionID	The connection ID to be used as a handle in all other `tspnet` function calls
ipAddress	IP address to which to connect in a string
portNumber	Port number (default 5025)
initString	Initialization string to send to *ipAddress*

Details

This command connects a device to another device through the LAN interface. If the *portNumber* is 23, the interface uses the Telnet protocol and sets appropriate termination characters to communicate with the device.

If a *portNumber* and *initString* are provided, it is assumed that the remote device is not TSP-enabled. The Model 2450 does not perform any extra processing, prompt handling, error handling, or sending of commands. In addition, the `tspnet.tsp.*` commands cannot be used on devices that are not TSP-enabled.

If neither a *portNumber* nor an *initString* is provided, the remote device is assumed to be a Keithley Instruments TSP-enabled device. Depending on the state of the `tspnet.tsp.abortonconnect` attribute, the Model 2450 sends an `abort` command to the remote device on connection.

The Model 2450 also enables TSP prompts on the remote device and event management. The Model 2450 places remote errors and events from the TSP-enabled device in its own event queue and prefaces these events with `Remote Error`, followed by an event description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show events by changing `localnode.showerrors` or `localnode.showevents` on the remote TSP-enabled device. If you do this, subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

`instrumentID = tspnet.connect("192.0.2.1")`
`if instrumentID then`
`   -- Use instrumentID as needed here`
`   tspnet.disconnect(instrumentID)`
`end`
	Connect to a TSP-enabled device.

Example 2

`instrumentID = tspnet.connect("192.0.2.1", 1394,`
`    "*rst\r\n")`
`if instrumentID then`
`   -- Use instrumentID as needed here`
`   tspnet.disconnect(instrumentID)`
`end`
	Connect to a device that is not TSP-enabled.

Also see

localnode.prompts (on page 8-71)
localnode.showevents (on page 8-73)
tspnet.tsp.abortonconnect (on page 8-245)
tspnet.disconnect() (on page 8-238)

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 8-237)

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 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
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 2450 decodes the response message according to the format specified in the format string and returns the message as return values from the function (see `tspnet.read()` for format specifiers).

When this command is sent to a TSP-enabled instrument, the Model 2450 suspends operation until a timeout error is generated or until the instrument responds, even if no format string is specified. The TSP prompt from the remote instrument is read and thrown away. The Model 2450 places any remotely generated errors and events into its event queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

`tspnet.execute(instrumentID, "runScript()")`	Command the remote device to run a script named `runScript`.

Example 2

`tspnet.termination(instrumentID, tspnet.TERM_CRLF)`
`tspnet.execute(instrumentID, "*idn?")`
`print("tspnet.execute returns:", tspnet.read(instrumentID))`
	Print the `*idn?` string from the remote device.

Also see

tspnet.connect() (on page 8-237)
tspnet.read() (on page 8-241)
tspnet.termination() (on page 8-243)
tspnet.write() (on page 8-248)

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 IDN string may appear as:
	`Keithley Instruments Inc., Model 2450, 00000170, 1.0.0`

Also see

tspnet.connect() (on page 8-237)

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 2450 pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When the Model 2450 reads from a TSP-enabled remote instrument, the Model 2450 removes Test Script Processor (TSP®) prompts and places any errors or events it receives from the remote instrument into its own event queue. The Model 2450 prefaces events and errors from the remote device with `Remote Error`, followed by the event number and description.

Example

`tspnet.write(deviceID, "*idn?\r\n")`	Send the `"*idn?\r\n"` message to the instrument connected as `deviceID`.
`print("write/read returns:", tspnet.read(deviceID))`	Display the response that is read from `deviceID` (based on the `*idn?` message).

Also see

tspnet.connect() (on page 8-237)
tspnet.readavailable() (on page 8-242)
tspnet.timeout (on page 8-244)
tspnet.write() (on page 8-248)

tspnet.readavailable()

This function checks to see if data is available from the remote device.

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

Usage

bytesAvailable = `tspnet.readavailable(`*connectionID*`)`

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from `tspnet.connect()`

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")

repeat bytes = tspnet.readavailable(ID) until bytes > 0

print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that will create data.

Wait for data to be available.

Also see

tspnet.connect() (on page 8-237)
tspnet.read() (on page 8-241)

tspnet.reset()

This function disconnects all TSP-Net sessions.

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

Usage

```
tspnet.reset()
```

Details

This command disconnects all remote instruments connected through TSP-Net. For TSP-enabled devices, this causes any commands or scripts running remotely to be terminated.

Also see

None

tspnet.termination()

This function sets the device line termination sequence.

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

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

type	The termination type: • `tspnet.TERM_LF` • `tspnet.TERM_CR` • `tspnet.TERM_CRLF` • `tspnet.TERM_LFCR`

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 8-237)
tspnet.disconnect() (on page 8-238)

tspnet.timeout

This attribute sets the timeout value for the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	20.0 (20 s)

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

`value`	The timeout duration in seconds (0.001 to 30.000)

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands will wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution, but is only accurate to the nearest 10 ms.

Example

`tspnet.timeout = 2.0`	Sets the timeout duration to two seconds.

Also see

tspnet.connect() (on page 8-237)
tspnet.execute() (on page 8-239)
tspnet.read() (on page 8-241)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

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

Usage

`tspnet.tsp.abort(`*`connectionID`*`)`

connectionID	Integer value used as a handle for other `tspnet` commands

Details

This function is appropriate only for TSP-enabled instruments.
Sends an abort command to the remote instrument.

Example

`tspnet.tsp.abort(testConnection)`	Stops remote instrument execution on `testConnection`.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	`1` (enable) or `0` (disable)

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument will not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

`tspnet.tsp.abortonconnect = 0`	Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

tspnet.connect() (on page 8-237)

tspnet.tsp.rbtablecopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

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

Usage

```
table = tspnet.tsp.rbtablecopy(connectionID, name)
table = tspnet.tsp.rbtablecopy(connectionID, name, startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other `tspnet` commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The *startIndex* and *endIndex* parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

`times = tspnet.tsp.rbtablecopy(testTspdevice, "testRemotebuffername.timestamps", 1, 3)` `print(times)`	Copy the specified timestamps table for items 1 through 3, then display the table. Sample output: `01/01/2011 10:10:10.0000013,01/01/2011 10:10:10.0000233,01/01/2011 10:10:10.0000576`

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

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

Usage

```
tspnet.tsp.runscript(connectionID, name, script)
```

connectionID	Integer value used as an identifier for other `tspnet` commands
name	The name that is assigned to the script
script	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions at a later time.

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 8-239)

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 2450 sends *inputString* to the remote instrument exactly as indicated. The *inputString* must contain any necessary new lines, termination, or other syntax elements needed to complete properly.

Because `tspnet.write()` does not process output from the remote instrument, do not send commands that generate too much output without processing the output. This command can stop executing if there is too much unprocessed output from previous commands.

Example

`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 8-237)
tspnet.read() (on page 8-241)

upgrade.previous()

This function returns to a previous version of the Model 2450 firmware.

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

Usage

```
upgrade.previous()
```

Details

This function allows you to revert to an earlier version of the firmware.

When you send this function, the instrument searches the flash drive that is inserted in the front-panel USB port for an upgrade file. If the file is found, the instrument performs the upgrade. An error is returned if an upgrade file is not found.

Also see

Upgrading the firmware (on page A-3)
upgrade.unit() (on page 8-249)

upgrade.unit()

This function upgrades the Model 2450 firmware.

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

Usage

```
upgrade.unit()
```

Details

When `upgrade.unit()` is used, the firmware is only loaded if the version of the firmware component is newer than the existing version. If the version is older or at the same revision level, it is not upgraded.

When you send this function, the instrument searches the flash drive that is inserted in the front-panel USB port for an upgrade file. If the file is found, the instrument verifies that the file is a newer version. If the version is older or at the same revision level, it is not upgraded. If it is a newer version, the instrument performs the upgrade. An error is returned if no upgrade file is found.

Also see

Upgrading the firmware (on page A-3)
upgrade.previous() (on page 8-248)

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.

Also see

userstring.catalog() (on page 8-250)
userstring.delete() (on page 8-251)
userstring.get() (on page 8-251)

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

`for name in userstring.catalog() do`
`   print(name .. " = " ..`
`      userstring.get(name))`
`end`
	Prints all `userstring` key-value pairs.
	
	Output:
	`product = Widgets`
	`assetnumber = 236`
	`contact = John Doe`
	The above output lists the user-defined strings added in the example for the `userstring.add()` function. Notice the key-value pairs are not listed in the order they were added.

Also see

userstring.add() (on page 8-249)
userstring.delete() (on page 8-251)
userstring.get() (on page 8-251)

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 8-249)
userstring.catalog() (on page 8-250)
userstring.get() (on page 8-251)

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

`value = userstring.get("assetnumber")` `print(value)`	Read the value associated with a 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 8-249)
userstring.catalog() (on page 8-250)
userstring.delete() (on page 8-251)

waitcomplete()

This function waits for all overlapped commands in a specified group to complete.

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

Usage

```
waitcomplete()
waitcomplete(group)
```

group	Specifies which TSP-Link group on which to wait

Details

This function will wait for all previously started overlapped commands to complete.

A group number may only be specified when this node is the master node.

If no *group* is specified, the local group is used.

If zero (0) is specified for the *group*, this function waits for all nodes in the system.

NOTE

Any nodes that are not assigned to a group (group number is 0) are part of the master node's group.

Example 1

`waitcomplete()`	Waits for all nodes in the local group.

Example 2

`waitcomplete(G)`	Waits for all nodes in group G.

Example 3

`waitcomplete(0)`	Waits for all nodes on the TSP-Link network.

Also see

None

Section 9

Frequently asked questions (FAQs)

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 view the system information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Information**. The system information displays, including the serial number.
3. To return to the Home page, select **Home**.

To view system information using SCPI commands:

Send the command:

```
*IDN?
```

To view system information using TSP commands:

Send the command:

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

What VISA resource name is required?

To determine the VISA resource name that is required to communicate with the instrument, you can run the Keithley Configuration Panel. The Configuration Panel automatically detects all instruments connected to the computer.

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

To run the Configuration Panel, click **Start > Programs > Keithley Instruments > Keithley Configuration Panel** and follow the steps in the wizard.

Can I use Agilent GPIB cards with Keithley drivers?

Yes, as long as the instrument driver uses VISA for instrument communication. This is true for any instrument driver that is IVI or VXI/PnP based.

How do I check the driver for the device?

To check the driver for the USB Test and Measurement Device:

1. Open the Windows Device Manager.

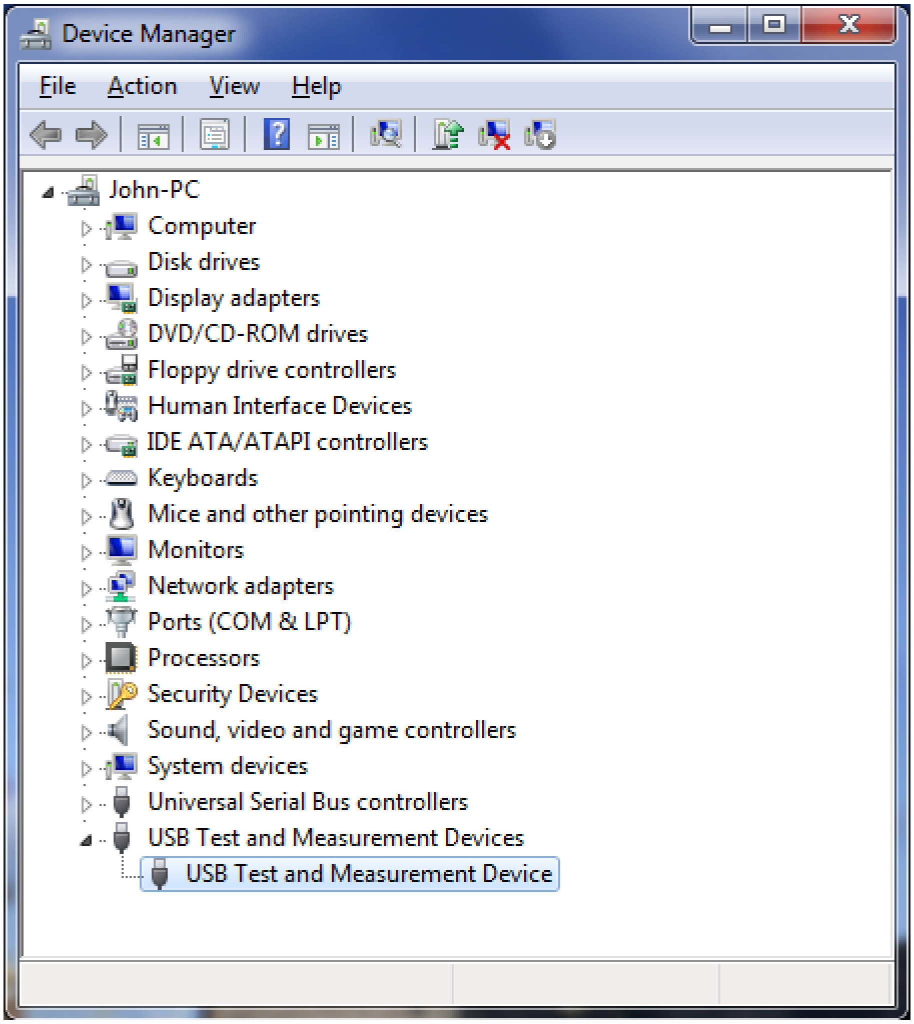
Quick Tip

From the Start menu, you can enter `Devmgmt.msc` in the Run box or the Windows 7 search box to start Device Manager.

2. Under USB Test and Measurement Devices, look for USB Test and Measurement Device.

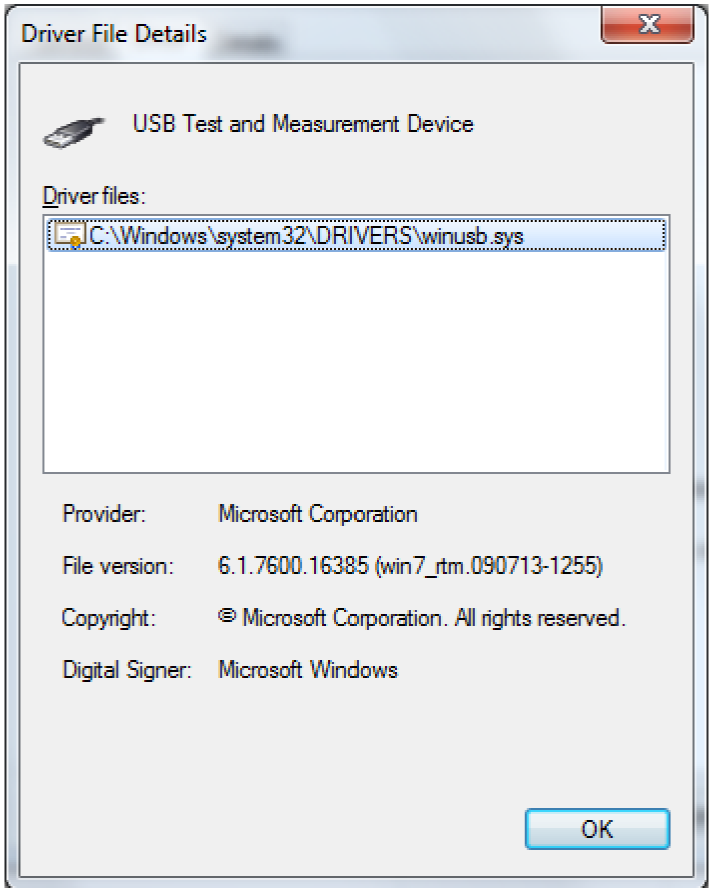
   If the device is not there, either VISA is not installed or the instrument is not plugged in and switched on.

Figure 142: Device Manager dialog box showing USB Test and Measurement Device



3. Right-click the device.
4. Select Properties.
5. Select the Driver tab.
6. Click **Driver Details**.
7. Verify that the device driver is the winusb.sys. driver from Microsoft.

Figure 143: Driver File Details dialog box



8. If the incorrect driver is installed, click **OK**.
9. On the Driver tab, click **Update Driver**.
10. Browse for the driver; select the C:\windows\inf folder and you should see the winusb.inf file. Select this and make sure the driver is now in use.
11. If this does not work, uninstall VISA, unplug the instrument and follow the steps to reinstall VISA in the section Modifying, repairing, or removing Keithley I/O Layer software.

Which Microsoft Windows operating systems are supported?

Microsoft Windows 2000, Windows XP, Windows Vista, and Windows 7 are supported.

What to do if the GPIB controller is not recognized?

If the hardware is not recognized by the computer:

1. Uninstall the software drivers.
2. Reboot the computer.
3. Check for newer drivers on the vendor's website. Check that the drivers are valid for the operating system you have and any updates that might be necessary. This information is typically found in the readme file that comes with the drivers.
4. Install software drivers.
5. Reboot the computer.
6. Plug in the hardware.

If it is still not recognized, you can try a different computer using a different operating system to rule out operating system issues.

If this does not resolve the issue, contact the vendor of the GPIB controller for assistance.

I'm receiving GPIB timeout errors. What should I do?

If your GPIB controller is recognized by the operating system, but you get a timeout error when you try to communicate with the instrument, check the following:

1. Confirm that the GPIB address you assigned to the instrument is unique and between the range of 0 to 30. It should not be 0 or 21 because they are common controller addresses.
2. Check cabling connection. GPIB cables are heavy and can fall out of the connectors if they are not screwed in securely.
3. Substitute cables to verify cable integrity. For example, if you can send and receive ASCII text, but you cannot do a binary transfer, check your program and the decoding of the binary data. If that does not resolve the problem, try another cable. ASCII text only uses seven data lines in the cable; the binary transfer requires all eight lines.

How do I change the command set?

You can change the command set that you use with the Model 2450. The remote command sets that are available include:

- SCPI: An instrument-specific language built on the SCPI standard.
- TSP: A programming language that can be used to send individual commands or combine commands into scripts.
- SCPI 2400: Allows you to run code developed for earlier Series 2400 instruments.

You cannot combine the command sets.

As delivered from Keithley Instruments, the Model 2450 is set to work with the Model 2450 SCPI command set.

NOTE

If you choose the SCPI2400 command set, you will not have access to some of the extended ranges and other features that are now available using the SCPI command set. In addition, some Series 2400 code will work differently in the Model 2450 than it did in the earlier instrument. See Model 2450 in a Model 2400 application (on page D-1) for information about the differences.

To set the command set from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Command Set.
4. Select the command set.
5. You are prompted to reboot.

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

Send the command:

