

# Series 2600 System SourceMeter® Instruments

## Reference Manual

260x-901-01 Rev. A / May 2005

**KEITHLEY**

A G R E A T E R M E A S U R E O F C O N F I D E N C E

# WARRANTY

Keithley Instruments, Inc. warrants this product to be free from defects in material and workmanship for a period of 1 year from date of shipment.

Keithley Instruments, Inc. warrants the following items for 90 days from the date of shipment: probes, cables, rechargeable batteries, diskettes, and documentation.

During the warranty period, we will, at our option, either repair or replace any product that proves to be defective.

To exercise this warranty, write or call your local Keithley representative, or contact Keithley headquarters in Cleveland, Ohio. You will be given prompt assistance and return instructions. Send the product, transportation prepaid, to the indicated service facility. Repairs will be made and the product returned, transportation prepaid. Repaired or replaced products are warranted for the balance of the original warranty period, or at least 90 days.

## LIMITATION OF WARRANTY

This warranty does not apply to defects resulting from product modification without Keithley's express written consent, or misuse of any product or part. This warranty also does not apply to fuses, software, non-rechargeable batteries, damage from battery leakage, or problems arising from normal wear or failure to follow instructions.

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES.

NEITHER KEITHLEY INSTRUMENTS, INC. NOR ANY OF ITS EMPLOYEES SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF ITS INSTRUMENTS AND SOFTWARE EVEN IF KEITHLEY INSTRUMENTS, INC., HAS BEEN ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH DAMAGES. SUCH EXCLUDED DAMAGES SHALL INCLUDE, BUT ARE NOT LIMITED TO: COSTS OF REMOVAL AND INSTALLATION, LOSSES SUSTAINED AS THE RESULT OF INJURY TO ANY PERSON, OR DAMAGE TO PROPERTY.

**KEITHLEY**

A GREATER MEASURE OF CONFIDENCE

**Keithley Instruments, Inc.**

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139  
440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY (534-8453) • [www.keithley.com](http://www.keithley.com)

---

# Series 2600 System SourceMeter<sup>®</sup> Instruments Reference Manual

©2005, Keithley Instruments, Inc.  
All rights reserved.  
Cleveland, Ohio, U.S.A.  
First Printing, May 2005  
Document Number: 260x-901-01 Rev. A

---

# Manual Print History

The print history shown below lists the printing dates of all Revisions and Addenda created for this manual. The Revision Level letter increases alphabetically as the manual undergoes subsequent updates. Addenda, which are released between Revisions, contain important change information that the user should incorporate immediately into the manual. Addenda are numbered sequentially. When a new Revision is created, all Addenda associated with the previous Revision of the manual are incorporated into the new Revision of the manual. Each new Revision includes a revised copy of this print history page.

Revision A (Document Number 260x-901-01) ..... May 2005

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 non-hazardous 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 manual for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product 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 manual. 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, and perform safe installations and repairs of products. Only properly trained service personnel may perform installation and service procedures.

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

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

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 volts, **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, make sure 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 ca-

bles or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

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


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


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


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


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

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

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

The  symbol on an instrument indicates that the user should refer to the operating instructions located in the manual.

The  symbol on an instrument shows that it can source or measure 1000 volts or more, including the combined effect of normal and common mode voltages. Use standard safety precautions to avoid personal contact with these voltages.

The  symbol indicates a connection terminal to the equipment frame.

The **WARNING** heading in a manual 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 a manual 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., data acquisition board for installation into a computer) should never require cleaning if handled according to instructions. If the board becomes contaminated and operation is affected, the board should be returned to the factory for proper cleaning/servicing.

---

# Table of Contents

<b>1</b>	<b>Getting Started</b>	
	Introduction .....	1-2
	Capabilities and features .....	1-2
	Organization of manual sections .....	1-2
	General information .....	1-3
	Warranty information .....	1-3
	Contact information .....	1-3
	Safety symbols and terms .....	1-3
	Unpacking and inspection .....	1-4
	Options and accessories .....	1-5
	User's manual .....	1-6
	Reference manual .....	1-6
	Front and rear panel familiarization .....	1-6
	Front panel summaries .....	1-6
	Rear panel summaries .....	1-6
	Cooling vents .....	1-12
	Power-up .....	1-13
	Line power connection .....	1-13
	Power-up sequence .....	1-14
	Beeper .....	1-14
	Display modes .....	1-15
	Editing controls .....	1-16
	Source and compliance editing .....	1-16
	Menu navigation .....	1-17
	Menus .....	1-18
	Main menu .....	1-18
	Configuration menus .....	1-18
	Interface selection .....	1-21
	GPIB interface .....	1-21
	RS-232 interface .....	1-21
	Error and status messages .....	1-21
	Default settings .....	1-22
	Front panel setups .....	1-22
	Remote operation setups .....	1-23
	Remote programming .....	1-25
	Requesting readings .....	1-25
	Requesting command settings .....	1-25

---

## 2

# TSP Programming Fundamentals

Introduction .....	2-2
Test Script Processor (TSP) .....	2-2
Run-time environment .....	2-3
Queries .....	2-3
Scripts .....	2-3
Named scripts .....	2-4
Functions .....	2-4
Scripts that create functions .....	2-5
Programming overview .....	2-6
What is a chunk? .....	2-6
What is a script? .....	2-6
Run-time environment .....	2-7
Non-volatile memory .....	2-8
TSP programming levels .....	2-8
Programming model for scripts .....	2-8
Installing the TSP software .....	2-10
System connections .....	2-10
GPIB .....	2-10
RS-232 .....	2-11
Using Test Script Builder .....	2-12
Starting Test Script Builder .....	2-14
Opening communications .....	2-15
Creating and modifying a script .....	2-17
Script launch configuration .....	2-22
Launching a script .....	2-25
Running a TSP file .....	2-26
Retrieving scripts from the Model 260x .....	2-26
Instrument Console .....	2-28
File management tasks .....	2-34
Sending commands and statements .....	2-38
Source-measure voltage and current .....	2-38
Read and write to Digital I/O port .....	2-38
Display user-defined messages .....	2-38
User scripts .....	2-39
Script examples .....	2-39
Creating a user script .....	2-42
Saving a user script .....	2-43
Running a user script .....	2-43
Modifying a user script .....	2-46
Script management .....	2-46
Factory scripts .....	2-48
Running a factory script .....	2-48
Modifying a factory script .....	2-48



---

Differences: remote vs. local state .....	2-49
Memory considerations .....	2-51
Test Script Language (TSL) reference .....	2-52
Introduction .....	2-52
Reserved words .....	2-52
Variables and types .....	2-52
Operators .....	2-53
Tables/arrays .....	2-53
Functions .....	2-54
Precedence .....	2-55
Logical operators .....	2-55
Concatenation .....	2-56
Branching .....	2-57
Loop control .....	2-58
Standard libraries .....	2-59

### **3 DUT Test Connections**

Input/Output terminal blocks .....	3-2
Input/Output LO and chassis ground .....	3-3
Sensing methods .....	3-5
2-wire local sensing .....	3-5
4-wire remote sensing .....	3-5
Sense mode selection .....	3-6
Multiple SMU connections .....	3-7
Guarding and shielding .....	3-8
Guarding .....	3-8
Noise shield .....	3-9
Safety shield .....	3-10
Using shielding and guarding together .....	3-11
Test fixture .....	3-12
Floating an SMU .....	3-13
Output-off states .....	3-15
Selecting the Output-off state .....	3-16

### **4 Basic Operation**

Overview .....	4-2
Operation overview .....	4-2
Source-measure capabilities .....	4-2
Compliance limit .....	4-3
Setting the compliance limit .....	4-4
Basic circuit configurations .....	4-5

---

Operation considerations .....	4-7
Warm-up .....	4-7
Auto zero .....	4-7
NPLC caching .....	4-7
Triggering .....	4-9
Triggering types .....	4-9
Measurement triggering .....	4-10
Front panel triggering .....	4-11
Remote triggering .....	4-12
Basic source-measure procedure .....	4-13
Front panel source-measure procedure .....	4-13
Remote source-measure procedure .....	4-14
Measure only .....	4-16
Sink operation .....	4-17
Ohms measurements .....	4-18
Ohms calculations .....	4-18
Ohms ranging .....	4-18
Basic ohms measurement procedure .....	4-18
Ohms sensing .....	4-19
Sense selection .....	4-20
Remote ohms programming .....	4-21
Power measurements .....	4-22
Power calculations .....	4-22
Basic power measurement procedure .....	4-22
Remote power programming .....	4-23

## 5 Sweep Operation

Overview .....	5-2
Section overview .....	5-2
Sweep overview .....	5-2
Sweep characteristics .....	5-3
Linear staircase sweeps .....	5-3
Logarithmic staircase sweeps .....	5-4
Pulse sweeps .....	5-6
Custom (list) sweeps .....	5-7
Sweep measurement storage .....	5-8
Sweep functions .....	5-8
Staircase sweep functions .....	5-9
Pulse sweep functions .....	5-10
Custom sweep functions .....	5-10
Running sweeps .....	5-11
Front panel .....	5-11
Sweep programming examples .....	5-11
Custom sweep example .....	5-14