```
*LANG SCPI
```

Reboot the instrument.

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

Send the command:

```
*LANG TSP
```

Reboot the instrument.

To change to the SCPI 2400 command set from a remote interface:

Send the command:

```
*LANG SCPI2400
```

Reboot the instrument after changing the command set.

To verify which command set is selected:

Send the command:

```
*LANG?
```

How do I upgrade the firmware?

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

From the front panel:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect any input and output terminals that are attached to the instrument.
4. Turn on instrument power.
5. Insert the flash drive into the USB port on the front panel of the instrument.
6. From the instrument front panel, press the **MENU** key
7. Under System, select **Manage**.
8. Select the type of upgrade you want to do:
   - To upgrade to a newer version of firmware: Select **Upgrade to New**.
   - To return to a previous version of firmware: Select **Downgrade to Older**.
9. If the instrument is controlled remotely, a message is displayed. Select **Yes** to continue.
10. When the upgrade is complete, reboot the instrument.

A message is displayed while the upgrade is in progress.

For additional information about upgrading the firmware, see Upgrading the firmware (on page A-3).

Where can I find updated drivers?

For the latest drivers and additional support information, see the Keithley Instruments support website.

To see what drivers are available for your instrument:

1.
2. Enter the model number of your instrument.
3. Select Software Driver from the list.

For LabVIEW™, you can also go to National Instrument's website and search their instrument driver database.

NOTE

LabVIEW can only be used with the SCPI command sets.

Why can't the Model 2450 read my USB flash drive?

Verify that the drive is formatted with the FAT file system. The Model 2450 only supports FAT drives.

In Windows, you can check the file system by checking the properties of the flash drive.

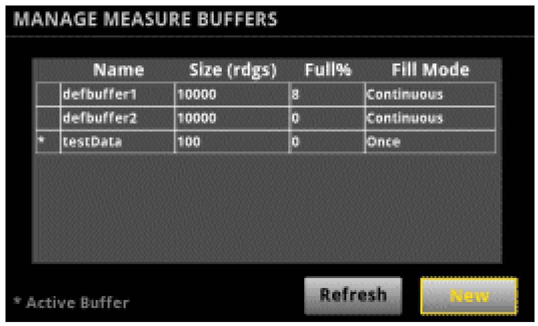
How do I download measurements onto the USB drive?

From the front panel, you can download measurements from a reading buffer to a .csv file on a USB flash drive.

Using the front panel to save or append buffer content to a file:

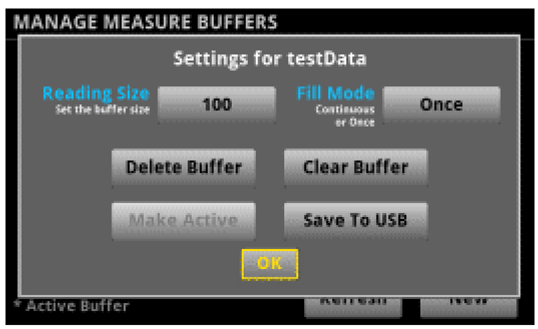
1. Insert a USB drive into the USB port.
2. Press the **MENU** key.
3. Under Measure, select **Data Buffers**. The MANAGE MEASURE BUFFERS window is displayed.

Figure 144: MANAGE MEASURE BUFFERS window



4. Select the reading buffer that you want to save.

Figure 145: MANAGE MEASURE BUFFERS



5. Select the **Save To USB** button.
6. Enter the name of the file in which to save the readings and select **ENTER**.

NOTE

You only have to enter the name of the file you want to save. You do not need to enter the file extension. All files are saved as `.csv` files.

When the MANAGE MEASURE BUFFERS window is displayed again, the file is saved.

7. Select **Yes** to confirm saving the file.

How do I save the present state of the instrument?

You can save the settings in the instrument using the front-panel menus or from a remote interface. After they are saved, you can recall them or copy them to a USB drive.

From the front panel:

1. Configure the Model 2450 to the settings that you want to save.
2. Press the **MENU** key.
3. Under Scripts, select **Create Config**. The CREATE CONFIG SCRIPTS window is displayed.
4. Select **Create**. A keyboard is displayed.
5. Use the keyboard to enter the name of the script.
6. Select **ENTER**. The script is added to internal memory.

Using SCPI commands:

Configure the instrument to the settings that you want to save. To save the setup, send the command:

```
*SAV <n>
```

Where `<n>` is an integer between 0 and 4.

NOTE

In the front panel script menus, the setups saved with the `*SAV` command have the name `Setup0x`, where *x* is the value you set for `<n>`.

Using TSP commands:

Configure the instrument to the settings that you want to save. To save the setup, send the command:

```
createconfigscript("setupName")
```

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

Why did my settings change?

Many of the commands in the Model 2450 are saved with the source or measurement function that was active when you set them. For example, assume you have the measurement function set to current and set a value for NPLCs. When you change the measurement function to voltage, the NPLC value changes to the value that was last set for the voltage measurement function. When you return to the current measurement function, the NPLC value returns to the value you set previously.

What is NPLC?

You can adjust the amount of time that the input signal is measured. This time affects the usable measurement resolution, the amount of reading noise, and the reading rate of the instrument.

The amount of time is specified in parameters that are based on the number of power line cycles (PLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50).

The shortest amount of time (0.01 PLC) results in the fastest reading rate, but increases reading noise and decreases the number of usable digits.

The longest amount of time (10 PLC) provides the lowest reading noise and more resolution, but has the slowest reading rate.

Settings between the fastest and slowest number of PLCs are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

What are the Quick Setup options?

The QUICKSET key opens a screen that provides predefined function, performance, and quick setups.

The Function button allows you to select the source and measurement functions. These are the same options that are available through the FUNCTION key

The Performance slider allows you to adjust speed and resolution. As you increase speed, you lower the amount of resolution. As you increase resolution, you decrease the reading speed. These settings take effect the next time the output is turned on and measurements are made.

The One-Touch Quick Setups allow you to set the instrument to operate as a Voltmeter, Ammeter, Ohmmeter, or Power Supply.

CAUTION

When you select a Quick Setup, the instrument turns the output on. Carefully consider and configure the appropriate output-off state, source, and limits before connecting the Model 2450 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure the settings that are recommended for the instrument before making connections to the device. Failure to consider the output-off state, source, and limits may result in damage to the instrument or to the device under test (DUT).

What is the output-off state?

When the source of the instrument is turned off, it may not completely isolate the instrument from the external circuit. You can use the output-off setting to place the Model 2450 in a known, noninteractive state during idle periods, such as when changing the device under test. The appropriate output-off state depends on your system and the device under test. Different types of connected devices or loads require different behaviors from the Model 2450 when the output is turned off. For example, a passive device such as a diode is not affected by a 0 V source connected across its terminals when the output is turned off. However, connecting a 0 V source to the terminals of a battery causes the battery to discharge.

The output-off states that can be selected for a Model 2450 are normal, high-impedance, zero, or guard.

⚠ CAUTION

Carefully consider and configure the appropriate output-off state, source, and source limits before connecting the Model 2450 to a device that can deliver energy, such as other voltage sources, batteries, capacitors, or solar cells. Configure recommended instrument settings before making connections to the device. Failure to consider the output-off state, source, and source limits may result in damage to the instrument or to the device under test (DUT).

When the Model 2450 is set to the normal output-off state, the following settings are made when the source is turned off:

- The measurement sense is set to 2-wire
- The voltage source is selected and set to 0 V
- The current limit is set to 10 percent of the full scale of the present current range
- OUTPUT OFF is displayed in the Home page Source area
- The Source button on the Home page shows the output that will be sourced when the output is turned on again

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

- The measurement sense is changed to 2-wire sense
- The source function is set to voltage
- The source voltage is set to 0
- Set the range to the presently selected range (turn off autorange)
- Program the voltage DAC to zero, and the current DAC to full scale of the present current range.
- When the zero output-off state is selected, you can use the instrument as an ammeter because it is outputting 0 V.

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

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

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

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

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

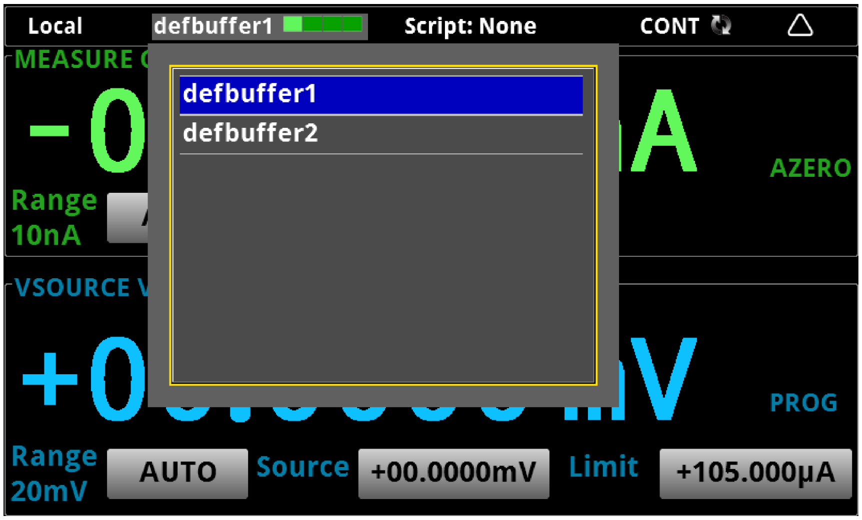
- The measurement sense is changed to 2-wire sense
- The current source is selected and set to 0 A
- The voltage limit is set to 10% full scale of the present voltage range

How do I store readings into the buffer?

Readings are automatically stored into default buffer 1 (`defbuffer1`).

To store readings into a different buffer, you can select another buffer from the buffer indicator on the home screen. Located to the right of the instrument active state indicator arrows, this indicator shows the name of the active reading buffer. Select the indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar. The green bar next to the buffer name indicates how full the buffer is.

Figure 146: Model 2450 active buffer indicator expanded



For more information on buffers, including information on creating a user-defined buffer, see Reading buffers (on page 3-10).

What should I do if I get an 5074 interlock error?

An interlock circuit is provided on the rear panel of the instrument. This circuit must be closed to enable the Model 2450 to produce voltages greater than 38 V DC. If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, you see event code 5074, "Output voltage limited by interlock."

WARNING

The Model 2450 is provided with an interlock circuit that must be positively activated in order for the high voltage output to be enabled. The interlock helps facilitate safe operation of the equipment in a test system. Bypassing the interlock could expose the operator to hazardous voltages that could result in personal injury or death.

If the safety interlock is not asserted and the source is turned on, the following actions occur:

- The nominal output is limited to ±37 V.
- The front-panel INTERLOCK indicator is not illuminated.

To recover from this error, properly engage the interlock using a safe test fixture before turning on the Model 2450 output.

You can only use the high-voltage outputs when the interlock is asserted. If you try to assign a high-voltage output and turn the source on when the interlock is not asserted, you see event code 5074, "Output voltage limited by interlock." Note that the SOURCE screen displays the value that was selected for the voltage source, but the source value is limited to ±21 V.

See Using the interlock (on page 2-70) for more information.

How do I trigger a sweep?

Sweeps are set up as a trigger model, so to start the sweep, initiate the trigger model. You can initiate the trigger model from the front panel by pressing the TRIGGER key.

What are source limits?

The source limits (also known as compliance) prevent the instrument from sourcing a voltage or current over a set value. This helps prevent damage to the device under test (DUT).

The values that can be set for the limits must be below the setting for the overvoltage protection limit.

This limit can also be restricted by the measurement range. If a specific measurement range is set, the limit must be more than 0.1 percent of the measurement range. If not, an event is generated and the limit is automatically changed to an appropriate value for the selected range. If you set the measurement range to be automatically selected, the measurement range does not affect the limit.

If you attempt to change the source limit to a value that is not appropriate for the selected source range, the source limit is not changed and a warning is generated. You must change the source range before you can select the new limit.

The lowest allowable limit is based on the load and the source value. For example, if you are sourcing 1 V to a 1 kohm resistor, the lowest allowable current limit is 1 mA (1 V/1 kohm = 1 mA). Setting a limit lower than 1 mA limits the source.

If the source output exceeds the source limit:

- On the Home screen, LIMIT is displayed to the right of the source voltage.
- The Source value changes to yellow.
- The Limit value on all swipe screens changes to yellow.

The source is clamped at the maximum limit value. For example, if the measurement limit is set to 1 V and the measurement range is 2 V, the output voltage is clamped at 1 V.

What is offset compensation?

Offset compensation is a measuring technique that reduces or eliminates thermal EMFs in low level resistance measurements. The voltage offsets because of the presence of thermal EMFs (V_{EMF}) can adversely affect resistance measurement accuracy.

To overcome these offset voltages, you can use offset-compensated ohms.

What is a configuration list?

A configuration list is a list of stored instrument settings. You can restore these instrument settings to change the active state of the instrument. Configuration lists allow you to record the active state of the instrument, store it, and then return the instrument to that state as needed.

If you are using TSP, configuration lists run faster than a script that is set up to configure the same settings.

The Model 2450 supports source configuration lists and measure configuration lists, making it possible to sequence through defined source settings, measurement settings, or both.

Each configuration list consists of a list of configuration points. A configuration point contains all instrument source or measure active settings at a specific point. You can cycle through the configuration points using a trigger model.

For more detail, see Configuration lists (on page 3-33).

Why do I see the "incompatible settings" message?

The "Continuous measurements have been terminated because of incompatible settings" message indicates that the combination of settings that are presently configured make it impossible for the instrument to make a valid measurement.

To resolve this problem, make changes to your settings.

How do I use the digital I/O port?

You can use the Model 2450 digital input/output with the trigger model or to control an external digital circuit, such as a device handler used to perform binning operations. To control or configure any of the six digital input/output lines, send commands to the Model 2450 over a remote interface.

To use the Model 2450 digital I/O in a trigger link system (TLINK), connect it using a Model 2450-TLINK Trigger Link Cable and configure the Model 2450 digital input and output lines.

For more information about the Model 2450 digital I/O port, see Digital I/O (on page 3-62).

How do I trigger other instruments?

You can use the Model 2450 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 2450 digital I/O port, see Digital I/O (on page 3-62).

You can also use the digital I/O in a trigger link system (TLINK) using a Model 2450-TLINK Trigger Link Cable.

Another option is Keithley Instruments TSP-Link®, 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 3-118) for additional information.

Section 10

Next steps

In this section:

Additional Model 2450 information

For additional information about the Model 2450, refer to:

- The Product Information CD-ROM (ships with the product): Contains software tools, drivers, and product documentation

  - The Knowledge Center, which contains the following handbooks:
    - *The Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*
    - *Semiconductor Device Test Applications Guide*
  - Application notes
  - Updated drivers
  - Updated firmware
  - Information about related products, including:
    - The Series 2600B System SourceMeter® Instruments
- Your local Field Applications Engineer: They can help you with product selection, configuration, and usage. Check the website for contact information.

Appendix A

Maintenance

In this appendix:

Introduction

The information in this section describes routine maintenance of the instrument that can be performed by the operator.

Line fuse replacement

A fuse located on the Model 2450 rear panel protects the power line input of the instrument.

WARNING

Disconnect the line cord at the rear panel and remove all test leads connected to the instrument before replacing the line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.

Use only the correct fuse type. Failure to do so could result in injury, death, or instrument damage.

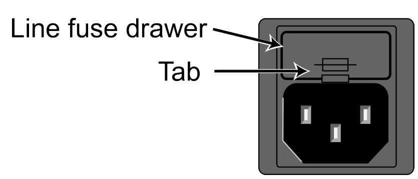
Use a 5 x 20 mm slow-blow fuse rated at 250 V, 2 A.

To replace the fuse, you will need a small flat-bladed screwdriver.

Perform the following steps to replace the line fuse:

1. Power off the instrument.
2. Remove the line cord.
3. Locate the fuse drawer, which is above the AC receptacle as shown in the graphic below.

Figure 147: Model 2450 line fuse



4. Use the screwdriver to lift the tab from the AC receptacle.
5. Slide the fuse drawer out. The fuse drawer does not pull completely out of the power module.
6. Snap the fuse out of the drawer.
7. Replace the fuse.
8. Push the fuse drawer back into the module.

If the power line fuse continues to blow, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

Front-panel display

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

Cleaning the front-panel display

If you need to clean the front-panel LCD touch screen display, use a soft dry cloth.

Abnormal display operation

If the display area is pushed hard during operation, you may see abnormal display operation. To restore normal operation, turn the instrument off and then back on.

Removing ghost images or contrast irregularities

If the display has been operating for a long time with the same display patterns, the display patterns may remain on the screen as ghost images and a slight contrast irregularity may appear. Note that if this occurs, it does not adversely affect the performance reliability of the display.

To regain normal operation, stop using the front-panel display for some time. You can turn off the front-panel display while continuing operation using remote commands.

To turn off the front-panel display using a SCPI command:

Send the command:

```
DISPlay:LIGHt:STATe OFF
```

To turn off the front-panel display using a TSP command:

Send the command:

```
display.lightstate = display.STATE_LCD_OFF
```

Upgrading the firmware

To upgrade the Model 2450 firmware, you load an upgrade file into the instrument. You can load the file from the USB port using the remote interface or the front panel of the instrument. If you are using Test Script Builder (TSB), you can upgrade the firmware from TSB using a file saved to the computer on which TSB is running.

During the upgrade process, the instrument verifies that the version you are loading is newer than what is on the instrument. If the version is older or at the same revision level, no changes are made.

If you want to revert to a previous version of the firmware, press the **MENU** key on the front panel, select **Manage**, and then select **Downgrade to Older** instead of Upgrade to New. When you return to a previous version, the instrument verifies that the version you are loading is earlier than what is on the instrument.

The upgrade process should take about five minutes.

To locate the upgrade files on the Keithley website:

1. Select the **Support** tab.
2. Search for your model number's firmware:
   a. In the model number box, type **2450**.
   b. Select **Firmware**.
   c. Click the search button.
3. A list of available firmware updates and any available documentation for the instrument is displayed. Click the desired file to download.

CAUTION

Disconnect the input and output terminals before you upgrade.

Do not remove power from the Model 2450 or remove the flash drive while an upgrade is in progress. Wait until the instrument completes the upgrade procedure and the opening display is shown.

From the front panel

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

From the front panel:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect any input and output terminals that are attached to the instrument.
4. Turn on instrument power.
5. Insert the flash drive into the USB port on the front panel of the instrument.
6. From the instrument front panel, press the **MENU** key
7. Under System, select **Manage**.
8. Select the type of upgrade you want to do:
   - To upgrade to a newer version of firmware: Select **Upgrade to New**.
   - To return to a previous version of firmware: Select **Downgrade to Older**.
9. If the instrument is controlled remotely, a message is displayed. Select **Yes** to continue.
10. When the upgrade is complete, reboot the instrument.

A message is displayed while the upgrade is in progress.

Using TSP

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

Using TSP over a remote interface:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect the input and output terminals that are attached to the instrument.
4. Power on the instrument.
5. Insert the flash drive into the USB port on the front panel of the instrument.
6. Send the command for the type of upgrade you want to do:
   - To upgrade to a newer version of firmware, send:
     `upgrade.unit()`
   - To return to a previous version of firmware, send:
     `upgrade.previous()`
7. After completion of the upgrade, reboot the instrument.

A message is displayed on the front panel of the instrument while the upgrade is in progress. In addition, the LEDs in the upper right of the front panel blink while the upgrade is in process.

Using SCPI

There are no SCPI commands that you can use to upgrade the firmware. To upgrade the firmware, you must either use the front panel or switch the command set to TSP.

To use the front panel to upgrade the firmware, see From the front panel (on page A-4).

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

If you need to upgrade the firmware from a remote interface and you are using one of the SCPI command sets, do the following steps:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect the input and output terminals that are attached to the instrument.
4. Power on the instrument.
5. Change the command set to TSP by sending the command:
   `*LANG TSP`
6. Turn the instrument off and then turn it on again.
7. Insert the flash drive into the USB port on the front panel of the instrument.
8. Send the command for the type of upgrade you want to do:
   - To upgrade to a newer version of firmware, send:
     `upgrade.unit()`
   - To return to a previous version of firmware, send:
     `upgrade.previous()`
9. After completion of the upgrade, turn the instrument off and then turn it on again.
10. To return to the SCPI 2400 command set, send the command:
    `*LANG SCPI2400`
11. To return to the SCPI 2450 command set, send the command:
    `*LANG SCPI`
12. Turn the instrument off and then turn it on again.

A message is displayed on the front panel of the instrument while the upgrade is in process. In addition, the LEDs in the upper right of the front panel blink while the upgrade is in process.

Using TSB

⚠ CAUTION

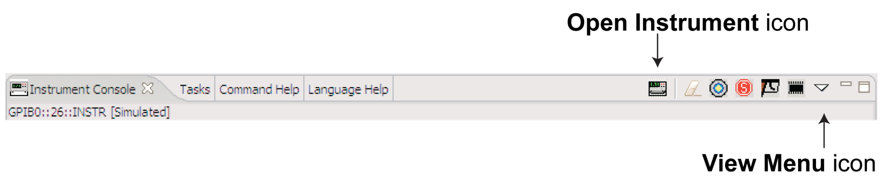
Do not turn off power or remove the USB flash drive until the upgrade process is complete.

After downloading an upgrade file from the Keithley Instruments website, you can use Test Script Builder (TSB) to upgrade the firmware of your instrument.

To upgrade the firmware using Test Script Builder:

1. Disconnect the input and output terminals that are attached to the instrument.
2. Start Test Script Builder.
3. On the Instrument Console toolbar, click the **Open Instrument** icon.

Figure 148: TSB Instrument Console toolbar



4. Select your communication interface from the Select Instrument dialog box. See the section on TSP Programming Fundamentals for details on opening communications.
5. On the Instrument Console toolbar, click the View Menu icon. Select **Instrument**, then select **Flash**.
6. From the Select a Firmware Image File dialog box, use the browser to select the file name of the new firmware or enter the path and file name.
7. Select the replacement mode:
   - Upgrade: Replace the existing firmware with a newer version of firmware.
   - Downgrade: Replace the existing firmware with an older version of firmware or repair the same version.
8. Click **OK**. A Progress Information bar is displayed on the instrument during the update. In addition, the LEDs in the upper right of the front panel blink while the upgrade is in process.
9. Wait until the instrument indicates that the firmware upgrade is complete. (TSB may indicate that the upgrade is complete before it is finalized on the instrument.)
10. Reboot the instrument.

Appendix B

Common commands

In this appendix:

Introduction

This section describes the general bus commands and common commands. Note that although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the Model 2450 does not strictly conform to that standard.

The general bus commands are commands that have the same general meaning, regardless of the instrument (for example, `DCL` always clears the GPIB interface and returns it to a known state).

The common commands perform operations such as reset, wait-to-continue, and status.

Common commands always begin with an asterisk (`*`) and may include one or more parameters. The command keyword is separated from the first parameter by a blank space.

If you are using the TSP remote interface, each command must be sent in a separate message.

If you are using a SCPI remote interface, the commands can be combined. Use a semicolon (`;`) to separate multiple commands, as shown below:

```
*RST; *CLS; *ESE 32; *OPC?
```

Although the commands in this section are shown in uppercase, common commands are not case sensitive (you can use either uppercase or lowercase).

NOTE

If you are using the TSP remote interface, note that the common commands cannot be used in scripts.

*CLS

This command clears the event registers and queues.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*CLS
```

Details

When the command language of the instrument is TSP, this command performs the same actions as sending `eventlog.clear()` and `status.clear()`. These actions are:

- Clear the event enable registers of the Questionable Event, Operation Event, and Status Byte Register sets.
- Clear the event log (which clears the Error Queue of the status model).

When the command language of the instrument is SCPI 2450, this command clears the event registers of the Questionable Event and Operation Event Register set. It also clears the event log. It does not affect the Questionable Event Enable or Operation Event Enable registers.

To reset all the bits of the Standard Event Enable Register, send the command:

```
*ESE 0
```

Also see

*ESE (on page B-2)
:STATus:PRESet (on page 6-92)

*ESE

This command sets and queries bits in the Status Enable register of the Standard Event Register.

Type	Affected by	Where saved	Default value
Command and query			See Details

Usage

```
*ESE <n>
*ESE?
```

`<n>`	The value of the Status Enable register of the Standard Event Register (0 to 255)

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or the value of the bit as the `<n>` parameter.

If you are using TSP, you can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *`standardRegister`* to the sum of their decimal weights. For example, to set bits B0 and B4, set *`standardRegister`* to 17 (which is the sum of 1 + 16). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.EXE
```

If you are using SCPI, you can only set the bit as a numeric value.

When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page B-6) command or TSP opc() (on page 8-77) function.
1	2	`Not used`	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	`status.standard.DDE`	An instrument operation did not execute properly due to an internal condition.
4	16	`status.standard.EXE`	The instrument detected an error while trying to execute a command.
5	32	`status.standard.CME`	A command error has occurred. See information following this table for descriptions of command errors.
6	64	`status.standard.URQ`	The instrument transitioned from remote control to local control.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

`*ESE 145`	`*ESE 145` sets the Status Enable register of the Standard Event Register to binary `10010001`, which enables the PON, EXE, and OPC bits. `*ESE?` might return the string `*ESE 186`, showing that the ESER contains the binary value `10111010`

Also see

*CLS (on page B-2)
Standard Event Register (on page C-3)
Status model (on page C-1)

*ESR?

This command reads and clears the contents of the Standard Event Status Register.

Type	Affected by	Where saved	Default value
Query only			

Usage

```
*ESR?
```

Details

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page B-6) command or TSP opc() (on page 8-77) function.
1	2	`Not used`	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	`status.standard.DDE`	An instrument operation did not execute properly due to an internal condition.
4	16	`status.standard.EXE`	The instrument detected an error while trying to execute a command.
5	32	`status.standard.CME`	A command error has occurred. See information following this table for descriptions of command errors.
6	64	`status.standard.URQ`	The instrument transitioned from remote control to local control.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

`*ESR?`	Might return the value `149`, showing that the Standard Event Status Register contains binary `10010101`

Also see

Status model (on page C-1)

*IDN?

This command retrieves the identification string of the instrument.

Type	Affected by	Where saved	Default value
Query only	None	Not applicable	Not applicable

Usage

```
*IDN?
```

Details

The identification string includes the manufacturer, model number, serial number, and firmware revision of the instrument. The string is formatted as follows:

```
KEITHLEY INSTRUMENTS INC.,MODEL nnnn,xxxxxxx,yyyyy
```

Where:

- `nnnn` is the model number
- `xxxxxxx` is the serial number
- `yyyyy` is the firmware revision level

Example

`*IDN?`	Output: `KEITHLEY INSTRUMENTS INC.,MODEL 2450,01234567,1.0.0i`

Also see

System information (on page 2-66)

*LANG

This command determines which command set is used by the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	SCPI

Usage

```
*LANG <commandSet>
*LANG?
```

`<commandSet>`	The command set to be used: • `TSP` • `SCPI` • `SCPI2400`

Details

The remote command sets that are available include:

- SCPI: An instrument-specific language built on the SCPI standard.
- TSP: A programming language that can be used to send individual commands or combine commands into scripts.
- SCPI 2400: Allows you to run code developed for earlier Series 2400 instruments.

You cannot combine the command sets.

Example

`*LANG TSP` `*LANG?`	Set the language to TSP. Verify setting by sending the language query. Output: `TSP` The TSP command set is in use.

Also see

Status model (on page C-1)

*OPC

This command sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	Affected by	Where saved	Default value
Command and query			

Usage

```
*OPC
*OPC?
```

Details

When `*OPC` is sent, the OPC bit (bit 0) in the Status Event Status Register is set after all pending command operations have been executed.

When `*OPC?` is sent, an ASCII "1" is placed in the output queue after all pending command operations have been executed.

Typically, either one of these commands is sent after the `INITiate` command. The `INITiate` command takes the instrument out of idle in order to perform measurements.

When the trigger model is executing, most sent commands are not executed. If a command cannot be processed, an error message is generated in the event log.

After all programmed operations are complete, the instrument returns to idle, at which time all pending commands (including `*OPC` and `*OPC?`) are executed. After the last pending command is executed, the OPC bit or an ASCII "1" is placed in the Output Queue.

Also see

:INITiate[:IMMediate] (on page 6-124)
opc() (on page 8-77)

*RST

This command resets the instrument settings to their default values.

Type	Affected by	Where saved	Default value
Command only			

Usage

```
*RST
```

Details

Returns the instrument to default settings, cancels all pending commands, and cancels the response to any previously received `*OPC` and `*OPC?` commands.

Also see

reset() (on page 8-82)

*SRE

This command sets or clears the bits of the Status Request Enable Register.

Type	Affected by	Where saved	Default value
Command only	None		

Usage

```
*SRE <n>
*SRE?
```

`<n>`	Clear the Status Request Enable Register: 0 Set the instrument for an SRQ interrupt: 32

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	`status.MSB`	An enabled event in the Measurement Event Register has occurred.
1	2	`Not used`	Not used.
2	4	`status.EAV`	An error or status message is present in the Error Queue.
3	8	`status.QSB`	An enabled event in the Questionable Status Register has occurred.
4	16	`status.MAV`	A response message is present in the Output Queue.
5	32	`status.ESB`	An enabled event in the Standard Event Status Register has occurred.
6	64	`Not used`	Not used.
7	128	`status.OSB`	An enabled event in the Operation Status Register has occurred.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

`*SRE 0`	Clear the bits of the Status Request Enable Register.

Also see

Understanding bit settings (on page C-15)

*STB?

This command gets the serial poll byte of the instrument without clearing the request service bit.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

`*STB?`

Details

The status byte query (`*STB?`) command is similar to a serial poll, but it is processed like any other instrument command.

The `*STB?` command returns the same result as a serial poll, but the request service bit (bit 6) is not cleared if a serial poll has occurred.

Example

`*STB?`	Queries the status byte.

Also see

None

*TRG

This command generates a trigger event from a remote command interface.

Type	Affected by	Where saved	Default value
Command only			

Usage

`*TRG`

Details

Use the `*TRG` command to generate a trigger event.

If you are using the SCPI command set, this command generates the COMMAND event. If you are using the TSP command set, this command generates the `trigger.EVENT_COMMAND` event. You can use this constant as the stimulus of any trigger object, which causes that trigger object to respond to the trigger events generated by `*TRG`. See Using trigger events to start actions in the trigger model (on page 3-109).

Also see

:INITiate[:IMMediate] (on page 6-124)

*TST?

This command is accepted and returns 0.

Type	Affected by	Where saved	Default value
Query only			0

Usage

```
*TST?
```

Also see

None

*WAI

This command postpones the execution of subsequent commands until all previous overlapped commands are finished.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*WAI
```

Details

There are two types of device commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `*WAI` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. The `*WAI` command is not needed for sequential commands.

Also see

waitcomplete() (on page 8-252)

Appendix C

Status model

In this appendix:

Overview

The status model consists of status register sets and queues. You can monitor the status model to view instrument events; you can also configure the status model to control the events.

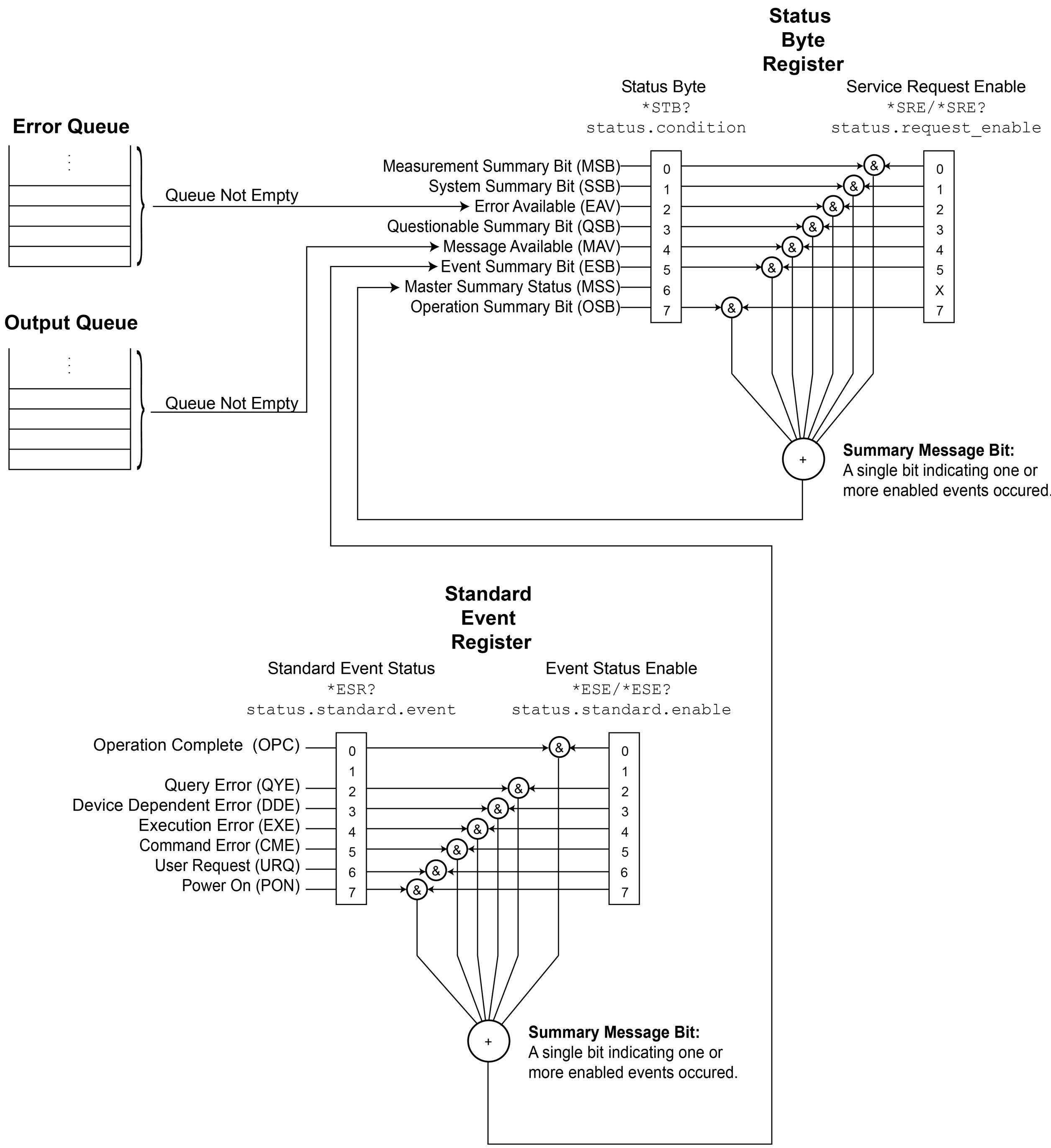
As you work with the status model, keep in mind the end result applies to the Status Byte Register. All the status register sets and queues flow into the Status Byte Register. Your test program can read this register to determine if a service request (SRQ) has occurred, and if so, which event caused it.

The Status Byte Register register sets and queues include:

- Standard Event Register
- Questionable Event Register
- Operation Event Register
- Output Queue
- Error Queue

The relationship between the Status Byte Register, Standard Event Register, event queue, and the output queue is shown in the Non-programmable status registers diagram (on page C-2). The relationship between the Status Byte Register, Questionable Event Register, and the Operation Event Register is shown in the Programmable status registers diagram (on page C-6).