---

<b>6</b>	<b>Range, Digits, Speed, Rel, and Filters</b>	
	Overview .....	6-2
	Range .....	6-2
	Available ranges .....	6-2
	Maximum source values and readings .....	6-3
	Ranging limitations .....	6-3
	Manual ranging .....	6-3
	Auto ranging .....	6-3
	Low range limits .....	6-4
	Range considerations .....	6-4
	Range programming .....	6-5
	Digits .....	6-7
	Setting display resolution .....	6-7
	Remote digits programming .....	6-7
	Speed .....	6-8
	Setting speed .....	6-9
	Remote speed programming .....	6-9
	Rel .....	6-10
	Front panel rel .....	6-10
	Remote rel programming .....	6-11
	Filters .....	6-12
	Filter types .....	6-12
	Response time considerations .....	6-12
	Front panel filter control .....	6-12
	Remote filter programming .....	6-15
<b>7</b>	<b>Buffer (Data Store)</b>	
	Overview .....	7-2
	Data store overview .....	7-2
	Front panel data store .....	7-2
	Buffer configuration .....	7-2
	Buffer configuration menu .....	7-3
	Storing readings .....	7-3
	Recalling readings .....	7-4
	Remote data store .....	7-5
	Data store commands .....	7-5
	Reading buffers .....	7-6
	Time and date values .....	7-10
	Buffer status .....	7-10
	Dynamically allocated buffers .....	7-11
	Buffer programming examples .....	7-12

---

## 8 Source-Measure Concepts

Overview .....	8-2
Compliance limit .....	8-2
Maximum compliance .....	8-2
Compliance principles .....	8-3
Sweep waveforms .....	8-4
Staircase sweeps .....	8-4
Pulse sweeps .....	8-5
Overheating protection .....	8-6
Power equations to avoid overheating .....	8-6
Power calculations .....	8-6
Operating boundaries .....	8-7
Source or sink .....	8-7
I -Source operating boundaries .....	8-8
V-Source operating boundaries .....	8-12
Source I measure I, source V measure V .....	8-16
Basic circuit configurations .....	8-16
Source I .....	8-16
Source V .....	8-17
Measure only (V or I) .....	8-18
Guard .....	8-20
Guard overview .....	8-20
Guard connections .....	8-20
Pulse concepts .....	8-23
Pulse period .....	8-23
Pulse rise and fall times .....	8-23
Pulse duty cycle .....	8-24

## 9 System Expansion (TSP-Link)

Overview .....	9-2
Master and Slaves .....	9-2
System configurations .....	9-2
Connections .....	9-3
Initialization .....	9-3
Assigning node numbers .....	9-3
Resetting the TSP-Link .....	9-4
Using the expanded system .....	9-6
Accessing nodes .....	9-6
System behavior .....	9-7

---

<b>10</b>	<b>Digital I/O and Output Enable</b>	
	Overview .....	10-2
	Digital I/O port .....	10-2
	Port configuration .....	10-2
	Digital I/O configuration .....	10-4
	Controlling digital I/O lines .....	10-4
	Output enable .....	10-8
	Overview .....	10-8
	Operation .....	10-8
	Control .....	10-9
<b>11</b>	<b>Communications Interfaces</b>	
	Overview .....	11-2
	Selecting an interface .....	11-2
	GPIB operation .....	11-3
	GPIB standards .....	11-3
	GPIB connections .....	11-3
	Primary address .....	11-5
	Terminator .....	11-6
	General bus commands .....	11-7
	REN (remote enable) .....	11-7
	IFC (interface clear) .....	11-7
	LLO (local lockout) .....	11-8
	GTL (go to local) .....	11-8
	DCL (device clear) .....	11-8
	SDC (selective device clear) .....	11-8
	GET (group execute trigger) .....	11-8
	SPE, SPD (serial polling) .....	11-8
	Front panel GPIB operation .....	11-9
	Error and status messages .....	11-9
	GPIB status indicators .....	11-9
	LOCAL key .....	11-10
	RS-232 interface operation .....	11-10
	Setting RS-232 interface parameters .....	11-10
	Sending and receiving data .....	11-11
	Terminator .....	11-11
	Baud rate .....	11-12
	Data bits and parity .....	11-12
	Flow control (signal handshaking) .....	11-12
	RS-232 connections .....	11-13
	Error messages .....	11-14

---

## 12 Instrument Control Library

Command programming notes .....	12-2
Conventions .....	12-2
Functions and attributes .....	12-3
TSP-Link nodes .....	12-5
Logical instruments .....	12-5
Reading buffers .....	12-6
Time and date values .....	12-8
ICL functions and attributes list .....	12-9
beeper function and attribute .....	12-11
bit functions .....	12-12
delay function .....	12-18
digio functions and attributes .....	12-19
display functions and attributes .....	12-24
errorqueue functions and attribute .....	12-40
exit function .....	12-42
format attributes .....	12-42
gpib attribute .....	12-45
localnode attributes .....	12-46
makegetter functions .....	12-48
opc function .....	12-50
printbuffer and printnumber functions .....	12-51
reset function .....	12-53
serial functions and attributes .....	12-53
setup functions and attribute .....	12-56
smuX functions and attributes .....	12-57
timer functions .....	12-86
trigger functions .....	12-87
tslink function and attributes .....	12-88
userstring functions .....	12-89
waitcomplete function .....	12-91

## 13 Factory Scripts

Introduction .....	13-2
Factory script .....	13-2
KIGeneral .....	13-2
Flash firmware upgrade .....	13-13

---

## 14

### Display Operations

Display functions and attributes .....	14-2
Display features .....	14-3
Display screen .....	14-3
Measurement functions .....	14-3
Display resolution .....	14-4
Display messages .....	14-4
Clearing the display .....	14-5
Cursor position .....	14-5
Displaying text messages .....	14-6
Input prompting .....	14-8
Menu .....	14-8
Parameter value prompting .....	14-10
Annunciators .....	14-11
LOCAL lockout .....	14-12
Load test menu .....	14-13
Saving a user script .....	14-13
Adding USER TESTS menu entries .....	14-13
Deleting USER TESTS menu entries .....	14-14
Running a test from the front panel .....	14-14
Display triggering .....	14-15
Key-press codes .....	14-16
Sending keycodes .....	14-16
Capturing key-press codes .....	14-17

## 15

### Performance Verification

Introduction .....	15-2
Verification test requirements .....	15-2
Environmental conditions .....	15-2
Warm-up period .....	15-3
Line power .....	15-3
Recommended test equipment .....	15-3
Verification limits .....	15-4
Restoring factory defaults .....	15-5
Performing the verification test procedures .....	15-5
Test summary .....	15-5
Test considerations .....	15-5
Setting the source range and output value .....	15-6
Setting the measurement range .....	15-6
Output voltage accuracy .....	15-7
Voltage measurement accuracy .....	15-8
Output current accuracy .....	15-9
Current measurement accuracy .....	15-11

---

<b>16</b>	<b>Calibration</b>	
	Introduction .....	16-2
	Environmental conditions .....	16-2
	Temperature and relative humidity .....	16-2
	Warm-up period .....	16-2
	Line power .....	16-2
	Calibration considerations .....	16-3
	Calibration cycle .....	16-3
	Recommended calibration equipment .....	16-4
	Calibration errors .....	16-5
	Calibration .....	16-5
	Calibration steps .....	16-5
	Calibration commands .....	16-6
	Calibration procedure .....	16-8
<b>17</b>	<b>Routine Maintenance</b>	
	Introduction .....	17-2
	Line fuse replacement .....	17-2
	Front panel tests .....	17-3
	KEYS test .....	17-3
	DISPLAY PATTERNS test .....	17-4
<b>A</b>	<b>Specifications</b>	
<b>B</b>	<b>Error and Status Messages</b>	
	Introduction .....	B-2
	Error summary .....	B-2
	Reading errors .....	B-2
<b>C</b>	<b>Common Commands</b>	
	Introduction .....	C-2
	Common commands .....	C-2
	Command summary .....	C-2
	Script command equivalents .....	C-3
	Command reference .....	C-4



---

<b>D</b>	<b>Status Model</b>	
	Overview .....	D-2
	Status byte and SRQ .....	D-2
	Status register sets .....	D-2
	Queues .....	D-2
	Status function summary .....	D-8
	Clearing registers and queues .....	D-9
	Programming and reading registers .....	D-10
	Programming enable and transition registers .....	D-10
	Reading registers .....	D-11
	Status byte and service request (SRQ) .....	D-11
	Status byte register .....	D-11
	Service request enable register .....	D-13
	Serial polling and SRQ .....	D-14
	SPE, SPD (serial polling) .....	D-14
	Status byte and service request commands .....	D-14
	Enable and transition registers .....	D-15
	Controlling node and SRQ enable registers .....	D-15
	Status register sets .....	D-17
	System Summary Event Registers .....	D-17
	Standard Event Register .....	D-19
	Operation Event Registers .....	D-21
	Questionable Event Registers .....	D-24
	Measurement Event Registers .....	D-26
	Register programming example .....	D-29
	Queues .....	D-29
	Output queue .....	D-29
	Error queue .....	D-29