Figure 149: Non-programmable status registers diagram

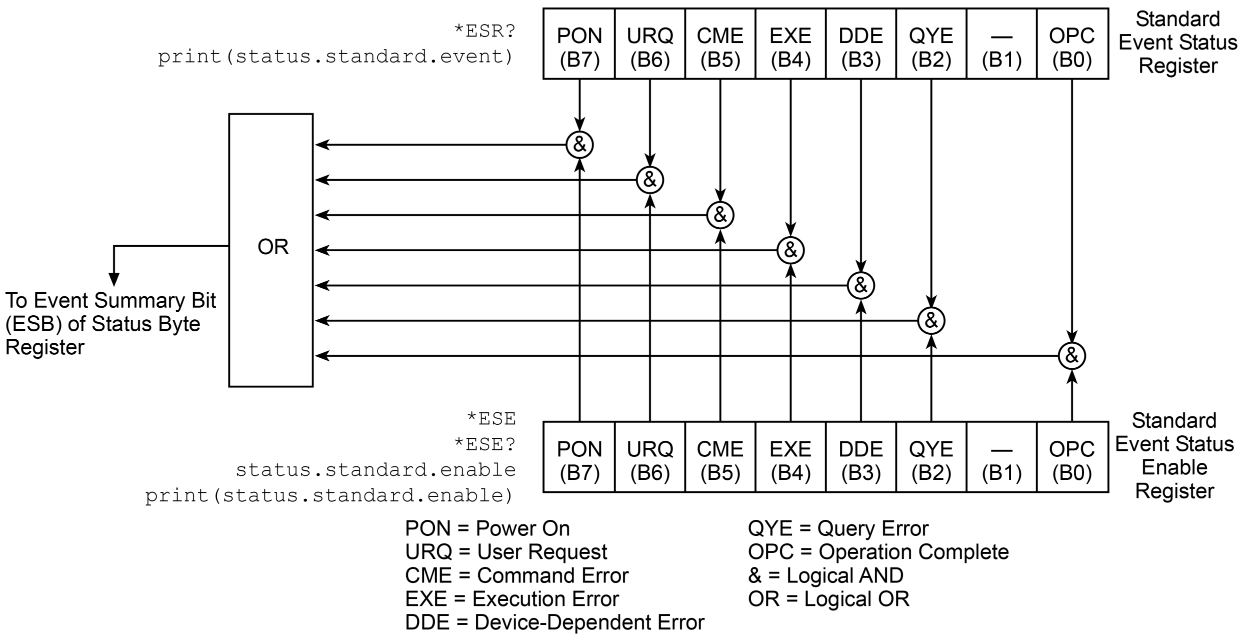


Standard Event Register

The Standard Event Register set includes two 8-bit registers:

- **Standard Event Status register:** Reports when a predefined event has occurred. The register latches the event and the corresponding bit remains set until it is cleared by a read.
- **Standard Event Status Enable register:** You can enable or disable bits in this register. This allows the predefined event (from the Standard Event Status Register) to set the ESB of the Status Byte Register.

Figure 150: Model 2450 Standard Event Register



Bit	When set, indicates the following has occurred:
0	Operation complete: All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page B-6) command or TSP opc() (on page 8-77) function.
1	Not used.
2	Query error: Attempt to read data from an empty Output Queue.
3	Device dependent error: An instrument operation did not execute properly due to an internal condition.
4	Execution error: The instrument detected an error while trying to execute a command.
5	Command error: A command error has occurred. Command errors include: • IEEE Std 488.2 syntax error: The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard. • Semantic error: The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument. • GET error: The instrument received a Group Execute Trigger (GET) inside a program message
6	User request: The instrument transitioned from remote control to local control.
7	Power-on: The instrument has been turned off and turned back on since the last time this register was read.

You can use the following commands to read and set bits contained in the Standard Event Register.

Description	SCPI command	TSP command
Read the Standard Event Status Register	*ESR? (on page B-4)	status.standard.event (on page 8-164)
Set or read the OR bits in the Standard Event Status Enable Register	*ESE (on page B-2)/ESE?	status.standard.enable (on page 8-162)

Programmable status register sets

You can program the registers in the Questionable Event Register and Operation Event Register sets.

These event registers contain bits that identify the state of an instrument condition or event. They also contain bits that determine if those events are sent to the Status Byte Register. You can enable the events which cause the associated bit to be set in the Status Byte Register.

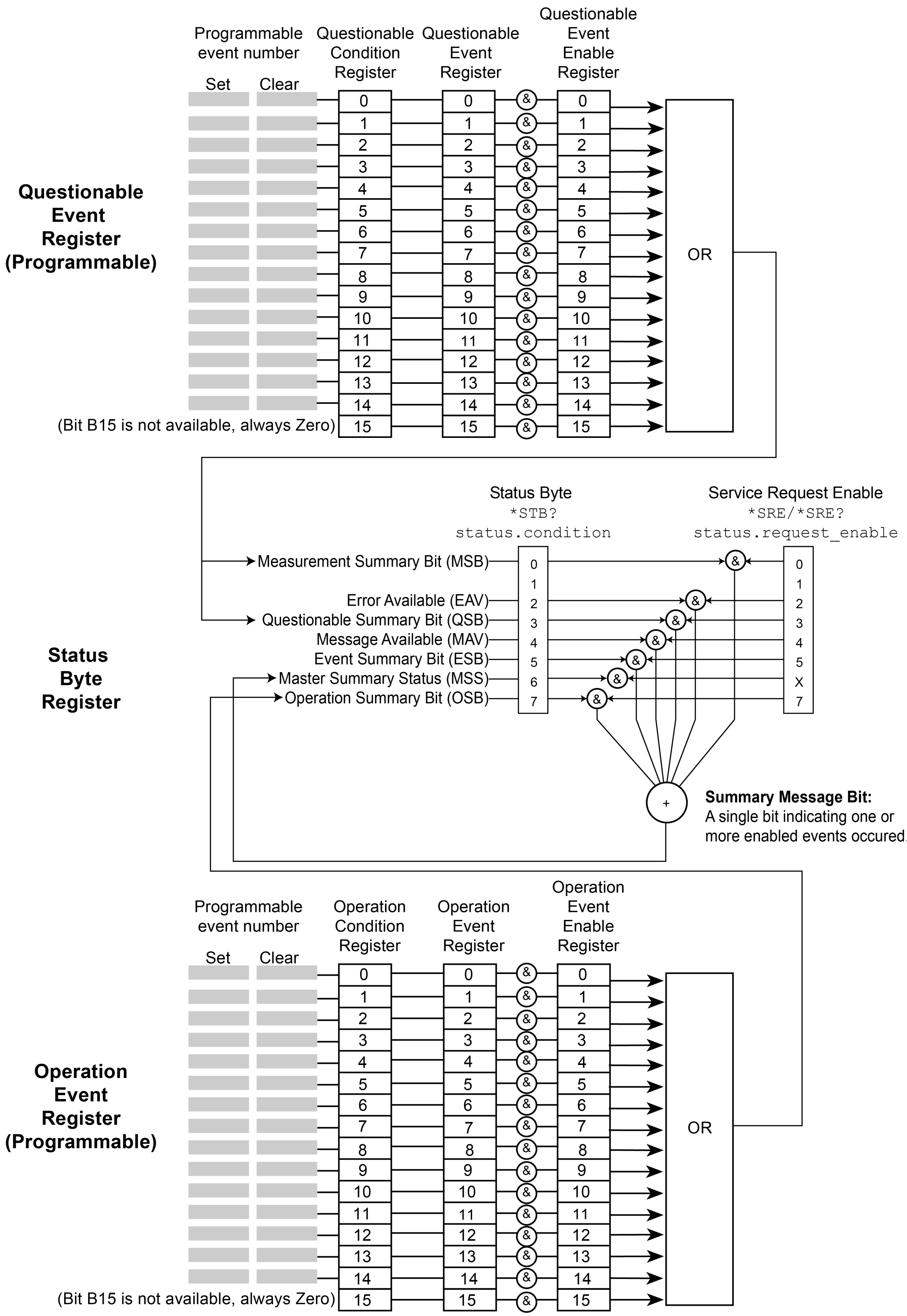
The Questionable and Operation Event Registers are identical except that they set different bits in the Status Byte Register. The Questionable Event Registers set both the MSB and QSM bits; the Operation Event Registers set the OSB bit.

Each 16-bit register set includes the following registers:

- **Condition:** A read-only register that is constantly updated to reflect the present operating conditions of the instrument. You can determine which events set or clear the bits.
- **Event:** A read-only register that sets a bit to 1 when an applicable event occurs. The bit remains at 1 until the register is reset. This register is reset when power is cycled or when a `*CLS` command is sent. You can determine which events set the bits.
- **Event enable:** A read-write register that determines which events set the summary bit in the Status Byte Register. For example, if a bit is a 1 in the event register and the corresponding bit is a 1 in the Event Enable Register, bits in the Status Byte Register are set. If the event enable bit is set in the Questionable Event Registers, the event sets the MSB and QSM bits in the Status Byte Register. If the event enable bit is set in the Operation Event Registers, the event sets the OSB bit in the Status Byte Register.

On power-up, all bits in the Questionable Event and Operation Event registers are set to 0.

Figure 151: Programmable status registers diagram



Questionable Event Register

You can program the bits in the Questionable Event register to be cleared or set when an event occurs.

When an enabled Questionable Event Register bit is set (because the enabled event occurs), the corresponding bit B0 (MSB) and Bit B3 (QSB) of the Status Byte Register is set. The corresponding Questionable Event Register Condition register reflects the present status of the instrument, so it will be set while the event occurs.

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Understanding bit settings (on page C-15).

You can use the following commands to read and set bits contained in the Questionable Event Register.

Description	SCPI command	TSP command
Read the Questionable Condition Register	:STATus:QUEStionable:CONDition? (on page 6-92)	status.questionable.condition (on page 8-158)
Set or read the contents of the Questionable Event Enable Register	:STATus:QUEStionable:ENABle (on page 6-93)	status.questionable.enable (on page 8-158)
Read the Questionable Event Register	:STATus:QUEStionable[:EVENt]? (on page 6-93)	status.questionable.event (on page 8-159)
Request the mapped set event and mapped clear event status for a bit in the Questionable Event Register	:STATus:QUEStionable:MAP (on page 6-94)	status.questionable.getmap() (on page 8-159)
Map events to bits in the Questionable Event Register	:STATus:QUEStionable:MAP (on page 6-94)	status.questionable.setmap() (on page 8-160)

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register. You can use a copy of the following table to record settings for your instrument.

Bit	Decimal value	Set Event	Clear Event	When set, indicates the following has occurred:
0	1			
1	2			
2	4			
3	8			
4	16			
5	32			
6	64			
7	128			
8	256			
9	512			
10	1024			
11	2048			
12	4096			
13	8192			
14	16 384			

Operation Event Register

You can program the bits in the Operation Condition and Operation Event Status registers to be cleared or set when an event occurs.

When an enabled Operation Event Register bit is set, the corresponding bit B7 (OSB) of the Status Byte Register is set.

When an enabled Operation Event Register bit is set (because the enabled event occurs), the corresponding bit B7 (OSB) of the Status Byte Register is set. The corresponding Operation Event Register Condition register reflects the present status of the instrument, so it will be set while the event occurs.

You can use the following commands to read and set bits contained in the Operation Event Register.

Description	SCPI command	TSP command
Read the Operation Event Register	:STATus:OPERation:CONDition? (on page 6-89)	status.operation.condition (on page 8-155)
Set or read the contents of the Operation Event Enable Register	:STATus:OPERation:ENABle (on page 6-89)	status.operation.enable (on page 8-155)
Read the Operation Event Register	:STATus:OPERation[:EVENt]? (on page 6-90)	status.operation.event (on page 8-156)
Request the mapped set event and mapped clear event status for a bit in the Operation Event Registers	:STATus:OPERation:MAP (on page 6-91)	status.operation.getmap() (on page 8-156)
Map events to bits in the Operation Event Registers	:STATus:OPERation:MAP (on page 6-91)	status.operation.setmap() (on page 8-157)

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register. You can use a copy of the following table to record settings for your instrument.

Bit	Decimal value	Set Event	Clear Event	When set, indicates the following has occurred:
0	1			
1	2			
2	4			
3	8			
4	16			
5	32			
6	64			
7	128			
8	256			
9	512			
10	1024			
11	2048			
12	4096			
13	8192			
14	16 384			

Mapping events to bits

To program the Questionable and Operation Event Registers, you map events to specific bits in the register. This causes a bit in the condition and event registers to be set (or cleared) when the specified event occurs. You can map events to bits B0 through B14 (bit B15 is always set to zero).

When you have a mapped-set event, the bits in the corresponding condition register and event register are set when the mapped-set event is detected. The bits remain at 1 until the event register is read or the status model is reset.

When you have a mapped-clear event, the bit in the condition register is cleared to 0 when the event is detected.

You can map any event to any bit in these registers. An event is the number that accompanies an error, warning, or informational message that is reported in the event log. For example, for the error "Error -221, Settings Conflict," the event is –221. Note that some informational messages do not have a related event number, so they cannot be mapped to a register.

You do not need to map clear events to generate SRQs. However, if you want to read the condition register to report status, you must map both a set event and a clear event. If no clear event is mapped, the bits are cleared only when the instrument power is turned off and turned on.

You can use the following SCPI commands to read and map events to bits in the programmable registers:

- :STATus:OPERation:MAP (on page 6-91)
  This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.
- :STATus:QUEStionable:MAP (on page 6-94)
  This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.

You can use the following TSP commands to read and map events to bits in the programmable registers:

- status.operation.getmap() (on page 8-156)
  This command reads the mapped set and clear status for the specified operation event bit.
- status.operation.setmap() (on page 8-157)
  This command maps the set and clear events to a specified operation event register bit.
- status.questionable.getmap() (on page 8-159)
  This command reads the mapped set and clear status for the specified questionable event bit.
- status.questionable.setmap() (on page 8-160)
  This command maps the set and clear events to a specified questionable event register bit.

You can map any event that appears with a number in the event queue to any available bit in a programmable register. The programmable registers and their relationships to the Status Byte Register are shown in the Programmable status registers diagram (on page C-6). The following example event queue log entries contain actual events that can be mapped to a status model bit.

```
2731 Trigger Model Initiated "Trigger model #1 has been initiated"
2732 Trigger Model Idle  "Trigger model #1 has been idled"
4917 Reading buffer cleared "Reading buffer <buffer name> is  0% filled"
4918 Reading buffer full "Reading buffer <buffer name> is  100% filled"
5080 SMU Source Limit Tripped "Source limiting is active on output"
5081 SMU Source Limit Cleared "Source limiting is no longer necessary and
output is normal"
```

See Using the event log (on page 2-117) for additional information on finding events.

Status Byte Register

The Status Byte Register monitors the registers and queues in the status model and generates service requests (SRQs).

When bits are set in the status model registers and queues, they generate summary messages that set or clear bits of the Status Byte Register. You can enable these bits to generate an SRQ.

Service requests (SRQs) instruct the controller that the instrument needs attention or that some event has occurred. When the controller receives an SRQ, the controller can interrupt existing tasks to perform tasks that address the request for service.

For example, you might program your instrument to send an SRQ when a specific instrument error occurs. To do this, you set the Status Request Enable bit 2 (EAV). In this example, the following actions occur:

- The errors occurs.
- The error is logged in the Error Queue.
- The Error Queue sets the EAV bit of the Status Byte Register.
- The EAV bits are summed.
- The RQS bit of the Status Byte Register is set.
- On a GPIB system, the SRQ line is asserted. On a VXI-11 or USB connection, an SRQ event is generated.

For an example of this, see the example code provided in SRQ on error (on page C-22).

The summary messages from the status registers and queues are used to set or clear the appropriate bits (B0, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are solely dependent on the summary messages (0 or 1). For example, if the Standard Event Register is read, its register will clear. As a result, its summary message will reset to 0, which in turn will reset the ESB bit in the Status Byte Register.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status (MSS) bit.

When using the GPIB, USB, or VXI-11 serial poll sequence of the Model 2450 to get the status byte (serial poll byte), bit B6 is the RQS bit. See Serial polling and SRQ (on page C-13) for details on using the serial poll sequence.

When using the `*STB?` common command or `status.condition` command to read the status byte, bit B6 is the MSS bit.

To reset the bits of the Service Request Enable Register to 0, use 0 as the parameter value for the command (for example, `*SRE 0` or `status.request_enable = 0`).

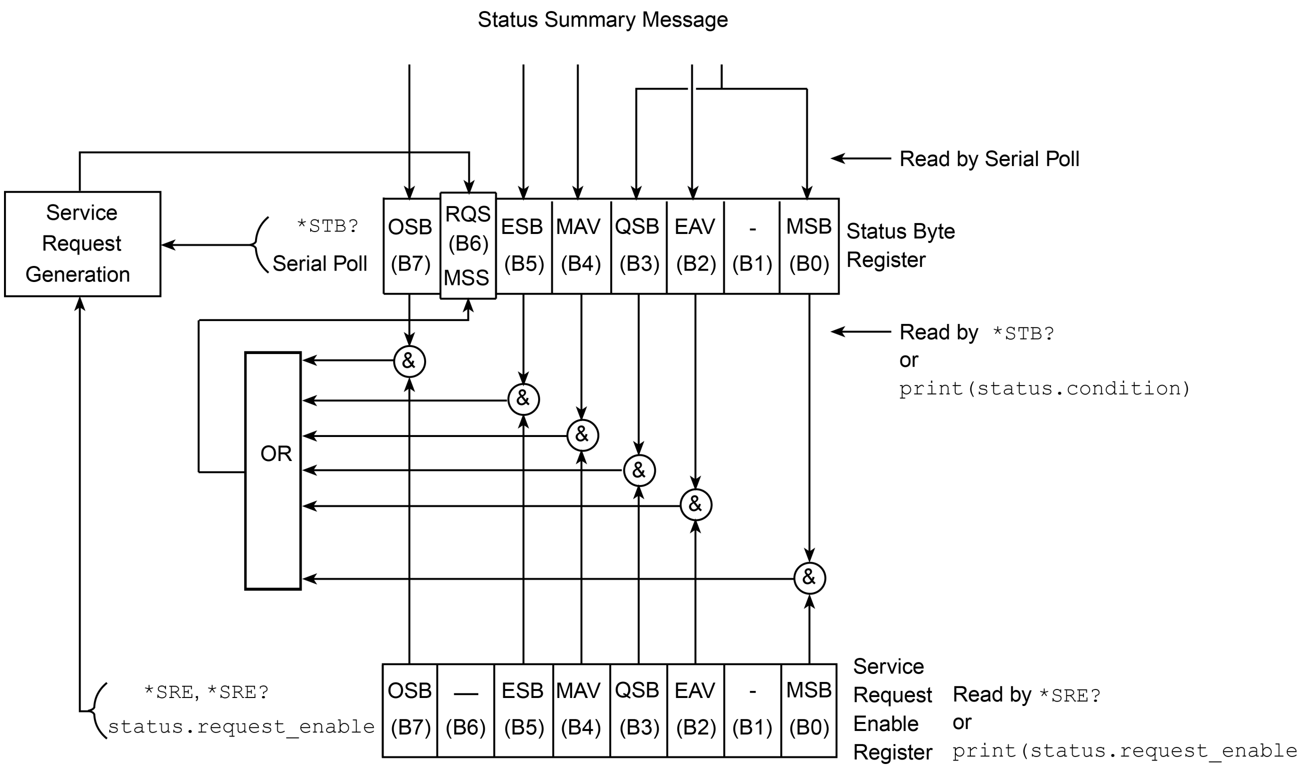
You can read and set which bits to AND in the Status Byte Register using the following commands.

Description	SCPI command	TSP command
Read the Status Byte Register	*STB? (on page B-9)	status.condition (on page 8-154)
Read the Status Request Enable Register	*SRE (on page B-8)?	status.request_enable (on page 8-160)
Enable bits in the Status Request Enable Register	*SRE (on page B-8)	status.request_enable (on page 8-160)

Status Byte Register diagram

The Status Byte Register consists of two 8-bit registers that control service requests, the Status Byte Register and the Service Request Enable Register. These registers are shown in the following figure.

Figure 152: Model 2450 Status Byte Register



The bits in the Status Byte Register are described in the following table.

Bit	Decimal value	Bit name	When set, indicates the following has occurred:
0	1	Measurement summary Bit (MSB)	An enabled questionable event
1	2	Not used	Not applicable
2	4	Error available (EAV)	An error is present in the error queue (warning and information messages do not affect this bit)
3	8	Questionable summary bit (QSB)	An enabled questionable event
4	16	Message available (MAV)	A response message is present in the output queue
5	32	Event summary bit (ESB)	An enabled standard event
6	64	Request for service (RQS)/Master summary status (MSS)	An enabled summary bit of the Status Byte Register is set; depending on how it is used, this is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit
7	128	Operation summary bit (OSB)	An enabled operation event

Service Request Enable Register

This register is programmed by the user and is used to enable or disable the setting of bit B6 (RQS/MSS) by the Status Summary Message bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in the Status Byte Register topic, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic "1" output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

The individual bits of the Service Request Enable Register can be set or cleared by using the `*SRE` common command or `status.request_enable`. To read the Service Request Enable Register, use the `*SRE?` query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a `*SRE 0` or `status.request_enable = 0` is sent). You can program and read the SRQ Enable Register using the following commands.

Description	SCPI command	TSP command
Read the Status Request Enable Register	*SRE (on page B-8)?	status.request_enable (on page 8-160)
Enable bits in the Status Request Enable Register	*SRE (on page B-8)	status.request_enable (on page 8-160)

Queues

The instrument includes an Output Queue and an Error Queue. The Output Queue holds messages from readings and responses. The Error Queue holds error messages from the event log. Both are first-in, first-out (FIFO) registers.

Output Queue

The output queue holds response messages to query and `print()` commands.

When data is placed in the Output Queue, the Message Available (MAV) bit in the Status Byte Register is set. The bit is cleared when the Output Queue is empty.

To clear data from the Output Queue, read the messages. To read a message from the Output Queue, address the instrument to talk after the appropriate query is sent.

Error Queue

The Error Queue holds any errors that are posted in the event log. When an error occurs, it is posted to the Error Queue, which sets the Error Available (EAV) bit in the Status Byte Register.

The instrument clears error messages from the event log when it retrieves the event log. When the error messages are cleared from the event log, the EAV bit in the Status Byte Register is cleared.

You can clear the Error Queue by sending the common command `*CLS` or the TSP command `status.clear()`. Note that `status.clear()` also clears all event registers.

For information regarding the event log, see Using the event log (on page 2-117).

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 has occurred and what caused it. If an SRQ occurs, the program can, for example, branch to an appropriate subroutine that will service the request.

SRQs can be managed by the serial poll sequence of the 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.

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 stays set until all Status Byte Register summary bits are reset.

For information on serial polling on a GPIB system, see SPE, SPD (on page 2-50).

Programming enable registers

You can program the bits in the enable registers of the Status Model registers.

When you program an enable register bit to 0, no action occurs if the bits in the corresponding registers are set (1).

When you program an enable register bit to 1, if the bits in the corresponding registers are set (1), the AND condition occurs and a bit in the Status Byte Register is set to (1).

You must program all bits in an enable register at the same time. This means you need to determine what each bit value in the register will be, then add them together to determine the value of all the bits in the register. See Understanding bit settings (on page C-15) for more information on determining the value of the bits in the registers.

For example, you might want to enable the Standard Event Register to set the ESB bit in the Status Byte Register whenever an operation complete occurs or whenever an operation did not execute properly because of an internal condition. To do this, you need to set bits 0 and 3 of the Standard Event Register to 1. These bits have decimal values of 1 and 8, so to set both bits to 1, you set the register to 9.

In SCPI, the command you would send is:

```
*ese 9
```

In TSP, the command you would send is:

```
status.standard.enable = 9
```

Reading the registers

You can read any register in the Status Model. The response is a decimal value that indicates which bits in the register are set. See Understanding bit settings (on page C-15) for information on how to convert the decimal value to bits.

Using SCPI commands:

If you are using SCPI, you use the query commands in the STATus subsystem and common commands to read registers.

Using TSP commands:

If you are using TSP, you print the TSP command to read the register. You can use either `print()`, which returns the decimal value, or `print(tostring())`, which returns the string equivalent of the decimal value.

You can also send the common commands to read the register.

For example, you can send any one of the following commands to read the Status Enable register of the Standard Event Register:

```
print(status.standard.enable)
*ese?
print(tostring(status.standard.enable))
```

Understanding bit settings

When you write to or read a status register, you can use binary, decimal, or hexadecimal values to represent the binary values of the bit states. When the value is converted to its binary equivalent, you can determine which bits are set on or clear. Zero (0) indicates that all bits are clear.

In the Model 2450, the least significant bit is always bit B0. The most significant bit differs for each register, but in most cases is either bit B7 or bit B15.

Bit position	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	1000 0000	0100 0000	0010 0000	0001 0000	1000	0100	0010	0000
Decimal value	128	64	32	16	8	4	2	1
Weight	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Bit position	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	1000 0000 0000 0000	0100 0000 0000 0000	0010 0000 0000 0000	0001 0000 0000 0000	1000 0000 0000	0100 0000 0000	0010 0000 0000	0001 0000 0000
Decimal value	32768	16384	8192	4096	2048	1024	512	256
Weight	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

For example, if a value of `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 and all other bits are cleared.

For example, if you read a value of 1.22880e+04 (12,288) for the condition register, the binary equivalent is 0011 0000 0000 0000. This value indicates that bits B12 and B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

When bit B12 (4096) and bit B13 (8192) are set (1), the decimal equivalent is 4096 + 8192 = 12,288.

Clearing registers

Registers in the status model can be cleared using commands or by instrument actions. When a registered is cleared, the bits in the register are set to 0.

The event log and all registers are cleared when instrument power is cycled.

Using SCPI commands:

To clear the event registers of the Questionable Event and Operation Event Register set, and the event log, send:

```
*CLS
```

When using the SCPI interface, this command does not affect the Questionable Event Enable or Operation Event Enable registers.

Using TSP commands:

To clear the event enable registers of the Questionable Event, Operation Event, and Status Byte Register sets, and clear the event log (which clears the Error Queue of the status model), send:.

```
*CLS
```

To clear the Questionable Event register, the Operation Event register sets, and the event log, send:

```
status.clear()
```

When using the TSP interface, this command does not affect the Questionable Event Enable or Operation Event Enable register sets.

Status model programming examples

The following examples illustrate how to generate an SRQ using the status model.

SRQ when the SMU reaches its source limit

This example demonstrates how to generate an SRQ when the SMU detects it has reached its source limit. After configuring the status model, in this example the source will be configured to output current and measure voltage. If the output terminals of the SMU are left open when running this example then the SMU will reach its source limit and generate an SRQ.

Using SCPI commands:

```
*RST
STAT:CLE
STAT:OPER:MAP 0, 5080, 5081
STAT:OPER:ENAB 1
*SRE 128
SOUR:FUNC CURR
SOUR:CURR:RANG 1e-3
SOUR:CURR 1e-3
SOUR:CURR:VLIM 1
SENS:FUNC "VOLT"
OUTP ON
READ?
OUTP OFF
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()

-- Map the event numbers
-- Map bit 0 of operational status register to set on reaching the
-- source limit (5080) and clear on dropping below the source
-- limit (5081)
status.operation.setmap(0, 5080, 5081)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB

-- This code will make SMU reach V limit if output terminals are open
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.range = 1e-3
smu.source.level = 1e-3
smu.source.vlimit.level = 1
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.source.output = 1
print(smu.measure.read())
smu.source.output = 0
```

SRQ when Trigger Model is finished

This example shows you how to generate an SRQ when the trigger model is completed and the SMU has returned to the Idle state. After configuring the status model, this code will configure and run the trigger model. When the trigger model completes, the instrument will generate an SRQ and the data will be returned.

Using SCPI commands:

```
*RST
TRAC:CLE
STAT:CLE
STAT:OPER:MAP 0, 2732, 2731
STAT:OPER:ENAB 1
*SRE 128
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
TRIG:BLOC:BUFF:CLE 1
TRIG:BLOC:SOUR:STAT 2, ON
TRIG:BLOC:DEL:CONS 3, 100e-3
TRIG:BLOC:MEAS 4, "defbuffer1"
TRIG:BLOC:BRAN:COUN 5, 9, 3
TRIG:BLOC:SOUR:STAT 6, OFF
INIT
*WAI
TRAC:DATA? 1, 9, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the reading buffer
defbuffer1.clear()

-- Clear the status byte
status.clear()

-- Map bit 0 of operational status register to set on trigger
-- model exit (2732) and clear on trigger model enter (2731).
status.operation.setmap(0, 2732, 2731)

-- Enable bit 0 to flow through to the status byte
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB

-- Configure a simple measurement loop using the trigger model
smu.source.level = 1
smu.source.ilimit.level = 10e-3

-- Configure a simple trigger model to take 10 measurements
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, 1)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 100e-3)
trigger.model.setblock(4, trigger.BLOCK_MEASURE, defbuffer1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 9, 3)
trigger.model.setblock(6, trigger.BLOCK_SOURCE_OUTPUT, 0)

-- Start the trigger model
trigger.model.initiate()
-- Instrument will generate an SRQ when done

waitcomplete()
printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ on Trigger Model Notify Event

This example shows you how to use the trigger model's Log Event block to generate an SRQ. This example configures the trigger model to perform a sweep and to repeat that sweep multiple times. The Log Event block is used to generate events to indicate when a sweep starts and when it ends. An SRQ is generated each time the sweep ends. This example is most useful when you are gathering several sweeps of data on a device and you want to notify the controlling computer when each sweep has completed so it can retrieve the data from the sweep without interrupting the test.

Using SCPI commands:

```
*RST
SOUR:CONF:LIST:CRE "sourceList"
SOUR:VOLT:RANG 10
SOUR:VOLT:ILIM 10e-3
SENS:NPLC 1
SENS:CURR:RANG 10e-3
SENS:AZER:ONCE
SOUR:VOLT 1
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 2
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 3
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 4
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 5
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 6
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 7
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 8
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 9
SOUR:CONF:LIST:STORE "sourceList"
SOUR:VOLT 10
SOUR:CONF:LIST:STORE "sourceList"
TRAC:CLE
TRIG:BLOC:CONF:RECALL 1, "sourceList"
TRIG:BLOC:SOUR:STAT 2, ON
TRIG:BLOC:LOG:EVEN 3, INFO1, "Sweep started."
TRIG:BLOC:BRAN:ONCE 4, 6
TRIG:BLOC:CONF:NEXT 5, "sourceList"
TRIG:BLOC:DEL:CONS 6, 1e-3
TRIG:BLOC:MEAS 7, "defbuffer1"
TRIG:BLOC:BRAN:COUN 8, 11, 5
TRIG:BLOC:LOG:EVEN 9, INFO2, "Sweep complete."
TRIG:BLOC:BRAN:COUN 10, 5, 3
TRIG:BLOC:SOUR:STAT 11, OFF
STAT:CLE
STAT:OPER:MAP 0, 2735, 2734
STAT:OPER:ENAB 1
*SRE 128
INIT
*WAI
TRAC:DATA? 1, 55, "defbuffer1", READ
```

Using TSP commands:

```
reset()
smu.source.configlist.create("sourceList")
smu.source.range = 10
smu.source.ilimit.level = 10e-3

smu.measure.nplc = 1
smu.measure.range = 10e-3
smu.measure.autozero.once()
for i=0, 10 do
   smu.source.level = i
   smu.source.configlist.store("sourceList")
end

defbuffer1.clear()

-- Configure the Trigger Model
trigger.model.setblock(1, trigger.BLOCK_CONFIG_RECALL, "sourceList")
trigger.model.setblock(2, trigger.BLOCK_SOURCE_OUTPUT, 1)
trigger.model.setblock(3, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "Sweep
   started.")
trigger.model.setblock(4, trigger.BLOCK_BRANCH_ONCE, 6)
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "sourceList")
trigger.model.setblock(6, trigger.BLOCK_DELAY_CONSTANT, 1e-3)
trigger.model.setblock(7, trigger.BLOCK_MEASURE, defbuffer1)
trigger.model.setblock(8, trigger.BLOCK_BRANCH_COUNTER, 11, 5)
trigger.model.setblock(9, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Sweep
   complete.")
trigger.model.setblock(10, trigger.BLOCK_BRANCH_COUNTER, 5, 3)
trigger.model.setblock(11, trigger.BLOCK_SOURCE_OUTPUT, 0)
```

```
-- Configure the Status Model
--=======================
-- Clear the status byte
status.clear()

-- Map the notify messages to operational register bit 0.
-- The Sweep Complete message will set the bit while the Sweep
-- Started message will clear the bit.
status.operation.setmap(0, trigger.LOG_INFO2, trigger.LOG_INFO1)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master Summary Bit/SRQ
status.request_enable = status.OSB

-- Start the trigger model
trigger.model.initiate()

waitcomplete()
print(string.format("Number of Readings in Buffer: %d", defbuffer1.n))
printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ on error

This example shows you how to generate an SRQ when an instrument error occurs.

Using SCPI commands:

```
*RST
SYST:CLE
STAT:CLE
*SRE 4
MAKEERROR
```

Using TSP commands:

```
reset()
-- Clear Error Queue so EAV bit can go low.
errorqueue.clear()

-- Clear the status byte
status.clear()

-- Enable SRQ on error available
status.request_enable = status.EAV

-- Send a line of code that will generate an error
smu = 1
```

SRQ when reading buffer becomes full

This example shows you how to generate an SRQ when the Model 2450 reading buffer is full. This is useful to notify the controlling computer that it needs to read back the data and empty the buffer. After configuring the status model, this code configures the default reading buffer 1 to a size of 100, and then configures the SMU to fill the buffer. After the buffer is full, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST
STAT:CLE
STAT:OPER:MAP 0, 4918, 4917
STAT:OPER:ENAB 1
*SRE 128
TRAC:CLE
TRAC:POIN 100, "defbuffer1"
SOUR:VOLT:RANG 1
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
COUNT 100
OUTP ON
READ? "defbuffer1"
OUTP OFF
TRAC:DATA? 1, 100, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()

-- Map bit 0 of operational status register to set on buffer
-- full (4918) and clear on buffer empty (4917).
status.operation.setmap(0, 4918, 4917)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB

-- Clear the buffer and make it smaller
defbuffer1.clear()
defbuffer1.capacity = 100
smu.source.range = 1
smu.source.level = 1
smu.source.ilimit.level = 10e-3

-- Set the measure count to fill the buffer
smu.measure.count = 100
smu.measure.range = 10e-3
smu.source.output = smu.ON
smu.measure.read(defbuffer1)
smu.source.output = smu.OFF

printbuffer(1, defbuffer1.n, defbuffer1)
```

SRQ when a measurement completes

This example shows you how to generate an SRQ when a measurement completes. This is most useful when you have a measurement that will take a long time to complete and you wish to free up the controlling computer to do other things while it is waiting. This can happen if you have configured the measurement with a long delay value, a large aperture, or have enabled filtering.

This code configures the SMU for a long reading, and then configures the trigger model to generate notify events before and after the measurement. The status model is then configured to generate an SRQ based on the "Measurement Done" notify event. The output is turned on and the trigger model is used to take the measurement. When the measurement completes, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST
TRAC:CLE
SOUR:VOLT:RANG 1
SOUR:VOLT 1
SOUR:VOLT:ILIM 10e-3
SENS:CURR:RANG 10e-3
SENS:NPLC 10
TRIG:BLOC:LOG:EVEN 1, INFO1, "Measurement Started."
TRIG:BLOC:MEAS 2, "defbuffer1"
TRIG:BLOC:LOG:EVEN 3, INFO2, "Measurement Done."
STAT:CLE
STAT:OPER:MAP 0, 2735, 2734
STAT:OPER:ENAB 1
*SRE 128
OUTP ON
INIT
*WAI
OUTP OFF
TRAC:DATA? 1, 1, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the reading buffer
defbuffer1.clear()

-- Configure the source and take a measurement
smu.source.range = 1
smu.source.level = 1
smu.source.ilimit.level = 10e-3

smu.measure.range = 10e-3

-- Setup the NPLC for a really long measurement
smu.measure.nplc = 10

-- Use the trigger model to create events before and after
-- the measurement to generate SRQ when measurement is done.
trigger.model.setblock(1, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO1, "Measurement
   Started.")
trigger.model.setblock(2, trigger.BLOCK_MEASURE, defbuffer1)
trigger.model.setblock(3, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Measurement
   Done.")
```

```
-- Clear the status byte
status.clear()

-- Map bit 0 of the Operation Event Register to set on the Measurement
-- Done log notification (trigger.LOG_INFO2) and clear on the
-- Measurement Started log notification (trigger.LOG_INFO1).
status.operation.setmap(0, trigger.LOG_INFO2, trigger.LOG_INFO1)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master Summary
-- Bit/RQS
status.request_enable = status.OSB

smu.source.output = smu.ON
trigger.model.initiate()
waitcomplete()
smu.source.output = smu.OFF

printbuffer(1, defbuffer1.n, defbuffer1)
```

Model 2450 in a Model 2400 application

In this appendix:

Introduction

This section provides information for using the Model 2450 as a drop-in replacement in an existing Model 2400 application.