<b>E</b>	<b>Speed Specification Test Conditions</b>	
	Introduction .....	E-2
	Test system used .....	E-2
	Overview .....	E-2
	Sweep Operation Rates .....	E-2
	Single Measurement Rates .....	E-3
	Function and Range Change Rates .....	E-3
	Command Processing .....	E-4
	Sweep Operation Rates .....	E-4
	Measure to Memory .....	E-5
	Measure to GPIB .....	E-5
	Source Measure to Memory .....	E-6
	Source Measure to GPIB .....	E-6
	Source Measure Pass/Fail to Memory .....	E-6
	Source Measure Pass/Fail to GPIB .....	E-6

---

Single Measurement Rates .....	E-6
Measure to GPIB .....	E-7
Source Measure to GPIB .....	E-7
Source Measure Pass/Fail to GPIB .....	E-7
Function/ Range Change Rates .....	E-8
Source Range Change Rate .....	E-8
Measure Range Change Rate .....	E-8
Function Change Rate .....	E-8
Command Processing .....	E-9

---

# List of Illustrations

## 1 Getting Started

Figure 1-1	Models 2601 and 2602 front panels .....	1-7
Figure 1-2	Models 2601 and 2602 rear panels .....	1-10
Figure 1-3	Display modes .....	1-15

## 2 TSP Programming Fundamentals

Figure 2-1	Script example .....	2-7
Figure 2-2	Programming model for scripts .....	2-9
Figure 2-3	GPIB cable .....	2-10
Figure 2-4	RS-232 cable (straight-through) .....	2-11
Figure 2-5	Test Script Builder (example) .....	2-13
Figure 2-6	Opening and closing communications .....	2-16
Figure 2-7	Creating and modifying a script using the Test Script Builder .....	2-17
Figure 2-8	Creating a project folder .....	2-18
Figure 2-9	Saving a script in the Test Script Builder .....	2-19
Figure 2-10	Creating a new script file .....	2-20
Figure 2-11	Renaming a project folder and/or script file .....	2-21
Figure 2-12	Changing a launch configuration .....	2-22
Figure 2-13	Opening the Run dialog box (launch configuration) .....	2-23
Figure 2-14	Run dialog box (Script Attributes tab) .....	2-25
Figure 2-15	Re-launching a script from the Test Script Builder toolbar .....	2-25
Figure 2-16	Re-launching a script from the Test Script Builder toolbar .....	2-26
Figure 2-17	Importing a script (e.g., KIGeneral_Script) from memory of the Model 260x .....	2-27
Figure 2-18	Instrument Console icons .....	2-28
Figure 2-19	Programming interaction tabs: Problems, Tasks and Command Help ..	2-32
Figure 2-20	Programming interaction tabs: Language Help, Bookmarks, Browser View .....	2-33
Figure 2-21	Workspace Launcher and Select Workspace Directory .....	2-35
Figure 2-22	Importing a project from another workspace folder .....	2-36
Figure 2-23	Deleting a project .....	2-37

## 3 DUT Test Connections

Figure 3-1	Input/output terminal blocks .....	3-3
Figure 3-2	Input/Output LO and chassis ground terminals .....	3-4
Figure 3-3	Low-Noise Chassis Ground Banana Jack and Chassis Screw .....	3-4
Figure 3-4	2-wire connections (local sense) .....	3-5
Figure 3-5	4-wire connections (remote sense) .....	3-6
Figure 3-6	Two SMUs connected to a 3-terminal device (local sense) .....	3-7
Figure 3-7	Three SMUs connected to a 3-terminal device (local sense) .....	3-8
Figure 3-8	High-impedance guarding .....	3-9

Figure 3-9	Noise shield.....	3-9
Figure 3-10	Safety shield for hazardous voltage (>42V) .....	3-10
Figure 3-11	Connections for test circuit shown in Figure 3-10 .....	3-11
Figure 3-12	Connections for noise shield, safety shield and guarding .....	3-11
Figure 3-13	Floating the Model 260x.....	3-14
Figure 3-14	SMU connections for the floating configuration shown in Figure 3-13 ..	3-14

## **4 Basic Operation**

Figure 4-1	Fundamental source measure configuration.....	4-6
Figure 4-2	Measurement triggering sequence.....	4-10
Figure 4-3	2-wire resistance sensing.....	4-19
Figure 4-4	4-wire resistance sensing.....	4-20

## **5 Sweep Operation**

Figure 5-1	Comparison of staircase sweep types.....	5-3
Figure 5-2	Linear staircase sweep.....	5-4
Figure 5-3	Logarithmic staircase sweep (1V to 10V, five steps).....	5-5
Figure 5-4	Pulse sweep example .....	5-7
Figure 5-5	Custom sweep example .....	5-7

## **6 Range, Digits, Speed, Rel, and Filters**

Figure 6-1	Moving average and repeating filters .....	6-14
------------	--	------

## **8 Source-Measure Concepts**

Figure 8-1	Two basic sweep waveforms .....	8-4
Figure 8-2	Pulse sweep example .....	8-5
Figure 8-3	Continuous power operating boundaries .....	8-7
Figure 8-4	I-Source boundaries.....	8-9
Figure 8-5	I-Source operating boundaries.....	8-11
Figure 8-6	V-Source boundaries.....	8-13
Figure 8-7	V-Source operating examples .....	8-15
Figure 8-8	Source I configuration .....	8-17
Figure 8-9	Source V configuration .....	8-18
Figure 8-10	Measure only configurations .....	8-19
Figure 8-11	Comparison of unguarded and guarded measurements.....	8-22
Figure 8-12	Pulse period .....	8-23
Figure 8-13	Pulse rise and fall times .....	8-24

## **9 System Expansion (TSP-Link)**

Figure 9-1	TSP-Link connections .....	9-3
------------	----------------------------	-----

---

<b>10</b>	<b>Digital I/O and Output Enable</b>	
Figure 10-1	Digital I/O port .....	10-3
Figure 10-2	Digital I/O port configuration.....	10-4
Figure 10-3	Using output enable .....	10-9
<b>11</b>	<b>Communications Interfaces</b>	
Figure 11-1	IEEE-488 connector .....	11-3
Figure 11-2	IEEE-488 connections.....	11-4
Figure 11-3	IEEE-488 and RS-232 connector locations.....	11-5
Figure 11-4	RS-232 interface connector.....	11-13
<b>14</b>	<b>Display Operations</b>	
Figure 14-1	Row/column format for display messaging.....	14-6
<b>15</b>	<b>Performance Verification</b>	
Figure 15-1	Connections for voltage verification .....	15-7
Figure 15-2	Current verification connections (100nA to 1A ranges).....	15-10
Figure 15-3	Current verification connections (3A range).....	15-11
<b>16</b>	<b>Calibration</b>	
Figure 16-1	Connections for voltage calibration .....	16-9
Figure 16-2	Connections for current calibration (100nA to 1A ranges) .....	16-13
Figure 16-3	Connections for current calibration (3A and 10A ranges) .....	16-17
<b>17</b>	<b>Routine Maintenance</b>	
Figure 17-1	Line fuse replacement.....	17-2
<b>D</b>	<b>Status Model</b>	
Figure D-1	Status model overview .....	D-3
Figure D-2	Status model (system summary and standard event registers) .....	D-4
Figure D-3	Status model (operation event registers).....	D-5
Figure D-4	Status model (questionable event registers) .....	D-6
Figure D-5	Status model (measurement event registers).....	D-7
Figure D-6	16-bit status register .....	D-10
Figure D-7	Status byte and service request (SRQ).....	D-12
Figure D-8	Standard event register .....	D-20

---

# List of Tables

<b>1</b>	<b>Getting Started</b>	
Table 1-1	Main menu.....	1-19
Table 1-2	Configuration menus .....	1-20
Table 1-3	Default settings.....	1-24
<b>2</b>	<b>TSP Programming Fundamentals</b>	
Table 2-1	Example script to sweep V and measure I.....	2-39
Table 2-2	Example script using a function.....	2-40
Table 2-3	Example interactive chunk fragment for a script .....	2-41
<b>3</b>	<b>DUT Test Connections</b>	
Table 3-1	Selecting the sense mode from the front panel.....	3-6
Table 3-2	Commands to select sense mode .....	3-7
Table 3-3	Commands to select Output-off state .....	3-16
<b>4</b>	<b>Basic Operation</b>	
Table 4-1	Source-measure capabilities .....	4-3
Table 4-2	Maximum compliance values .....	4-4
Table 4-3	Compliance commands .....	4-5
Table 4-4	Auto zero settings.....	4-8
Table 4-5	Auto zero command and options.....	4-9
Table 4-6	Trigger commands .....	4-12
Table 4-7	Basic source-measure commands .....	4-15
<b>5</b>	<b>Sweep Operation</b>	
Table 5-1	Logarithmic sweep points.....	5-6
Table 5-2	Staircase sweep functions.....	5-9
Table 5-3	Pulse sweep functions.....	5-10
Table 5-4	Custom sweep functions .....	5-10
Table 5-5	Sweep example parameters.....	5-11
<b>6</b>	<b>Range, Digits, Speed, Rel, and Filters</b>	
Table 6-1	Source and measurement ranges .....	6-2
Table 6-2	Range commands .....	6-6
Table 6-3	Digits commands.....	6-8
Table 6-4	Speed command .....	6-9
Table 6-5	Rel commands .....	6-11
Table 6-6	Filter commands.....	6-15

---

<b>7</b>	<b>Buffer (Data Store)</b>	
Table 7-1	Data store commands .....	7-5
Table 7-2	Buffer storage control attributes .....	7-7
Table 7-3	Buffer read-only attributes .....	7-7
Table 7-4	Buffer control programming examples .....	7-8
Table 7-5	Buffer read-only attribute programming examples .....	7-8
Table 7-6	Recall attributes .....	7-9
Table 7-7	Buffer status bits.....	7-10
<b>8</b>	<b>Source-Measure Concepts</b>	
Table 8-1	Maximum compliance values .....	8-3
<b>9</b>	<b>System Expansion (TSP-Link)</b>	
Table 9-1	Assigning a node number to an instrument from the front panel.....	9-4
Table 9-2	Resetting the TSP-Link from the front panel .....	9-5
Table 9-3	TSP-Link reset commands .....	9-5
<b>10</b>	<b>Digital I/O and Output Enable</b>	
Table 10-1	Digital I/O bit weighting.....	10-5
Table 10-2	Digital I/O commands .....	10-7
<b>11</b>	<b>Communications Interfaces</b>	
Table 11-1	General bus commands .....	11-7
Table 11-2	RS-232 interface commands.....	11-11
Table 11-3	RS-232 connector pinout .....	11-13
Table 11-4	PC serial port pinout.....	11-14
<b>14</b>	<b>Display Operations</b>	
Table14-1	Cross referencing functions/attributes to section topics .....	14-2
Table14-2	Bit identification for annunciators .....	14-12
Table14-3	Keycodes to send for display.sendkey .....	14-16
Table14-4	Keycode values returned for display.getlastkey .....	14-17
<b>15</b>	<b>Performance Verification</b>	
Table 15-1	Recommended verification equipment.....	15-4
Table 15-2	Output voltage accuracy limits .....	15-8
Table 15-3	Voltage measurement accuracy limits.....	15-9
Table 15-4	Output current accuracy limits.....	15-10
Table 15-5	Current measurement accuracy limits.....	15-12

---

<b>16</b>	<b>Calibration</b>	
Table 16-1	Recommended calibration equipment.....	16-4
Table 16-2	Calibration steps.....	16-6
Table 16-3	Calibration commands.....	16-7
<b>17</b>	<b>Routine Maintenance</b>	
Table 17-1	Line fuse.....	17-3
<b>B</b>	<b>Error and Status Messages</b>	
Table B-1	Error queue commands.....	B-2
Table B-2	Error summary.....	B-3
<b>C</b>	<b>Common Commands</b>	
Table C-1	Common commands .....	C-2
Table C-2	Script command equivalents .....	C-3
<b>D</b>	<b>Status Model</b>	
Table D-1	Status functions and registers .....	D-8
Table D-2	Commands to reset registers and clear queues.....	D-9
Table D-3	Status Byte and Service Request Enable Register commands.....	D-15
Table D-4	System node and SRQ enable register bit attributes .....	D-16
Table D-5	System summary event commands .....	D-17
Table D-6	Standard event commands.....	D-20
Table D-7	Status event status registers and bits.....	D-21
Table D-8	Operation event commands .....	D-22
Table D-9	Questionable event commands.....	D-24
Table D-10	Measurement event commands .....	D-27
Table D-11	Error queue commands.....	D-30



# 1 Getting Started

---

## Section 1 topics

### Introduction, page 1-2

- Capabilities and features, page 1-2
- Organization of manual sections, page 1-2

### General information, page 1-3

- Warranty information, page 1-3
- Contact information, page 1-3
- Safety symbols and terms, page 1-3
- Unpacking and inspection, page 1-4
- Options and accessories, page 1-5
- User's manual, page 1-6
- Reference manual, page 1-6

### Front and rear panel familiarization, page 1-6

- Front panel summaries, page 1-6
- Rear panel summaries, page 1-6

### Cooling vents, page 1-12

### Power-up, page 1-13

- Line power connection, page 1-13
- Power-up sequence, page 1-14
- Beeper, page 1-14

### Display modes, page 1-15

### Editing controls, page 1-16

- Source and compliance editing, page 1-16
- Menu navigation, page 1-17

### Menus, page 1-18

- Main menu, page 1-18
- Configuration menus, page 1-18

### Interface selection, page 1-21

- GPIB interface, page 1-21
- RS-232 interface, page 1-21

### Error and status messages, page 1-21

### Default settings, page 1-22

- Front panel setups, page 1-22
- Remote operation setups, page 1-23

### Remote programming, page 1-25

- Requesting readings, page 1-25
- Requesting command settings, page 1-25

# Introduction

## Capabilities and features

- Source  $\pm$ DC voltage from 1 $\mu$ V to 40.4V.
- Source  $\pm$ DC current from 1pA to 3.03A.
- Measure  $\pm$ DC voltage from 1 $\mu$ V to 40.4V.
- Measure  $\pm$ DC current from 1pA to 3.03A.
- Resistance and power measurement functions.
- Two independent SourceMeter channels (Model 2602 only).
- 40W, four-quadrant sink or source operation (per channel).
- Embedded Test Script Processor (TSP) accessible from any host interface responds to high-speed test scripts comprised of instrument control commands.
- Factory script sweep functions: linear staircase, logarithmic staircase, fixed pulse, and custom sweeps.
- Five user-saved setups.
- Buffer storage and recall for at least 100,000 source-measure readings.
- Filtering to reduce reading noise.
- Supported remote interfaces: IEEE-488 (GPIB) and RS-232.
- TSP-Link: allows TSP enabled instruments to trigger and communicate with each other.
- Digital I/O port: allows the Model 260x to control other devices.

## Organization of manual sections

While viewing the PDF version of this manual, the manual sections can be viewed by clicking the “Bookmarks” tab on the left side of this window. This tab also provides direct links to the various sections and section topics.

The manual sections are also listed in the Table of Contents located at the beginning of this manual.

# General information

## Warranty information


Warranty information is located at the front of this manual. Should your Model 260x require warranty service, contact the Keithley representative or authorized repair facility in your area for further information. When returning the instrument for repair, be sure to fill out and include the service form at the back of this manual to provide the repair facility with the necessary information.


## Contact information


Worldwide phone numbers are listed at the front of this manual. If you have any questions, please contact your local Keithley representative or call one of our Application Engineers at 888-Keithley (534-8453) or 800-552-1115 (U.S. and Canada only). You can also contact Application Engineering online at [www.keithley.com](http://www.keithley.com).

## Safety symbols and terms

The following symbols and terms may be found on the instrument or used in this manual:

The  symbol on an instrument indicates that the user should refer to the operating instructions located in the manual.

The  symbol on the instrument shows that high voltage may be present on the terminal(s). 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 **WARNING** heading used in this manual 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 used in this manual explains hazards that could damage the instrument. Such damage may invalidate the warranty.

## Unpacking and inspection

### Inspection for damage

The Model 260x was carefully inspected electrically and mechanically before shipment. After unpacking all items from the shipping carton, check for any obvious signs of physical damage that may have occurred during transit. (There may be a protective film over the display lens, which can be removed.) Report any damage to the shipping agent immediately. Save the original packing carton for possible future shipment. Before removing the Model 260x from the bag, observe the following handling precautions.

### Handling precautions

- Always grasp the Model 260x by the covers or by the handle.
- After removing the Model 260x from its anti-static bag, inspect it for any obvious signs of physical damage. Report any such damage to the shipping agent immediately.
- When the Model 260x is not installed and connected, keep the unit in its anti-static bag and store it in the original packing carton.

### Package content

The following items are included with every Model 260x order:

- Model 2601 or Model 2602 SourceMeter with line cord.
- DUT interface connector kit for each SourceMeter channel. Kit includes one hooded screw terminal connector that mates with the SourceMeter measurement terminals.
- TSP-Link cable.
- Accessories as ordered.
- Certificate of calibration.
- Model 260x User's Manual (P/N 2600-900-00).
- CD-ROMs that contain:
  - PDFs of the User's and Reference Manuals.
  - Test Script Builder script development software.
  - IVI/VISA drivers for VB, VC/C++, LabVIEW, TestPoint, and LabWindows/CVI.