It also provides information on converting existing Model 2400 SCPI code to Model 2450 SCPI code.

Using a Model 2450 in an existing Model 2400 application

You can use a Model 2450 in an existing Model 2400 application. However, because of updates to the Model 2450, some Model 2400 options are no longer available or operate differently than they did for earlier Series 2400 products. Also, you will not have access to the extended ranges and other features that were introduced with the Model 2450.

This section details the differences between the 2400 SCPI command set that is available in the Model 2450 and the SCPI command set that was available in previous Series 2400 products.

If a command is not listed in this section, you can use the command in the same way that you did for the previous products.

Selecting the 2400 SCPI command set

The options discussed in the next few topics only apply if you have selected the 2400 SCPI command set. You can select the 2400 SCPI command set from the front panel or over the remote interface.

After you change to the 2400 SCPI command set, you must reboot the instrument. The front panel will prompt you to reboot, but remote commands will not.

Using the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Next to Command Set, select the button.
4. Select **2400 SCPI**.
5. Select **HOME** to return to the operating display.

Using SCPI or TSP remote commands:

Send the command:

```
*lang SCPI2400
```

Reboot the instrument.

Model 2400 to 2450 differences

You can use a Model 2450 in an existing Model 2400 application. The instruments are mostly compatible, with the following exceptions:

- There is no RS-232 communication port.
- The contact check function is not available.
- Some commands operate differently or are no longer available. These differences are described in the following sections.

Model 2400 commands that are supported but operate differently

The command `:DISPlay:CNDisplay` is supported in the 2400 SCPI command set, but will return the display to the home screen.

The commands `:OUTPut:ENABle[:STATe]` and `:OUTPut:ENABle:TRIPped?` are supported in the 2400 SCPI command set, but they affect the Model 2450 rear panel INTERLOCK connection instead of the DIGITAL I/O connection.

Commands that were added to the SCPI 2400 command set

To replace some features that are needed to use the Model 2450 in a Series 2400 application, the following commands were added.

- `:SYSTem:GPIB:ADDRess`: Assigns a GPIB address through a remote interface. See :SYSTem:GPIB:ADDRess (on page 6-104) for detail.
- `:SYSTem:TLINk`: Sets the digital I/O port to digital I/O or trigger link. See below for detail.

:SYSTem:TLINk information

The usage for this command is:

`:SYSTem:TLINk <n>`

Where `<n>` is:

- `0`: The DIGITAL I/O port on the rear panel of the instrument is set for digital I/O connections.
- `1`: The DIGITAL I/O port on the rear panel of the instrument is set for trigger link.

When the port is set for digital I/O, the following signals are available on the digital I/O connector:

- Pin 1: Out 1
- Pin 2: Out 2
- Pin 3: Out 3
- Pin 4: Out 4 (end of test (EOT) or BUSY)
- Pin 6: Input (start-of-test (SOT))

The commands that support digital I/O and Trigger Link can be used with these settings. For example, `SOURce2:TTL` can be used to set the I/O port bit pattern for the digital I/O state. `:ARM[:SEQuence[1]][LAYer[1]]:SOURce TLINk` can be selected when the state is `TLINk`.

To use Trigger Link connections, you must use the Model 2450-TLINK DB-9 to Trigger Link Connector Adapter.

If you have an application where you need to switch between the digital I/O and trigger link connections, you can use the `:SYSTem:TLINk` command and recall setups (`*SAV` and `*RCL`) to switch between the two states. After you save the setups, an example sequence is:

```
SYST:TLIN 0
*RCL 1
SYST:TLIN 1
*RCL 2
SYST:TLIN 0
*RCL 1
```

Model 2400 commands that are not supported in Model 2450

The Model 2450 introduced new features and hardware changes that made some earlier commands obsolete. These commands are documented in this section.

RS-232 commands

The Model 2450 no longer supports the RS-232 interface, so commands related to RS-232 operation are not supported even when you are using the 2400 SCPI command set

If you have existing code that sets RS-232 parameters, the commands will be accepted and ignored.

The Model 2400 commands that are related to the RS-232 interface that are no longer available are listed below.

- `:SYSTem:LOCal`
- `:SYSTem:RWLock`

Contact check commands

The Model 2450 no longer supports contact check, so commands related to this feature are not supported even when you are using the 2400 SCPI command set.

If you have existing code that sets contact check parameters, the commands will be accepted and ignored or are accepted and return a default value.

The Model 2400 commands related to this feature and the Model 2450 response to them are listed below.

Commands that are accepted and ignored

- `:CALCulate2:LIMit4:FAIL?`
- `:CALCulate2:LIMit4:SOURce2`
- `:CALCulate2:LIMit4:SOURce2?`
- `:CALCulate2:LIMit4:STATe`
- `:CALCulate2:LIMit4:STATe?`
- `:SYSTem:CCHeck`
- `:SYSTEM:CCHeck?`
- `:SYSTem:CCHEck:RESistance`
- `:SYSTem:CCHeck:RESistance?`
- `:TRIGger:SEQuence2:TOUT`
- `:TRIGger:SEQuence2:TOUT?`
- `:TRIGger:SEQuence2:SOURce`
- `:TRIGger:SEQuence2:SOURce?`

Commands that are accepted and return a default value

- `*OPT?` (0 is returned)

Display commands

The Model 2450 display is significantly different than the display of earlier models, so some commands related to the display no longer apply when you are using the 2400 SCPI command set.

If you have existing code that sets display parameters, the commands will either be accepted and return defaults, or be accepted and ignored.

The Model 2400 commands related to this displays that are no longer available are listed below.

Commands that are accepted and return a default value:

- `:DISPlay:ENABle`
- `:DISPlay:ENABle?`
- `:DISPlay:WINDow[1]:ATTRibutes?`
- `:DISPlay:WINDow[1]:DATA?`
- `:DISPlay:WINDow2:ATTRibutes?`
- `:DISPlay:WINDow2:DATA?`

Other commands

Some additional commands are no longer supported when you are using the 2400 SCPI command set.

The Model 2450 accepts the following command but ignores it:

- `:SYSTem:MEMory:INITialize`

The Model 2450 accepts the following command but returns a default of 0:

- `*TST?`

The Model 2450 accepts the following command but returns the last key that was remapped to the ENTER or EXIT:

- `:SYSTem:KEY`

Model 2400 SCPI to Model 2450 SCPI command cross-reference

This section provides information to help you convert existing Model 2400 SCPI code to Model 2450 SCPI code.

You must use the command set SCPI (not 2400 SCPI) to use the new commands. See Determining the command set you will use (on page 2-65) for information on setting the command set.

This section lists the SCPI commands that were available with the Model 2400, cross-referenced to the equivalent commands in the Model 2450 where available. Differences between the commands are noted. If no differences are noted, the command should operate the same on the Model 2450 as it did in the Model 2400.

CALCulate[1] subsystem

Model 2400 command	`:CALCulate[1]:DATA?`
Model 2450 command	Not supported
Notes	Use buffer to get user math data; see the TRACe subsystem (on page 6-107).

Model 2400 command	`:CALCulate[1]:DATA:LATest?`
Model 2450 command	Not supported
Notes	Use buffer to get user math data; see the TRACe subsystem (on page 6-107).

Model 2400 command	`:CALCulate[1]:MATH[:EXPression]:CATalog?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate[1]:MATH[:EXPression][:DEFine]`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate[1]:MATH[:EXPression]:DELete:ALL`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate[1]:MATH[:EXPression]:DELete[:SELected]`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate[1]:MATH[:EXPression]:NAME` `:CALCulate[1]:MATH[:EXPression]:NAME?`
Model 2450 command	POWER: [:SENSe[1]]:<function>:UNIT (on page 6-59), where `<function>` is `VOLTage` or `CURRent` and the setting is `WATT` `OFFCOMPOHM`, `VOLTCOEFF`, and `VARALPHA` are not available
Notes	

Model 2400 command	`:CALCulate[1]:STATe` `:CALCulate[1]:STATe?`
Model 2450 command	:CALCulate[1]:<function>:MATH:STATe (on page 6-12), where `<function>` is `VOLTage[:DC]`, `CURRent[:DC]`, or `RESistance` `:CALCulate[1]:<function>:MATH:STATe?`
Notes	If you send `:CALCulate[1]:STATe`, it will set the math state for all functions. If you send `:CALCulate[1]:STATe?`, a header error occurs.

Model 2400 command	`:CALCulate[1]:MATH:UNITs` `:CALCulate[1]:MATH:UNITs?`
Model 2450 command	Not available
Notes	

CALCulate2 subsystem

Model 2400 command	`:CALCulate2:CLIMits:BCONtrol` `:CALCulate2:CLIMits:BCONtrol?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:CLEar:AUTO` `:CALCulate2:CLIMits:CLEar:AUTO?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:CLEar[:IMMediate]`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:FAIL:SMLocation` `:CALCulate2:CLIMits:FAIL:SMLocation?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:FAIL:SOURce2` `:CALCulate2:CLIMits:FAIL:SOURce2?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:MODE` `:CALCulate2:CLIMits:MODE?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:PASS:SMLocation` `:CALCulate2:CLIMits:PASS:SMLocation?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:CLIMits:PASS:SOURce2` `:CALCulate2:CLIMits:PASS:SOURce2?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:DATA?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:DATA:LATest?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:CALCulate2:FEED` `:CALCulate2:FEED?`
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:LIMit[1]:COMPliance:FAIL :CALCulate2:LIMit[1]:COMPliance:FAIL?
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:LIMit[1]:COMPliance:SOURce2 :CALCulate2:LIMit[1]:COMPliance:SOURce2?
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:LIMit[1]:FAIL?
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:LIMit[1]:STATe :CALCulate2:LIMit[1]:STATe?
Model 2450 command	Not available
Notes	To disable source limits, set the limit value to the maximum allowed by the instrument

Model 2400 command	:CALCulate2:LIMit<x>:FAIL?
Model 2450 command	:CALCulate2:<function>:LIMit<y>:FAIL? (on page 6-15), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance
Notes	Note that :CALCulate2:LIMit4:FAIL? is not supported (related to contact check, which is not provided on Model 2450). Note that this only available for two limits in the Model 2450.

Model 2400 command	:CALCulate2:LIMit<x>:LOWer[:DATA] :CALCulate2:LIMit<x>:LOWer[:DATA]?
Model 2450 command	:CALCulate2:<function>:LIMit<y>:LOWer[:DATA] (on page 6-16), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance
Notes	Note that this only available for two limits in the Model 2450.

Model 2400 command	:CALCulate2:LIMit<x>:LOWer:SOURce2 :CALCulate2:LIMit<x>:LOWer:SOURce2?
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:LIMit<x>:PASS:SOURce2 :CALCulate2:LIMit<x>:PASS:SOURce2?
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:LIMit<x>:STATe :CALCulate2:LIMit<x>:STATe?
Model 2450 command	:CALCulate2:<function>:LIMit<y>:STATe (on page 6-17), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance and <y> is 1 or 2.
Notes	Note that this only available for two limits in the Model 2450. To disable source limits, set the limit value to the maximum allowed by the instrument.

Model 2400 command	:CALCulate2:LIMit<x>:UPPer[:DATA] :CALCulate2:LIMit<x>:UPPer[:DATA]?
Model 2450 command	:CALCulate2:<function>:LIMit<y>:UPPer[:DATA] (on page 6-18), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance and <y> is 1 or 2.
Notes	Note that this only available for two limits in the Model 2450.

Model 2400 command	:CALCulate2:LIMit<x>:UPPer:SOURce2 :CALCulate2:LIMit<x>:UPPer:SOURce2?
Model 2450 command	Not available
Notes	

Model 2400 command	:CALCulate2:NULL:ACQuire
Model 2450 command	[:SENSe[1]]:<function>:RELative:ACQuire (on page 6-56), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance
Notes	

Model 2400 command	:CALCulate2:NULL:OFFSet :CALCulate2:NULL:OFFSet?
Model 2450 command	[:SENSe[1]]:<function>:RELative (on page 6-55), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance [:SENSe[1]]:<function>:RELative?
Notes	

Model 2400 command	:CALCulate2:NULL:STATe :CALCulate2:NULL:STATe?
Model 2450 command	[:SENSe[1]]:<function>:RELative:STATe (on page 6-57), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance
Notes	

CALCulate3 subsystem

Model 2400 command	:CALCulate3:DATA?
Model 2450 command	Not available
Notes	Use reading buffers; see the TRACe subsystem (on page 6-107).

Model 2400 command	:CALCulate3:FORMat :CALCulate3:FORMat?
Model 2450 command	Not available
Notes	Use reading buffers; see the TRACe subsystem (on page 6-107).

CONFigure

Model 2400 command	:CONFigure :CONFigure?
Model 2450 command	Not available
Notes	

DISPlay subsystem

Model 2400 command	:DISPlay:CNDisplay
Model 2450 command	:DISPlay:SCReen (on page 6-26)
Notes	
Model 2400 command	:DISPlay:DIGits :DISPlay:DIGits?
Model 2450 command	:DISPlay:<function>:DIGits (on page 6-23), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance :DISPlay:<function>:DIGits?
Notes	Settings are 3 to 6 instead of 4 to 7
Model 2400 command	:DISPlay:ENABle :DISPlay:ENABle?
Model 2450 command	:DISPlay:LIGHt:STATe (on page 6-24) :DISPlay:LIGHt:STATe?
Notes	
Model 2400 command	:DISPlay[:WINDow[1]]:ATTRibutes?
Model 2450 command	Not available
Notes	
Model 2400 command	:DISPlay[:WINDow[1]]:DATA?
Model 2450 command	Not available
Notes	
Model 2400 command	:DISPlay:WINDow<n>:TEXT:DATA :DISPlay:WINDow<n>:TEXT:DATA?
Model 2450 command	:DISPlay:USER<n>:TEXT[:DATA] (on page 6-27) :DISPlay:USER<n>:TEXT[:DATA]?
Notes	
Model 2400 command	:DISPlay:WINDow<n>:TEXT:STATe :DISPlay:WINDow<n>:TEXT:STATe?
Model 2450 command	:DISPlay:USER<n>:TEXT[:DATA] (on page 6-27) :DISPlay:USER<n>:TEXT[:DATA]?
Notes	
Model 2400 command	:DISPlay:WINDow2:ATTRibutes?
Model 2450 command	Not available
Notes	
Model 2400 command	:DISPlay:WINDow2:DATA?
Model 2450 command	Not available
Notes	

FETCh?

Model 2400 command	`:FETCh?`
Model 2450 command	:FETCh? (on page 6-3)
Notes	Can choose different buffers and which buffer elements to access from the buffer for Model 2450.

FORMat subsystem

Model 2400 command	`:FORMat:BORDer` `:FORMat:BORDer?`
Model 2450 command	:FORMat:BORDer (on page 6-29) `:FORMat:BORDer?`
Notes	

Model 2400 command	`:FORMat[:DATA]` `:FORMat[:DATA]?`
Model 2450 command	:FORMat[:DATA] (on page 6-30) `:FORMat[:DATA]?`
Notes	

Model 2400 command	`:FORMat:ELEMents[:SENSe[1]]` `:FORMat:ELEMents[:SENSe[1]]?`
Model 2450 command	Not available
Notes	In the Model 2450, format elements are specified with the `READ?`, `FETCh?`, `MEASure?`, and `TRACe:DATA?` commands with each use instead of using a global setting for all commands. The elements may be unique for each command and are unique each time the command is processed.

Model 2400 command	`:FORMat:ELEMents:CALCulate` `:FORMat:ELEMents:CALCulate?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:FORMat:SOURce2` `:FORMat:SOURce2?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:FORMat:SREGister` `:FORMat:SREGister?`
Model 2450 command	Not available
Notes	

OUTPut subsystem

Model 2400 command	`:OUTPut[1]:ENABle[:STATe]` `:OUTPut[1]:ENABle[:STATe]?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:OUTPut[1]:ENABle:TRIPped?`
Model 2450 command	Not available
Notes	

Model 2400 command	`OUTPut[1]:INTerlock:TRIPped?`
Model 2450 command	:OUTPut[1]:INTerlock:TRIPped? (on page 6-33)
Notes	

Model 2400 command	`:OUTPut[1]:SMODe` `:OUTPut[1]:SMODe?`
Model 2450 command	:OUTPut[1]:<function>:SMODe (on page 6-31), where `<function>` is `VOLTage[:DC]` or `CURRent[:DC]` `OUTPut[1]:<function>:SMODe?`
Notes	

Model 2400 command	`:OUTPut[1][:STATe]` `:OUTPut[1][:STATe]?`
Model 2450 command	:OUTPut[1][:STATe] (on page 6-34) `:OUTPut[1][:STATe]?`
Notes	

READ?

Model 2400 command	`:READ?`
Model 2450 command	:READ? (on page 6-6)
Notes	Model 2450 allows you to choose different buffers and which buffer elements to access from the buffers when you send the command.

ROUTe subsystem

Model 2400 command	`:ROUTe:TERMinals` `:ROUTe:TERMinals?`
Model 2450 command	:ROUTe:TERMinals (on page 6-35) `:ROUTe:TERMinals?`
Notes	

SENSe subsystem

Model 2400 command	`:MEASure:CURRent[:DC]?`
	`:MEASure:RESistance?`
	`:MEASure:VOLTage[:DC]?`
	`:MEASure[:<function>]?`

Model 2450 command	:MEASure:<function>? (on page 6-4), where `<function>` is `VOLTage[:DC]`, `CURRent[:DC]`, or `RESistance`.
Notes	Can specify a buffer in Model 2450 and which buffer elements to access from the buffer when you specify the command. In the Model 2450, this command changes the measurement function to the function in the command if it is not already active, makes readings, and stores them in a reading buffer, which you can specify. When it changes to that function, it recalls the settings as they were the last time that function was active. It does not go to factory default settings for the function (which the Series 2400 did). Also, in the Model 2450, this command does not map to `CONFigure - READ - FETCh`.