## Options and accessories

### Connectors

**SourceMeter DUT Interface Connector Kit** – Includes one hooded screw terminal connector that mates with the SourceMeter measurement terminals. One kit per channel is included with the SourceMeter.

### GPIB cables and adapter (connects Model 260x to the GPIB bus):

**Model 7006-1 and Model 7006-2** – Single-shielded GPIB cables. Terminated with one straight connector (non-stacking) and one feed-through style connector. Model 7006-1 is 1m long; Model 7006-2 is 2m long.

**Models 7007-05, 7007-1, 7007-2 and 7007-4** – Double-shielded premium GPIB cables. Each end is terminated with a feed through metal housing for longest life and best performance. Model 7007-05 is 0.5m long; 7007-1 is 1m long; Model 7007-2 is 2m long; Model 7007-4 is 4m long.

**Model 7010** – Shielded IEEE-to-IEEE Adapter. Provides additional clearance between the rear panel and GPIB cable connector. Allows easier access to cables and other connectors.

### RS-232 cable (connects Model 260x to the RS-232):

**Model 7009-5 shielded RS-232 cable** – This straight-through cable connects the RS-232 of the Model 260x to the RS-232 interface of the PC. This cable is 5ft. long and uses shielded cable and connectors to reduce electromagnetic interference (EMI).

### TSP-Link cable (connects Model 260x to the TSP-Link):

**CA-180-3A CAT 5 cable** – This crossover CAT5 LAN cable connects the TSP-Link of the Model 260x to the TSP-Link of other instruments.

### Digital I/O port cables (connects Digital I/O to other devices):

**Model 2600-TLINK trigger cable** – Connects the Digital I/O port of series 2600 instruments to other Keithley instruments equipped with Trigger Link (TLINK).

**CA-126-1 DB-25 cable** – DB-25 male to female DB-25 cable, 1.5m (5ft) long, used to connect the Digital I/O port to other instruments.

## User's manual

A printed copy of the User's Manual is a supplied item for the Model 260x. It is also provided on the product information CD-ROM as a PDF. This manual provides the fundamental operating information for the instrument.

## Reference manual

The Reference Manual is provided on the product information CD-ROM as a PDF. This manual provides additional information on the topics covered in the User's Manual. It also includes advanced operation topics and maintenance information.

# Front and rear panel familiarization

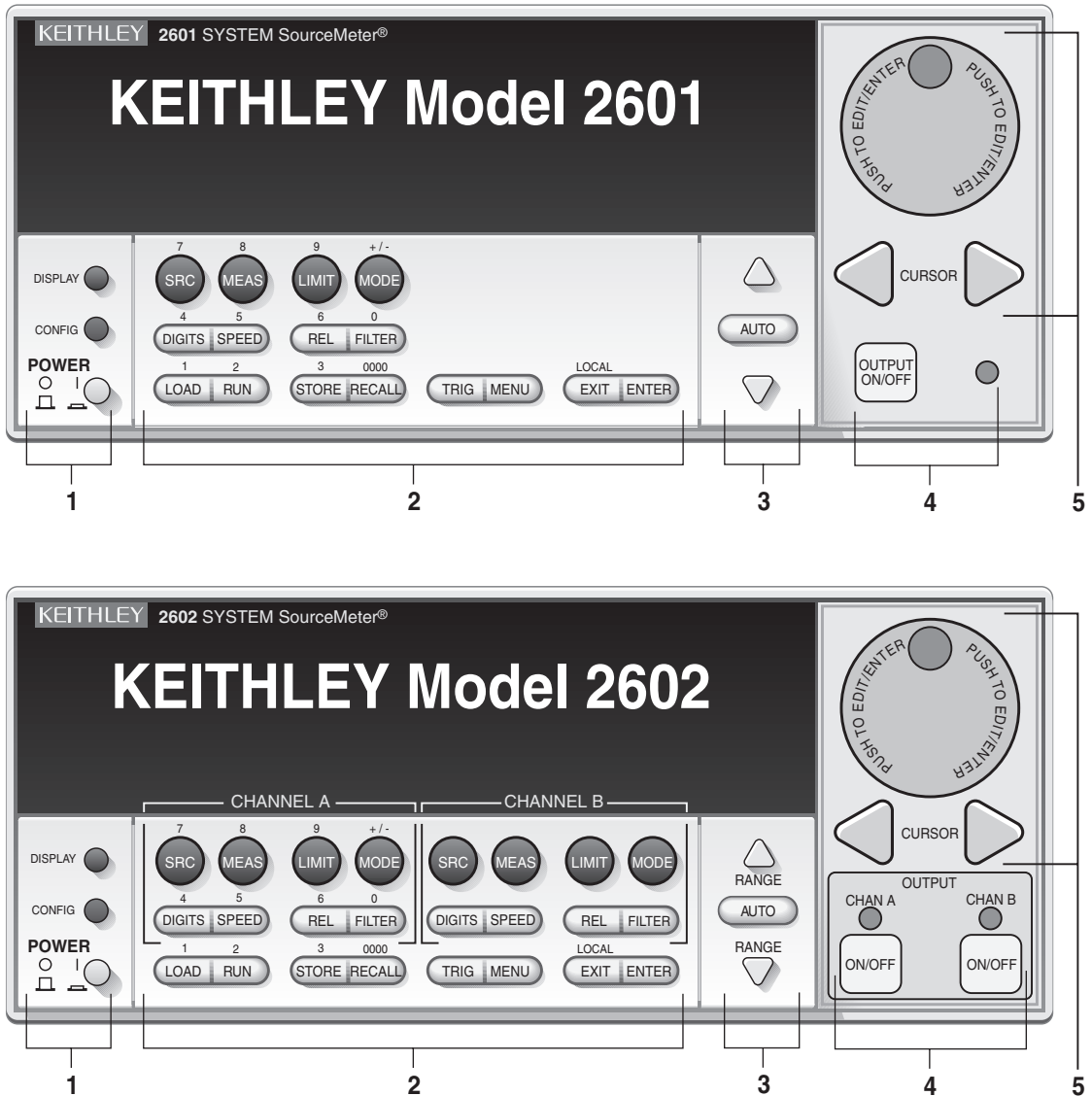
## Front panel summaries

The front panels of the Models 2601 and 2602 are shown in [Figure 1-1 on page 1-7](#). The descriptions of the front panel controls follow [Figure 1-1](#).

## Rear panel summaries

The rear panels of the Models 2601 and 2602 are shown in [Figure 1-2 on page 1-10](#). The descriptions of the rear panel components follow [Figure 1-2](#).

Figure 1-1  
Models 2601 and 2602 front panels



**NOTE** The Model 2601 has one SourceMeter channel (Channel A) and the Model 2602 has two SourceMeter channels (Channel A and Channel B).

## 1 Special keys and power switch:

DISPLAY	Toggles between the various source-measure displays and the user message mode. Selects Model 2602 single or dual-channel display.
CONFIG	Use to configure a function or operation.
POWER	Power switch – In position turns SourceMeter on (I), out position turns it off (O).
Number Keys	The Number Keys (0-9, +/-, 0000) allow direct numeric entry in the EDIT mode.

## 2 Source-measure setup, performance control and special operation:

### Top Row – Source-measure setup

#### Models 2601 and 2602:

SRC	Channel A – Selects the source function (V or A) and places cursor in the source field for editing.
MEAS	Channel A – Cycles through measure functions (V, A, $\Omega$ or W).
LIMIT	Channel A – Places the cursor in the compliance limit field for editing.
MODE	Channel A – Directly chooses the measurement function (V, A, $\Omega$ or W).

#### Model 2602 only:

SRC	Channel B – Selects the source function (V or A) and places cursor in the source field.
MEAS	Channel B – Cycles through measure functions (V,A, $\Omega$ or W).
LIMIT	Channel B – Places the cursor in the compliance limit field for editing.
MODE	Channel B – Directly chooses the measurement function (V, A, $\Omega$ or W).

### Middle Row

#### Models 2601 and 2602:

DIGITS	Channel A – Changes resolution display to 4-1/2, 5-1/2, or 6-1/2 digits.
SPEED	Channel A – Sets the measurement speed by controlling the A/D converter measurement aperture.
REL	Channel A – Controls relative, which allows a baseline value to be subtracted from a reading.
FILTER	Channel A – Controls the digital filter, which can be used to reduce reading noise.

#### Model 2602 only:

DIGITS	Channel B – Changes resolution display to 4-1/2, 5-1/2, or 6-1/2 digits.
SPEED	Channel B – Sets the measurement speed by controlling the A/D converter measurement aperture.
REL	Channel B – Controls relative, which allows a baseline value to be subtracted from a reading.
FILTER	Channel B – Controls the digital filter, which can be used to reduce reading noise.



**Bottom Row**

LOAD	Loads factory or user-defined scripts for execution.
RUN	Runs last selected factory or user-defined scripts.
STORE	Stores readings, source values, and timestamp values in one of two internal buffers for later recall.
RECALL	Recalls stored readings, source values, and timestamp values from either of the two buffers.
TRIG	Triggers readings.
MENU	Accesses the main menu for saving and recalling setups, selecting remote interface, line frequency, self-tests, serial number and beeper control.
EXIT	Cancels selection, backs out of menu structure. Also used as a LOCAL key to take the unit out of remote.
ENTER	Accepts selection, moves to next choice or exits menu.

**3 Range keys:**

$\triangle$ and $\nabla$	Selects the next higher or lower source or measure range.
AUTO	Enables or disables source or measure auto range.

**4 Output control and LED status indicator:**

OUTPUT ON/OFF	Turns source output on or off.
LED indicator	Turns on when output is on.

**5 Rotary knob and CURSOR keys:**

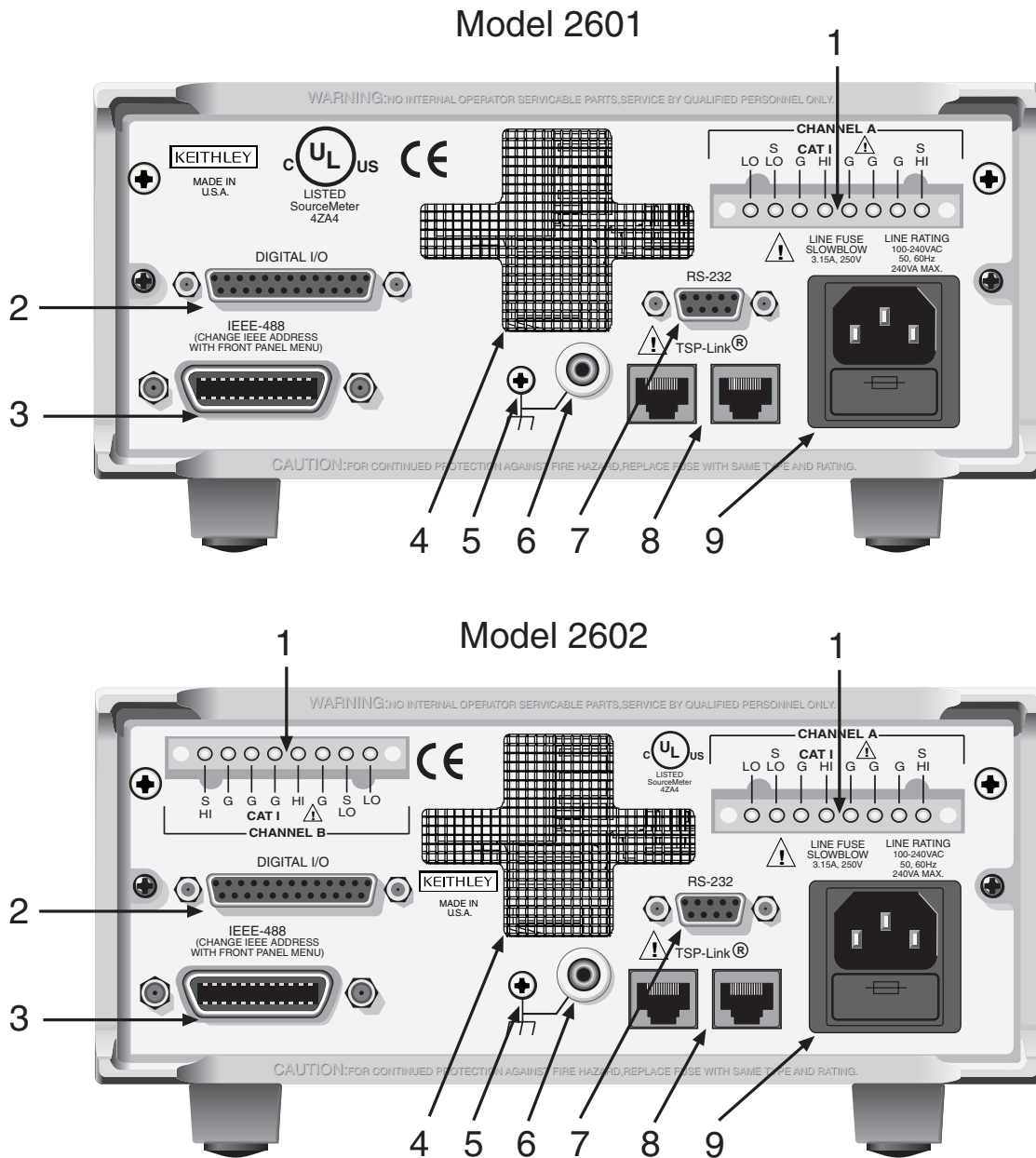
When in source edit, use CURSOR keys for cursor control and then rotate the knob to change a source or compliance value. The rotary knob can also be used to enable or disable the source edit mode.

When in a menu, use the CURSOR keys or rotary knob for menu item cursor control. When displaying a menu value, use the CURSOR keys for cursor control and rotate the knob to change the value. Pressing the knob opens a menu item, or selects a menu option or value.

**6 Display annunciators (not shown):**

EDIT	Unit is in the source editing mode.
ERR	Questionable reading or invalid cal step.
REM	Unit in remote mode.
TALK	Unit addressed to talk.
LSTN	Unit addressed to listen.
SRQ	Service request.
REL	Relative mode enabled.
FILT	Analog filter or Averaging filter is enabled.
AUTO	Auto source or measure range selected.
ARM	Unit armed and ready to run.
TRIG	External triggering selected.
* (asterisk)	Readings being stored in buffer.

Figure 1-2  
Models 2601 and 2602 rear panels



## **1 CHANNEL A and CHANNEL B (Channel B on 2602 only)**

Input/output connections for source, sense, and guard.

## **2 DIGITAL I/O**

Female DB-25 connector. 14 pins for digital input or output, one pin for Output Enable. Use a cable equipped with a male DB-25 connector (Keithley part number CA-126-1CA).

## **3 IEEE-488**

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

## **4 Cooling exhaust vent**

Exhaust vent for internal cooling fan. Keep vent free of obstructions to prevent overheating.

## **5 Chassis ground**

Ground screw for connections to chassis ground.

## **6 Low noise chassis ground**

Ground jack for connecting Output HI or LO to chassis.

## **7 RS-232**

Female DB-9 connector. For RS-232 operation, use a straight-through (not null modem) DB-9 shielded cable (Keithley Model 7009-5) for connection to the PC.

## **8 TSP-Link**

Expansion interface that allows a Model 260x and other TSP-enabled instruments to trigger and communicate with each other. Use a category 5e or higher LAN crossover cable (Keithley part number CA-180-3A).

## **9 Power module**

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

## Cooling vents

The Model 260x has side intake and rear exhaust vents. One side must be unobstructed when rack mounted to dissipate heat. NEVER place a container of liquid (e.g., water, coffee, etc.) on the top cover. If it spills, the liquid will enter the case through the vents and cause severe damage.

Excessive heat could damage the Model 260x and degrade its performance. The Model 260x must be operating in an environment where the ambient temperature does not exceed 50°C.

**CAUTION** To prevent damaging heat build-up, and ensure specified performance, adhere to the following precautions:

- The rear exhaust vent and at least one side vent must be kept free of any obstructions. Even partial blockage could impair proper cooling.
- DO NOT position any devices adjacent to the Model 260x that force air (heated or unheated) into or onto its cooling vents or surfaces. This additional airflow could compromise accuracy performance.
- When rack mounting the Model 260x, make sure there is adequate airflow around at least one side to ensure proper cooling. Adequate airflow enables air temperatures within approximately one inch of the Model 260x surfaces to remain within specified limits under all operating conditions.
- Rack mounting high power dissipation equipment adjacent to the Model 260x could cause excessive heating to occur. The specified ambient temperature must be maintained around the surfaces of the Model 260x to specified accuracies. A good measure to ensure proper cooling in rack situations with convection cooling only is to place the hottest equipment (e.g., power supply) at the top of the rack. Precision equipment, such as the Model 260x, should be placed as low as possible in the rack where temperatures are coolest. Adding space panels below the Model 260x will help ensure adequate air flow.

# Power-up

## Line power connection

Follow the procedure below to connect the Model 260x to line power and turn on the instrument. The SourceMeter operates from a line voltage of 100V to 240V at a frequency of 50Hz or 60Hz. Line voltage is automatically sensed. There are no switches to set. Make sure the operating voltage in your area is compatible.

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

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

**WARNING** The power cord supplied with the Model 260x contains a separate ground wire for use with grounded outlets. When proper connections are made, instrument chassis is connected to power line ground through the ground wire in the power cord. Failure to use a grounded outlet may result in personal injury or death due to electric shock.