Model 2400 command	[:SENSe[1]]:AVERage:COUNt [:SENSe[1]]:AVERage:COUNt?
Model 2450 command	[:SENSe[1]]:<function>:AVERage:COUNt (on page 6-43), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance.
Notes	This is now set for each measurement function.

Model 2400 command	[:SENSe[1]]:AVERage[:STATe] [:SENSe[1]]:AVERage[:STATe]?
Model 2450 command	[:SENSe[1]]:<function>:AVERage[:STATe] (on page 6-44), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance.
Notes	This is now set for each measurement function.

Model 2400 command	[:SENSe[1]]:AVERage:TCONtrol [:SENSe[1]]:AVERage:TCONtrol?
Model 2450 command	[:SENSe[1]]:<function>:AVERage:TCONtrol (on page 6-45), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance.
Notes	This is now set for each measurement function.

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:NPLCycles [:SENSe[1]]:CURRent[:DC]:NPLCycles? [:SENSe[1]]:RESistance:NPLCycles [:SENSe[1]]:RESistance:NPLCycles? [:SENSe[1]]:VOLTage[:DC]:NPLCycles [:SENSe[1]]:VOLTage[:DC]:NPLCycles?
Model 2450 command	[:SENSe[1]]:<function>:NPLCycles (on page 6-48), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance. [:SENSe[1]]:<function>:NPLCycles?
Notes	If you send [:SENSe[1]]:NPLCycles, it sets NPLCs for all functions.

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:PROTection[:LEVel] [:SENSe[1]]:CURRent[:DC]:PROTection[:LEVel]?
Model 2450 command	:SOURce[1]:<function>:<x>LIMit[:LEVel] (on page 6-71)
Notes	

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:PROTection:RSYNchronize [:SENSe[1]]:CURRent[:DC]:PROTection:RSYNchronize? [:SENSe[1]]:VOLTage[:DC]:PROTection:RSYNchronize [:SENSe[1]]:VOLTage[:DC]:PROTection:RSYNchronize?
Model 2450 command	Not available
Notes	Range synchronization is always turned on in Model 2450

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:PROTection:TRIPped?
Model 2450 command	:SOURce[1]:<function>:<x>LIMit[:LEVel]:TRIPped? (on page 6-72)
Notes	

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:RANGe:AUTO [:SENSe[1]]:CURRent[:DC]:RANGe:AUTO? [:SENSe[1]]:RESistance:RANGe:AUTO [:SENSe[1]]:RESistance:RANGe:AUTO? [:SENSe[1]]:VOLTage[:DC]:RANGe:AUTO [:SENSe[1]]:VOLTage[:DC]:RANGe:AUTO?
Model 2450 command	[:SENSe[1]]:<function>:RANGe:AUTO (on page 6-50), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance. [:SENSe[1]]:<function>:RANGe:AUTO?
Notes	

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:RANGe:AUTO:LLIMit [:SENSe[1]]:CURRent[:DC]:RANGe:AUTO:LLIMit? [:SENSe[1]]:RESistance:RANGe:AUTO:LLIMit [:SENSe[1]]:RESistance:RANGe:AUTO:LLIMit? [:SENSe[1]]:VOLTage[:DC]:RANGe:AUTO:LLIMit [:SENSe[1]]:VOLTage[:DC]:RANGe:AUTO:LLIMit?
Model 2450 command	[:SENSe[1]]:<function>:RANGe:AUTO:LLIMit (on page 6-51), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance. [:SENSe[1]]:<function>:RANGe:AUTO:LLIMit?
Notes	

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:RANGe:AUTO:ULIMit? [:SENSe[1]]:RESistance:RANGe:AUTO:ULIMit [:SENSe[1]]:RESistance:RANGe:AUTO:ULIMit? [:SENSe[1]]:VOLTage[:DC]:RANGe:AUTO:ULIMit?
Model 2450 command	[:SENSe[1]]:<function>:RANGe:AUTO:ULIMit (on page 6-52), where <function> is VOLTage[:DC] or RESistance.
Notes	Upper limit is not available for current for Model 2450. For voltage, you can query the upper limit for voltage, but not set it.

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:RANGe:HOLDoff [:SENSe[1]]:CURRent[:DC]:RANGe:HOLDoff?
Model 2450 command	Not available
Notes	

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:RANGe:HOLDoff:DELay [:SENSe[1]]:CURRent[:DC]:RANGe:HOLDoff:DELay?
Model 2450 command	Not available
Notes	

Model 2400 command	[:SENSe[1]]:CURRent[:DC]:RANGe[:UPPer] [:SENSe[1]]:CURRent[:DC]:RANGe[:UPPer]? [:SENSe[1]]:RESistance:RANGe[:UPPer] [:SENSe[1]]:RESistance:RANGe[:UPPer]? [:SENSe[1]]:VOLTage[:DC]:RANGe[:UPPer] [:SENSe[1]]:VOLTage[:DC]:RANGe[:UPPer]?
Model 2450 command	[:SENSe[1]]:<function>:RANGe[:UPPer] (on page 6-53), where <function> is VOLTage[:DC], CURRent[:DC], or RESistance. [:SENSe[1]]:<function>:RANGe[:UPPer]?
Notes	

Model 2400 command	`[:SENSe[1]]:DATA[:LATest]?`
Model 2450 command	Not available
Notes	Retrieve buffers instead; see TRACe subsystem (on page 6-107).

Model 2400 command	`[:SENSe[1]]:FUNCtion:CONCurrent` `[:SENSe[1]]:FUNCtion:CONCurrent?`
Model 2450 command	Not available
Notes	To do a similar action, set :SOURce[1]:<function>:READ:BACK (on page 6-77) and store both a measure and source values in the reading buffer.

Model 2400 command	`[:SENSe[1]]:FUNCtion:OFF` `[:SENSe[1]]:FUNCtion:OFF?`
Model 2450 command	Not available
Notes	Only one measurement function is active at a time.

Model 2400 command	`[:SENSe[1]]:FUNCtion:OFF:ALL`
Model 2450 command	Not available
Notes	Only one measurement function is active at a time.

Model 2400 command	`[:SENSe[1]]:FUNCtion:OFF:COUNt?`
Model 2450 command	Not available
Notes	Only one measurement function is active at a time.

Model 2400 command	`[:SENSe[1]]:FUNCtion[:ON]` `[:SENSe[1]]:FUNCtion[:ON]?`
Model 2450 command	[:SENSe[1]]:FUNCtion[:ON] (on page 6-49) `[:SENSe[1]]:FUNCtion[:ON]?`
Notes	Does not support a list parameter in the Model 2450

Model 2400 command	`[:SENSe[1]]:FUNCtion[:ON]:ALL`
Model 2450 command	Not available
Notes	Only one measurement function is active at a time.

Model 2400 command	`[:SENSe[1]]:FUNCtion[:ON]:COUNt?`
Model 2450 command	Not available
Notes	Only one measurement function is active at a time.

Model 2400 command	`[:SENSe[1]]:FUNCtion:STATe?`
Model 2450 command	`[:SENSe[1]]:FUNCtion[:ON]?`
Notes	Only one measurement function is active at a time. `[:SENSe[1]]:FUNCtion[:ON]?` queries the active measurement function.

Model 2400 command	`[:SENSe[1]]:RESistance:HOLDoff` `[:SENSe[1]]:RESistance:HOLDoff?`
Model 2450 command	Not available
Notes	

Model 2400 command	[:SENSe[1]]:RESistance:HOLDoff:DELay [:SENSe[1]]:RESistance:HOLDoff:DELay?
Model 2450 command	Not available
Notes	

Model 2400 command	[:SENSe[1]]:RESistance:MODE [:SENSe[1]]:RESistance:MODE?
Model 2450 command	[:SENSe[1]]:<function>:MODE, where <function> is RESistance. [:SENSe[1]]:<function>:MODE?
Notes	

Model 2400 command	[:SENSe[1]]:RESistance:OCOMpensated [:SENSe[1]]:RESistance:OCOMpensated?
Model 2450 command	[:SENSe[1]]:<function>:OCOMpensated (on page 6-49), where <function> is RESistance. [:SENSe[1]]:<function>:OCOMpensated?
Notes	

Model 2400 command	[:SENSe[1]]:VOLTage[:DC]:PROTection[:LEVel] [:SENSe[1]]:VOLTage[:DC]:PROTection[:LEVel]?
Model 2450 command	:SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73), where <function> is VOLTage[:DC]. :SOURce[1]:VOLTage:ILIMit[:LEVel]?
Notes	

Model 2400 command	[:SENSe[1]]:VOLTage[:DC]:PROTection:TRIPped?
Model 2450 command	:SOURce[1]:<function>:PROTection[:LEVel]:TRIPped? (on page 6-74), where <function> is VOLTage[:DC].
Notes	

SOURce[1] subsystem

Model 2400 command	:SOURce[1]:CLEar:AUTO :SOURce[1]:CLEar:AUTO?
Model 2450 command	Not available
Notes	Use the Trigger model (on page 3-95).

Model 2400 command	:SOURce[1]:CLEar:AUTO:MODE :SOURce[1]:CLEar:AUTO:MODE?
Model 2450 command	Not available
Notes	Use the Trigger model (on page 3-95).

Model 2400 command	:SOURce[1]:CLEar[:IMMediate]
Model 2450 command	Not available
Notes	Use the Trigger model (on page 3-95).

Model 2400 command	:SOURce[1]:CURRent:CENTer :SOURce[1]:CURRent:CENTer? :SOURce[1]:VOLTage:CENTer :SOURce[1]:VOLTage:CENTer?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:CURRent[:LEVel][:IMMediate][:AMPLitude] :SOURce[1]:CURRent[:LEVel][:IMMediate][:AMPLitude]? :SOURce[1]:VOLTage[:LEVel][:IMMediate][:AMPLitude] :SOURce[1]:VOLTage[:LEVel][:IMMediate][:AMPLitude]?
Model 2450 command	:SOURce[1]:<function>[:LEVel][:IMMediate][:AMPLitude] (on page 6-70), where <function> is CURRent or VOLTage. :SOURce[1]:VOLTage[:LEVel][:IMMediate][:AMPLitude]?
Notes	

Model 2400 command	:SOURce[1]:CURRent[:LEVel]:TRIGgered[:AMPLitude] :SOURce[1]:CURRent[:LEVel]:TRIGgered[:AMPLitude]? :SOURce[1]:VOLTage[:LEVel]:TRIGgered[:AMPLitude] :SOURce[1]:VOLTage[:LEVel]:TRIGgered[:AMPLitude]?
Model 2450 command	Not available
Notes	Use the Trigger model (on page 3-95).

Model 2400 command	:SOURce[1]:CURRent[:LEVel]:TRIGgered:SFACtor :SOURce[1]:CURRent[:LEVel]:TRIGgered:SFACtor? :SOURce[1]:VOLTage[:LEVel]:TRIGgered:SFACtor :SOURce[1]:VOLTage[:LEVel]:TRIGgered:SFACtor?
Model 2450 command	Not available
Notes	Use the Trigger model (on page 3-95).

Model 2400 command	:SOURce[1]:CURRent[:LEVel]:TRIGgered:SFACtor:STATe :SOURce[1]:CURRent[:LEVel]:TRIGgered:SFACtor:STATe? :SOURce[1]:VOLTage[:LEVel]:TRIGgered:SFACtor:STATe :SOURce[1]:VOLTage[:LEVel]:TRIGgered:SFACtor:STATe?
Model 2450 command	Not available
Notes	Use the Trigger model (on page 3-95).

Model 2400 command	:SOURce[1]:CURRent:MODE :SOURce[1]:CURRent:MODE? :SOURce[1]:VOLTage:MODE :SOURce[1]:VOLTage:MODE?
Model 2450 command	:SOURce[1]:FUNCtion[:MODE] (on page 6-72) :SOURce[1]:FUNCtion[:MODE]?
Notes	

Model 2400 command	:SOURce[1]:CURRent:RANGe :SOURce[1]:CURRent:RANGe? :SOURce[1]:VOLTage:RANGe :SOURce[1]:VOLTage:RANGe?
Model 2450 command	:SOURce[1]:<function>:RANGe (on page 6-74), where <function> is CURRent or VOLTage. :SOURce[1]:<function>:RANGe?
Notes	

Model 2400 command	:SOURce[1]:CURRent:RANGe:AUTO :SOURce[1]:CURRent:RANGe:AUTO? :SOURce[1]:VOLTage:RANGe:AUTO :SOURce[1]:VOLTage:RANGe:AUTO?
Model 2450 command	:SOURce[1]:<function>:RANGe:AUTO (on page 6-75), where <function> is CURRent or VOLTage. :SOURce[1]:<function>:RANGe:AUTO?
Notes	

Model 2400 command	:SOURce[1]:CURRent:SPAN :SOURce[1]:CURRent:SPAN? :SOURce[1]:VOLTage:SPAN :SOURce[1]:VOLTage:SPAN?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:CURRent:STARt :SOURce[1]:CURRent:STARt? :SOURce[1]:VOLTage:STARt :SOURce[1]:VOLTage:STARt?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:CURRent:STEP :SOURce[1]:CURRent:STEP? :SOURce[1]:VOLTage:STEP :SOURce[1]:VOLTage:STEP?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:CURRent:STOP :SOURce[1]:CURRent:STOP? :SOURce[1]:VOLTage:STOP :SOURce[1]:VOLTage:STOP?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:DELay :SOURce[1]:DELay?
Model 2450 command	:SOURce[1]:<function>:DELay (on page 6-66), where <function> is CURRent or VOLTage. :SOURce[1]:<function>:DELay?
Notes	When a delay is set, source auto delay is turned off.

Model 2400 command	:SOURce[1]:DELay:AUTO :SOURce[1]:DELay:AUTO?
Model 2450 command	:SOURce[1]:<function>:DELay:AUTO (on page 6-67) :SOURce[1]:DELay:AUTO?
Notes	

Model 2400 command	:SOURce[1]:FUNCtion[:MODE] :SOURce[1]:FUNCtion[:MODE]?
Model 2450 command	:SOURce[1]:FUNCtion[:MODE] (on page 6-72) :SOURce[1]:FUNCtion[:MODE]?
Notes	

Model 2400 command	:SOURce[1]:LIST:CURRent :SOURce[1]:LIST:CURRent?
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this sets up a list of customer values for a sweep.

Model 2400 command	:SOURce[1]:LIST:CURRent:APPend
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this setting is set as part of the configuration list that is created by the sweep command.

Model 2400 command	:SOURce[1]:LIST:CURRent:POINts?
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this setting is set as part of the configuration list that is created by the sweep command.

Model 2400 command	:SOURce[1]:LIST:CURRent:STARt :SOURce[1]:LIST:CURRent:STARt?
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this setting is set as part of the configuration list that is created by the sweep command.

Model 2400 command	`:SOURce[1]:LIST:VOLTage` `:SOURce[1]:LIST:VOLTage?`
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this sets up a list of customer values for a sweep.

Model 2400 command	`:SOURce[1]:LIST:VOLTage:APPend`
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this setting is set as part of the configuration list that is created by the sweep command.

Model 2400 command	`:SOURce[1]:LIST:VOLTage:POINts?`
Model 2450 command	Not available
Notes	In the Model 2450, this setting is set as part of the configuration list that is created by the sweep command.

Model 2400 command	`:SOURce[1]:LIST:VOLTage:STARt` `:SOURce[1]:LIST:VOLTage:STARt?`
Model 2450 command	:SOURce[1]:SWEep:<function>:LIST (on page 6-85)
Notes	In the Model 2450, this setting is set as part of the configuration list that is created by the sweep command.

Model 2400 command	`:SOURCe[1]:MEMory:POINts` `:SOURCe[1]:MEMory:POINts?`
Model 2450 command	Not available
Notes	You can achieve functionality that is close to source memory with the Model 2450 configuration lists. See Configuration lists (on page 3-33) for information.

Model 2400 command	`:SOURCe[1]:MEMory:RECall`
Model 2450 command	Not available
Notes	You can achieve functionality that is close to source memory with the Model 2450 configuration lists. See Configuration lists (on page 3-33) for information.

Model 2400 command	`:SOURce[1]:MEMory:SAVE`
Model 2450 command	Not available
Notes	You can achieve functionality that is close to source memory with the Model 2450 configuration lists. See Configuration lists (on page 3-33) for information.

Model 2400 command	`:SOURce[1]:MEMory:STARt` `:SOURce[1]:MEMory:STARt?`
Model 2450 command	Not available
Notes	You can achieve functionality that is close to source memory with the Model 2450 configuration lists. See Configuration lists (on page 3-33) for information.

Model 2400 command	`:SOURce[1]:SOAK` `:SOURce[1]:SOAK?`
Model 2450 command	Not available
Notes	

Model 2400 command	:SOURce[1]:SWEep:CABort :SOURce[1]:SWEep:CABort?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:SWEep:DIRection :SOURce[1]:SWEep:DIRection?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:SWEep:POINts :SOURce[1]:SWEep:POINts?
Model 2450 command	Use the sweep commands to define sweeps: :SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:SWEep:RANGing :SOURce[1]:SWEep:RANGing?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:SWEep:SPACing :SOURce[1]:SWEep:SPACing?
Model 2450 command	:SOURce[1]:SWEep:<function>:LINear (on page 6-81) :SOURce[1]:SWEep:<function>:LINear:STEP (on page 6-83) :SOURce[1]:SWEep:<function>:LIST (on page 6-85) :SOURce[1]:SWEep:<function>:LOG (on page 6-87)
Notes	Sweep parameters are built into the sweep command path. See Sweep operation (on page 3-51) for information.

Model 2400 command	:SOURce[1]:VOLTage:PROTection[:LEVel] :SOURce[1]:VOLTage:PROTection[:LIMit]?
Model 2450 command	:SOURce[1]:<function>:PROTection[:LEVel] (on page 6-73)
Notes	

Model 2400 command	:SOURce[1]:VOLTage:PROTection[:LEVEL]:TRIPped?
Model 2450 command	:SOURce[1]:<function>:PROTection[:LEVEL]:TRIPped? (on page 6-74)
Notes	

SOURce2 subsystem

Model 2400 command	`:SOURce2:BSIZe` `:SOURce2:BSIZe?`
Model 2450 command	Not available
Notes	All digital inputs and outputs on the Model 2450 are general; you can choose as appropriate. See Digital I/O (on page 3-62) for information.