3. Turn on the instrument by pressing the front panel power switch to the on (I) position.

## Line frequency

The Model 260x will operate at line frequencies of either 50Hz or 60Hz. For best measurement noise performance, the unit should be configured to match the actual line frequency used as follows:

1. Press the MENU key.
2. Select LINE-FREQ, then press ENTER or the Rotary Knob.
3. Choose 50Hz or 60Hz, then press ENTER or the Rotary Knob.
4. Press EXIT to back out of the menu structure.

Via remote, use the `localnode.linefreq` command to set the line frequency. For example, the following command sets the line frequency to 60Hz:

```
localnode.linefreq = 60
```

## Fuse replacement

A rear panel fuse drawer is located below the AC receptacle (see [Figure 1-2](#)). This fuse protects the power line input of the instrument. If the line voltage fuse needs to be replaced, refer to “[Line fuse replacement](#)” on [page 17-2](#).

## Power-up sequence

On power-up, the Model 260x performs self-tests on its ROM, NVRAM, and RAM and momentarily lights all segments and annunciators. If a failure is detected, the instrument momentarily displays an error message and the ERR annunciator turns on. (Error messages are listed in [Appendix B](#)).

**NOTE** If a problem develops while the instrument is under warranty, return it to Keithley Instruments, Inc., for repair.

Assuming no errors occur, the Model 260x will power-up as follows:

- After a few seconds with the OUTPUT indicators and display pixels on, the instrument model number, firmware revision levels, and line frequency setting are briefly displayed.
- The node and the GPIB address are displayed briefly as follows:

```
KEITHLEY MODEL 2602
```

```
NODE = 1           GPIB = 26
```

- The node and serial port parameters are displayed briefly:

```
KEITHLEY MODEL 2602
```

```
NODE = 1           SERIAL = 9600,8,N,1,NONE
```

## System identification

Serial number, firmware revision, and calibration dates can be displayed by selecting the SERIAL# item of the main menu (press MENU > GENERAL > SYSTEM-INFO). Select FIRMWARE, SERIAL#, or CAL as desired.

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

## Beeper

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

- A short beep, emulating a keyclick, is issued when a front panel key is pressed.
- A short beep is also issued when the rotary knob is turned or pressed.
- A longer beep is issued when the source output is turned on.

To control the beeper from the front panel, select MENU > GENERAL > BEEPER, then ENABLE or DISABLE the beeper as desired.

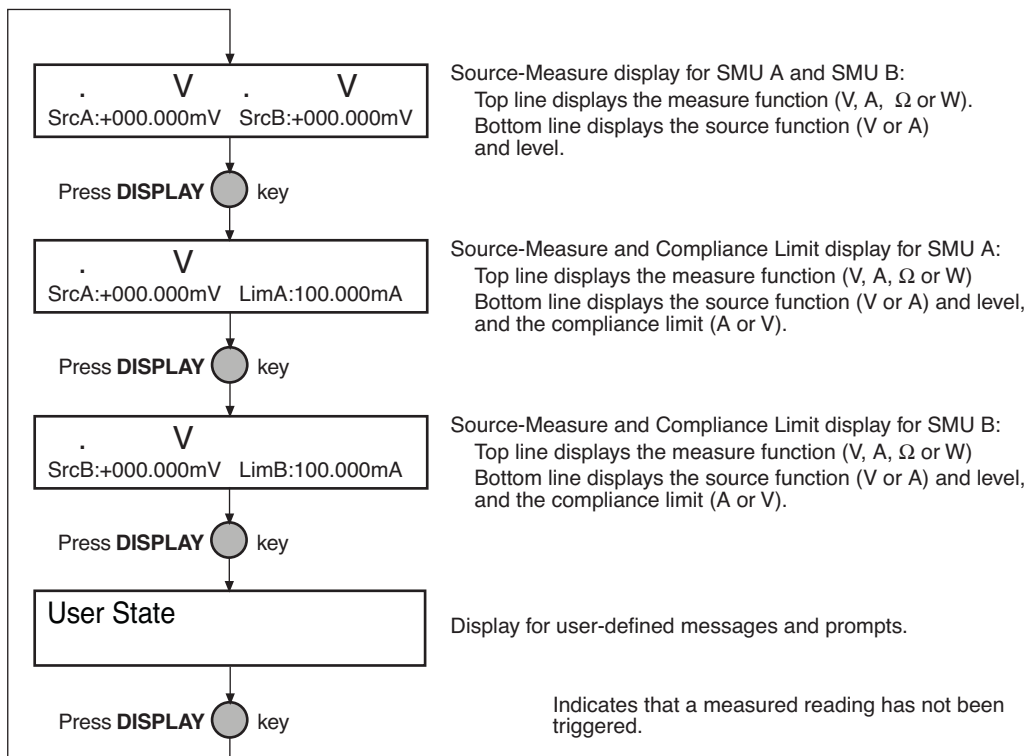
Via remote, use the `beeper.enable` command to control the beeper. For example, the following enables the beeper:

```
beeper.enable = 1
```

## Display modes

Use the DISPLAY key to cycle through the various display modes shown in [Figure 1-3](#). For the Model 2602 only, pressing DISPLAY cycles between the dual-channel display mode and single-channel modes for CHANNEL A (SMU A) and CHANNEL B (SMU B). The Model 2601 has only one channel (SMU A). The User State display messages are defined with specific display commands (see [Section 14](#) for more information on display messaging).

Figure 1-3  
Display modes



# Editing controls

## Source and compliance editing

When the Model 260x is in the edit mode (EDIT annunciator on), the editing controls are used to set source and compliance values. Note that source auto ranging will turn off when editing the source value.

### Editing source values

1. Press the SRC key to edit the source. Note that the cursor will flash in the source value field of the display.
2. Place the blinking cursor on the desired digit by using either the CURSOR keys, or by turning the Rotary Knob.
3. Press in on the Rotary Knob to enter the edit mode, as indicated by the EDIT annunciator.
4. Modify the source value in either one of two ways:
  - Rotate the Rotary Knob to adjust the digit. Note that the digit will automatically overflow or underflow to the next digit when the maximum is reached.
  - Directly enter the desired value using the numeric keys (0-9, +/-, 0000). Note that +/- toggles the polarity, while 0000 sets the value to 0.
5. Once the desired value is displayed, either press ENTER, or press in on the Rotary Knob to complete editing. Note that the EDIT annunciator goes off.
6. To cancel source editing without making a change, press the EXIT key instead of using ENTER or press in on the Rotary Knob.

### Editing compliance values

1. For the Model 2601 or the Model 2602 single-channel display mode, press the LIMIT key to directly edit the compliance value.
2. For the Model 2602 dual-channel display mode, press LIMIT or CONFIG then LIMIT to edit the compliance limit. Next, select either VOLTAGE or CURRENT either by using the CURSOR keys or by rotating the Rotary Knob. Press ENTER or the Rotary Knob.
3. Place the blinking cursor on the desired digit by using either the CURSOR keys, or by turning the Rotary Knob.
4. Press in on the Rotary Knob to enter the edit mode, as indicated by the EDIT annunciator.



5. Modify the compliance limit value in either one of two ways:
  - Rotate the Rotary Knob to adjust the digit. Note that the digit will automatically overflow or underflow to the next digit when the minimum or maximum is reached.
  - Directly enter the desired value using the numeric keys (0-9).
6. Once the desired value is displayed, either press ENTER, or press in on the Rotary Knob to complete editing. Note that the EDIT annunciator goes off.
7. To cancel editing without making a change, press the EXIT key as needed to back out of the menu structure instead of using ENTER or pressing in on the Rotary Knob.

## Menu navigation

When the Model 260x is not in the edit mode (EDIT annunciator off), the editing controls are used to navigate the Main and Configuration menus (see [page 1-18](#)) to make selections and/or set values. After entering a menu structure, use the editing keys as follows:

### Selecting menu items

1. Use the CURSOR keys or rotate the Rotary Knob to place the blinking cursor on a menu item to be opened or selected.
2. Press the ENTER key or the Rotary Knob to select an item or open a sub menu.
3. Use the EXIT key to cancel a change or back out of the menu structure.

### Setting a value

There are two ways to adjust a value: value adjust or numeric entry. Both methods use the following editing techniques:

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

### Value adjust method

1. Use the CURSOR keys or rotate the Rotary Knob to place the blinking cursor on the digit to be edited.
2. Press the Rotary Knob to enter the edit mode (EDIT annunciator on).
3. Rotate the Rotary Knob to set the value as needed. Adjusting past the maximum or minimum digit value will automatically move the cursor to the next higher or lower digit for editing.
4. Press ENTER to select the value. Press EXIT to cancel the change.

### Numeric entry method

1. Use the CURSOR keys or rotate the Rotary Knob to place the blinking cursor on the most significant digit to be edited.
2. Key in a digit by pressing a number entry key (0 to 9). The cursor will move to the next digit on the right.
3. Repeat step 2 as needed to set the desired value.
4. Press ENTER to select the value. Pressing EXIT will cancel the change.