Model 2400 command	`:SOURce2:CLEar:AUTO` `:SOURce2:CLEar:AUTO?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:SOURce2:CLEar:AUTO:DELay` `:SOURce2:CLEar:AUTO:DELay?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:SOURce2:CLEar[:IMMediate]`
Model 2450 command	Not available
Notes	

Model 2400 command	`:SOURce2:TTL[:LEVel][:DEFault]` `:SOURce2:TTL[:LEVel][:DEFault]?`
Model 2450 command	Not available
Notes	All inputs and outputs on the Model 2450 are general; you can choose as appropriate. See Digital I/O (on page 3-62) for information.

Model 2400 command	`:SOURce2:TTL[:LEVel]:ACTual?`
Model 2450 command	Not available
Notes	All inputs and outputs on the Model 2450 are general; you can choose as appropriate. See Digital I/O (on page 3-62) for information.

Model 2400 command	`:SOURce2:TTL4:BSTate` `:SOURce2:TTL4:BSTate?`
Model 2450 command	Not available
Notes	All inputs and outputs on the Model 2450 are general; you can choose as appropriate. See Digital I/O (on page 3-62) for information.

Model 2400 command	`:SOURce2:TTL4:MODE` `:SOURce2:TTL4:MODE?`
Model 2450 command	Not available
Notes	All inputs and outputs on the Model 2450 are general; you can choose as appropriate. See Digital I/O (on page 3-62) for information.

STATus subsystem

Model 2400 command	`:STATus:MEASurement:CONDition?`
Model 2450 command	Not available
Notes	Use the questionable register to emulate this register; see the STATus subsystem (on page 6-89) for information.

Model 2400 command	`:STATus:MEASurement:ENABle` `:STATus:MEASurement:ENABle?`
Model 2450 command	Not available
Notes	Use the questionable register to emulate this register; see the STATus subsystem (on page 6-89) for information.

Model 2400 command	`:STATus:MEASurement[:EVENt]?`
Model 2450 command	Not available
Notes	Use the questionable register to emulate this register; see the STATus subsystem (on page 6-89) for information.

Model 2400 command	`:STATus:OPERation:CONDition?`
Model 2450 command	:STATus:OPERation:CONDition? (on page 6-89)
Notes	In the Model 2450, you need to map events into the register (there are no set bits). See Status model (on page C-1) for information.

Model 2400 command	`:STATus:OPERation:ENABle` `:STATus:OPERation:ENABle?`
Model 2450 command	:STATus:OPERation:ENABle (on page 6-89) `:STATus:OPERation:ENABle?`
Notes	In the Model 2450, you need to map events into the register (there are no set bits). See Status model (on page C-1) for information.

Model 2400 command	`:STATus:OPERation[:EVENt]?`
Model 2450 command	:STATus:OPERation[:EVENt]? (on page 6-90)
Notes	In the Model 2450, you need to map events into the register (there are no set bits). See Status model (on page C-1) for information.

Model 2400 command	`:STATus:PRESet`
Model 2450 command	:STATus:PRESet (on page 6-92)
Notes	

Model 2400 command	`:STATus:QUEStionable:CONDition?`
Model 2450 command	:STATus:QUEStionable:CONDition? (on page 6-92)
Notes	In the Model 2450, you need to map events into the register (there are no set bits). See Status model (on page C-1) for information.

Model 2400 command	`:STATus:QUEStionable:ENABle` `:STATus:QUEStionable:ENABle?`
Model 2450 command	:STATus:QUEStionable:ENABle (on page 6-93) `:STATus:QUEStionable:ENABle?`
Notes	In the Model 2450, you need to map events into the register (there are no set bits). See Status model (on page C-1) for information.

Model 2400 command	`:STATus:QUEStionable[:EVENt]?`
Model 2450 command	`:STATus:QUEStionable[:EVENt]? (on page 6-93)`
Notes	In the Model 2450, you need to map events into the register (there are no set bits). See Status model (on page C-1) for information.

Model 2400 command	`:STATus:QUEue:CLEar`
Model 2450 command	Not available
Notes	Use the event log; see Using the event log (on page 2-117).

Model 2400 command	`:STATus:QUEue:DISable` `:STATus:QUEue:DISable?`
Model 2450 command	Not available
Notes	Use the event log; see Using the event log (on page 2-117).

Model 2400 command	`:STATus:QUEue:ENABle` `:STATus:QUEue:ENABle?`
Model 2450 command	Not available
Notes	Use the event log; see Using the event log (on page 2-117).

Model 2400 command	`:STATus:QUEue[:NEXT]?`
Model 2450 command	Not available
Notes	Use the event log; see Using the event log (on page 2-117).

SYStem subsystem

Model 2400 command	`:SYSTem:AZERo:CACHing:NPLCycles?`
Model 2450 command	Not available
Notes	Caching is always on in Model 2450.

Model 2400 command	`:SYSTem:AZERo:CACHing:REFResh`
Model 2450 command	Not available
Notes	Caching is always on in Model 2450.

Model 2400 command	`:SYSTem:AZERo:CACHing:RESet`
Model 2450 command	Not available
Notes	Caching is always on in Model 2450.

Model 2400 command	`:SYSTem:AZERo:CACHing[:STATe]` `:SYSTem:AZERo:CACHing[:STATe]?`
Model 2450 command	Not available
Notes	Caching is always on in Model 2450.

Model 2400 command	`:SYSTem:AZERo:STATe` `:SYSTem:AZERo:STATe?`
Model 2450 command	[:SENSe[1]]:<function>:AZERo[:STATe] (on page 6-46), where `<function>` is `CURRent[:DC]`, `RESistance`, or `VOLTage[:DC]`. `[:SENSe[1]]:<function>:AZERo[:STATe]?`
Notes	

Model 2400 command	`:SYSTem:BEEPer[:IMMediate]`
Model 2450 command	:SYSTem:BEEPer[:IMMediate] (on page 6-96)
Notes	

Model 2400 command	`:SYSTem:BEEPer:STATe` `:SYSTem:BEEPer:STATe?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:SYSTem:CCHeck` `:SYSTem:CCHeck?`
Model 2450 command	Not available
Notes	No contact check on Model 2450.
Model 2400 command	`:SYSTem:CCHeck:RESistance` `:SYSTem:CCHeck:RESistance?`
Model 2450 command	Not available
Notes	No contact check on Model 2450.
Model 2400 command	`:SYSTem:CLEar`
Model 2450 command	:SYSTem:CLEar (on page 6-96)
Notes	
Model 2400 command	`:SYSTem:ERRor:ALL?`
Model 2450 command	Not available
Notes	See Using the event log (on page 2-117) for changes to error reporting.
Model 2400 command	`:SYSTem:ERRor:CODE:ALL?`
Model 2450 command	Not available
Notes	See Using the event log (on page 2-117) for changes to error reporting.
Model 2400 command	`:SYSTem:ERRor:CODE[:NEXT]?`
Model 2450 command	:SYSTem:ERRor:CODE[:NEXT]? (on page 6-99)
Notes	Returns only errors from the event log. See Using the event log (on page 2-117) for changes to error reporting.
Model 2400 command	`:SYSTem:ERRor:COUNt?`
Model 2450 command	:SYSTem:ERRor:COUNt? (on page 6-99)
Notes	Returns only errors from the event log. See Using the event log (on page 2-117) for changes to error reporting.
Model 2400 command	`:SYSTem:ERRor[:NEXT]?`
Model 2450 command	:SYSTem:ERRor[:NEXT]? (on page 6-98)
Notes	Returns only errors from the event log. See Using the event log (on page 2-117) for changes to error reporting.
Model 2400 command	`:SYSTem:GUARd` `:SYSTem:GUARd?`
Model 2450 command	Not available
Notes	Cable guard is the only option available on Model 2450.
Model 2400 command	`:SYSTem:KEY` `:SYSTem:KEY?`
Model 2450 command	Not available
Notes	

Model 2400 command	:SYSTem:LFRequency :SYSTem:LFRequency?
Model 2450 command	:SYSTem:LFRequency? (on page 6-104)
Notes	Line frequency is always automatically detected in the Model 2450.

Model 2400 command	:SYSTem:LFRequency:AUTO :SYSTem:LFRequency:AUTO?
Model 2450 command	Not available
Notes	Line frequency is always automatically detected in the Model 2450.

Model 2400 command	:SYSTem:LOCal
Model 2450 command	Not available
Notes	No RS-232 communications available in the Model 2450.

Model 2400 command	:SYSTem:MEMory:INITialize
Model 2450 command	Not available
Notes	No battery-backed RAM in the Model 2450.

Model 2400 command	:SYSTem:MEP:HOLDoff
Model 2450 command	
Notes	

Model 2400 command	:SYSTem:MEP[:STATe]?
Model 2450 command	
Notes	

Model 2400 command	:SYSTem:POSetup :SYSTem:POSetup?
Model 2450 command	:SYSTem:POSetup (on page 6-105) :SYSTem:POSetup?
Notes	

Model 2400 command	:SYSTem:PRESet
Model 2450 command	Not available
Notes	

Model 2400 command	:SYSTem:RCMode :SYSTem:RCMode?
Model 2450 command	Not available
Notes	

Model 2400 command	:SYSTem:RSENse :SYSTem:RSENse?
Model 2450 command	[:SENSe[1]]:<function>:RSENse (on page 6-58), where <function> is VOLTage[:DC] or RESistance. [:SENSe[1]]:<function>:RSENse?
Notes	

Model 2400 command	`:SYSTem:RWLock` `:SYSTem:RWLock?`
Model 2450 command	Not available
Notes	No RS-232 communications available in the Model 2450.

Model 2400 command	`:SYSTem:TIME?`
Model 2450 command	:SYSTem:TIME (on page 6-106) `:SYSTem:TIME?`
Notes	Model 2450 uses absolute time.

Model 2400 command	`:SYSTem:TIME:RESet`
Model 2450 command	Not available
Notes	Model 2450 uses absolute time.

Model 2400 command	`:SYSTem:TIME:RESet:AUTO`
Model 2450 command	`Not available`
Notes	Model 2450 uses absolute time.

Model 2400 command	`:SYSTem:VERSion?`
Model 2450 command	:SYSTem:VERSion? (on page 6-107)
Notes	

TRACe subsystem

Model 2400 command	`:TRACe:CLEar`
Model 2450 command	:TRACe:CLEar (on page 6-108)
Notes	Allows selection of the buffer to clear.

Model 2400 command	`:TRACe:DATA?`
Model 2450 command	:TRACe:DATA? (on page 6-109)
Notes	In the Model 2450, this command allows you to dynamically specify the buffer elements to retrieve from the reading buffer.

Model 2400 command	`:TRACe:FEED` `:TRACe:FEED?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:TRACe:FEED:CONTrol` `:TRACe:FEED:CONTrol?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:TRACe:FREE?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:TRACe:POINts` `:TRACe:POINts?`
Model 2450 command	:TRACe:POINts (on page 6-115) `:TRACe:POINts?`
Notes	The command allows you to resize the buffer. The query returns the maximum capacity of the buffer.

Model 2400 command	`:TRACe:POINts:ACTual?`
Model 2450 command	:TRACe:ACTual? (on page 6-107)
Notes	In the Model 2450, you can specify the buffer.

Model 2400 command	`:TRACe:TSTamp:FORMat` `:TRACe:TSTamp:FORMat?`
Model 2450 command	Not available
Notes	In the Model 2450, you can specify a timestamp element using :TRACe:DATA? (on page 6-109).

TRIGger subsystem

Model 2400 command	`ABORt`
Model 2450 command	:ABORt (on page 6-2)
Notes	

Model 2400 command	`:ARM[:SEQuence[1]][:LAYer[1]]:COUNt` `:ARM[:SEQuence[1]][:LAYer[1]]:COUNt?`
Model 2450 command	Not available
Notes	Use :TRIGger:BLOCk:BRANch:COUNter (on page 6-128).

Model 2400 command	`:ARM[:SEQuence[1]][:LAYer[1]]:SOURce` `:ARM[:SEQuence[1]][:LAYer[1]]:SOURce?`
Model 2450 command	Not available
Notes	Similar functionality available with :TRIGger:BLOCk:WAIT (on page 6-148).

Model 2400 command	`:ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:DIRection` `:ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:DIRection?`
Model 2450 command	Not available
Notes	For similar functionality, see the following commands: • :TRIGger:BLOCk:BRANch:ALWays (on page 6-128) • :TRIGger:BLOCk:BRANch:ONCE (on page 6-134) • :TRIGger:BLOCk:BRANch:ONCE:EXCLuded (on page 6-135)

Model 2400 command	`:ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:ILINe` `:ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:ILINe?`
Model 2450 command	Not available
Notes	For similar functionality, use a digital I/O line with the trigger model. Generate a notify event in the trigger model that feeds the stimulus setting to a digital I/O line to pulse as needed in the trigger model. See: • :TRIGger:BLOCk:NOTify (on page 6-146) • :TRIGger:DIGital<n>:OUT:STIMulus (on page 6-153)

Model 2400 command	:ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:OLINe :ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:OLINe?
Model 2450 command	Not available
Notes	For similar functionality, use a digital I/O line with the trigger model. Generate a notify event in the trigger model that feeds the stimulus setting to a digital I/O line to pulse as needed in the trigger model. See: • :TRIGger:BLOCk:NOTify (on page 6-146) • :TRIGger:DIGital<n>:OUT:STIMulus (on page 6-153)

Model 2400 command	:ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:OUTPut :ARM[:SEQuence[1]][:LAYer[1]][:TCONfigure]:OUTPut?
Model 2450 command	Not available
Notes	Similar functionality available with :TRIGger:BLOCk:NOTify (on page 6-146) set to a digital I/O line.

Model 2400 command	:ARM[:SEQuence[1]][:LAYer[1]]:TIMer :ARM[:SEQuence[1]][:LAYer[1]]:TIMer?
Model 2450 command	Not available
Notes	Similar functionality available using :TRIGger:TIMer<n>:COUNt (on page 6-167) and :TRIGger:BLOCk:WAIT (on page 6-148).

Model 2400 command	:INITiate[:IMMediate]
Model 2450 command	:INITiate[:IMMediate] (on page 6-124)
Notes	

Model 2400 command	:TRIGger:CLEar
Model 2450 command	Not available
Notes	

Model 2400 command	:TRIGger[:SEQuence[1]]:COUNt :TRIGger[:SEQuence[1]]:COUNt?
Model 2450 command	Not available
Notes	Similar functionality availability using :TRIGger:BLOCk:BRANch:COUNter (on page 6-128).

Model 2400 command	:TRIGger[:SEQuence[1]]:DELay :TRIGger[:SEQuence[1]]:DELay?
Model 2450 command	Not available
Notes	Similar functionality availability using :TRIGger:BLOCk:DELay:CONStant (on page 6-139).

Model 2400 command	:TRIGger[:SEQuence[1]]:SOURce :TRIGger[:SEQuence[1]]:SOURce?
Model 2450 command	Not available
Notes	

Model 2400 command	:TRIGger[:SEQuence[1]][:TCONfigure][:ASYNchronous]:INPut :TRIGger[:SEQuence[1]][:TCONfigure][:ASYNchronous]:INPut?
Model 2450 command	Not available
Notes	

Model 2400 command	`:TRIGger[:SEQuence[1]][:TCONfigure]:DIRection` `:TRIGger[:SEQuence[1]][:TCONfigure]:DIRection?`
Model 2450 command	Not available
Notes	Similar functionality available with :TRIGger:BLOCk:WAIT (on page 6-148).

Model 2400 command	`:TRIGger[:SEQuence[1]][:TCONfigure]:ILINe` `:TRIGger[:SEQuence[1]][:TCONfigure]:ILINe?`
Model 2450 command	Not available
Notes	For similar functionality, use a digital I/O line with the trigger model. Generate a notify event in the trigger model that feeds the stimulus setting to a digital I/O line to pulse as needed in the trigger model. See: • :TRIGger:BLOCk:NOTify (on page 6-146) • :TRIGger:DIGital<n>:OUT:STIMulus (on page 6-153)

Model 2400 command	`:TRIGger[:SEQuence[1]][:TCONfigure]:OLINe` `:TRIGger[:SEQuence[1]][:TCONfigure]:OLINe?`
Model 2450 command	Not available
Notes	For similar functionality, use a digital I/O line with the trigger model. Generate a notify event in the trigger model that feeds the stimulus setting to a digital I/O line to pulse as needed in the trigger model. See: • :TRIGger:BLOCk:NOTify (on page 6-146) • :TRIGger:DIGital<n>:OUT:STIMulus (on page 6-153)

Model 2400 command	`:TRIGger[:SEQuence[1]][:TCONfigure]:OUTPut` `:TRIGger[:SEQuence[1]][:TCONfigure]:OUTPut?`
Model 2450 command	Not available
Notes	Similar functionality available with :TRIGger:BLOCk:NOTify (on page 6-146) set to a digital I/O line.

Model 2400 command	`:TRIGger:SEQuence2:SOURce` `:TRIGger:SEQuence2:SOURce?`
Model 2450 command	Not available
Notes	

Model 2400 command	`:TRIGger:SEQuence2:TOUT` `:TRIGger:SEQuence2:TOUT?`
Model 2450 command	Not available
Notes	

Common commands

Model 2400 command	Model 2450 command	Notes
`*CLS`	*CLS (on page B-2)	Model 2450 has fewer registers
`*ESE` `*ESE?`	*ESE (on page B-2) `*ESE?`	Model 2450 has changes to registers
`*ESR?`	*ESR? (on page B-4)	
`*IDN?`	*IDN? (on page B-5)	
`*OPC` `*OPC?`	*OPC (on page B-6) `*OPC?`	
`*OPT?`	Not available	Supported the contact check feature, which is not available on the Model 2450
`*RCL`	*RCL (on page 6-1)	
`*RST`	*RST (on page B-7)	
`*SAV`	*SAV (on page 6-2)	
`*SRE`	*SRE (on page B-8)	
`*STB?`	*STB? (on page B-9)	
`*TRG`	*TRG (on page B-9)	
`*TST?`	*TST? (on page B-10)	Command accepted and returns 0
`*WAI`	*WAI (on page B-10)	

Index

T

U

V

W