## Menus

Many aspects of operation are configured through menus. There are two types of menus: the Main menu (accessed with the MENU key), and Configuration menu (accessed with the CONFIG key). Each of these is covered below. See [“Menu navigation” on page 1-17](#) for details on using menus.

### Main menu

The Main menu is summarized in [Table 1-1 on page 1-19](#) along with the reference for each main selection. To access menu items, press the MENU key, then make your selection.

### Configuration menus

The configuration menus are summarized in [Table 1-2 on page 1-20](#) along with the reference for each main selection. There are two ways to make selections:

- Press CONFIG, then navigate to the desired submenu.
- Press CONFIG, then press the associated key. For example, pressing CONFIG followed by REL takes you directly to the Relative menu.

Table 1-1  
Main menu

Menu selection	Description	Reference
SAVESETUP SAVE RECALL POWERON RESET	Save/recall user and factory setups. Save up to five user setups. Recall user setups. Set power-on default configuration. Reset Model 260x to factory defaults.	<a href="#">Page 1-22</a>
COMMUNICATION INTERFACE_CFG INTERFACE_SEL TSPLINK_CFG	Select interface, control settings. Configure GPIB and RS-232 interfaces. Select GPIB or RS-232 interface, or use AUTO. Select NODE and RESET TSP-Link.	<a href="#">Section 11</a> <a href="#">Section 11</a> <a href="#">Section 9</a>
TEST KEYS DISPLAY_PATTERNS	Perform display tests. Test keys. Perform display patterns tests.	<a href="#">Section 17</a>
LINE-FREQ	Set line frequency to 50Hz or 60Hz.	<a href="#">Page 1-13</a>
GENERAL DIGOUT BEEPER SYSTEM-INFO	Control digital I/O, beeper, system information. Set digital I/O bits, control write protect. ENABLE or DISABLE beeper. Display FIRMWARE, SERIAL#, CAL dates.	<a href="#">Section 10</a> <a href="#">Page 1-14</a> <a href="#">Page 1-14</a>

Table 1-2  
**Configuration menus**

Menu selection <sup>1</sup>	Shortcut <sup>2</sup>	Description	Reference
CHANNEL-A SRC MEAS LIMIT SPEED REL FILT OUTPUT	SRC MEAS LIMIT SPEED REL FILTER OUTPUT	Configure channel A: V-source sense, low range; I-source low range. V and I -Measure sense, low range; auto zero. V-source and I-source compliance limits. Measurement speed (NPLC). Set relative values. Control digital filter. Set off-state, control digital I/O.	<a href="#">Section 4</a> <a href="#">Section 4</a> <a href="#">Section 4</a> <a href="#">Section 6</a> <a href="#">Section 6</a> <a href="#">Section 6</a> <a href="#">Section 3,</a> <a href="#">Section 10</a>
CHANNEL-B SRC MEAS LIMIT SPEED REL FILT OUTPUT	SRC MEAS LIMIT SPEED REL FILTER OUTPUT	Configure channel B: V-source sense, low range; I-source low range. V and I -Measure sense, low range; auto zero. V-source and I-source compliance limits. Measurement speed (NPLC). Set relative values. Control digital filter. Set off-state, control digital I/O.	<a href="#">Section 4</a> <a href="#">Section 4</a> <a href="#">Section 4</a> <a href="#">Section 6</a> <a href="#">Section 6</a> <a href="#">Section 6</a> <a href="#">Section 3,</a> <a href="#">Section 10</a>
COMMON TRIG STORE	TRIG STORE	Configure common functions: Set trigger in, count, interval, and delay. Set buffer count and destination.	<a href="#">Section 9</a> <a href="#">Section 7</a>

1. Channel B available only on Model 2602.

2. Press CONFIG followed by indicated key to access directly.

# Interface selection

The following summarizes basic interface selection for the Model 260x. Details on the interfaces, including configuration, are provided in [Section 11](#). Use the editing controls for “[Menu navigation](#)” on [page 1-17](#) to select and configure the interface.

## GPIB interface

1. Press MENU to open up the Main menu.
2. Select COMMUNICATION, then press ENTER.
3. Select INTERFACE\_SEL, then press ENTER.
4. Choose GPIB, then press ENTER.
5. Select INTERFACE\_CFG, then press ENTER.
6. Choose GPIB, then press ENTER.
7. Set the GPIB address (0 to 30), and press ENTER.
8. Press EXIT to back out of the menu structure.

## RS-232 interface

1. Press MENU to open up the Main menu.
2. Select COMMUNICATION, then press ENTER.
3. Select INTERFACE\_SEL, then press ENTER.
4. Choose RS-232, then press ENTER.
5. Select INTERFACE\_CFG, then press ENTER.
6. Choose RS-232, then press ENTER.
7. Configure the RS-232 interface as follows:
  - Set the BAUD rate: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 or 115200.
  - Set BITS: 7 or 8.
  - Set PARITY: NONE, ODD, or EVEN.
  - Set the FLOW-CTRL: NONE or HARDWARE.
8. Press EXIT to back out of the menu structure.

# Error and status messages

Error and status messages are displayed momentarily. During operation and programming, you will encounter a number of front panel messages. Typical messages are either status or error, as listed in [Appendix B](#).

Messages, both status and error, are held in queues. For information on retrieving error messages from queues, see [Appendix D](#).

# Default settings

The Model 260x can be restored to one of six setup configurations: five user-saved setups, and the original factory defaults. As shipped from the factory, the Model 260x powers up to original default settings, which are also saved in the five user setup locations. Original default settings are listed in [Table 1-3 on page 1-24](#). The instrument will power up to whichever default setup was saved as the power-on setup.

## Front panel setups

### To save a user setup

1. Configure the Model 260x for the desired operating modes to be saved.
2. Press the MENU key to access the Main menu.
3. Select SAVE-SETUP, then press ENTER.
4. Select the SAVE menu item, then press ENTER.
5. Select the desired user memory location (1 through 5), and press ENTER.

### To restore a setup

1. Press the MENU key to access the Main menu.
2. Select SAVE-SETUP, then press ENTER.
3. Select the RECALL menu item, then press ENTER.
4. Select the desired user memory location (1 through 5), or select FACTORY to restore factory defaults, and press ENTER.

### To select power-on setup

1. Press the MENU key to access the Main menu.
2. Select SAVE-SETUP, then press ENTER.
3. Select the POWERON menu item, then press ENTER.
4. Choose FACTORY to power-on to original defaults, or choose USER-SETUP-NUMBER to power-on to one of the five user setups (1-5). Press ENTER.

## Remote operation setups

### Saving and restoring user setups

The `setup.save` and `setup.recall` commands are used to save and recall user setups:

```
setup.save (n)      --Save present setup in memory.  
setup.recall (n)   --Recall saved user setup from memory.  
                  --n = 1, 2, 3, 4 or 5
```

### Restoring default setups

The reset commands return the Model 260x to the original factory defaults:

```
reset ()           --Restore all factory defaults.  
smua.reset ()     --Restore Channel A defaults.  
smub.reset ()     --Restore 2602 Channel B defaults.
```

### Selecting power-on setup

The `setup.poweron` command is used to select which setup to return to on power-up:

```
setup.poweron = n  --Select power-on setup.  
                  --n = 0 (*RST defaults)  
                  --n = 1 to 5 (user setups 1-5)
```

Table 1-3  
**Default settings**

Setting*	Default
A/D Controls: Auto-zero Line frequency	Auto No effect
Beeper	On
Data Store	No effect
Digital output: Output value Write protect	No effect No effect
Digits	5-1/2
Display mode (Model 2602)	Dual-channel
Filter: Averaging type Count	Off Repeat 1
GPIB address	No effect
Limit value: Current limit Voltage limit	1A 40V
Measure: Function I-meter range V-meter range	Voltage 100mA 100mV
Output Off state	Off Normal
Rel Current value Voltage value Ohms value Watts value	Off 0.0pA 0mV 0mΩ 0mW
RS-232	No effect
Sense mode	2-wire

\* Where applicable, settings apply to both Model 2602 channels.



Table 1-3 (cont.)  
**Default settings**

Setting*	Default
Source:	
Function	Voltage
Current value	0A
Voltage value	0V
Current range	100nA
Voltage range	100mV
Speed	Normal (1 PLC)
Triggering:	
Trigger-in source	Immediate
Count	Finite
Interval	0s
Delay	0s
TSP-Link node	1

\* Where applicable, settings apply to both Model 2602 channels.

## Remote programming

Programming information is integrated with front panel operation throughout this manual. Basic command information is listed in tables. For specific information on programming, refer to [Sections 2, 12, and 13](#) of this manual.

### Requesting readings

You can request readings by including the appropriate measurement command as the argument for the `print` command. For example, the following will request a channel A current reading:

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

### Requesting command settings

In a similar manner, settings for commands can be requested by including the command as the argument for the `print` command. For example, the following command will request the voltage source setting for channel A:

```
print(smua.source.levelv)
```

# TSP Programming Fundamentals

---

## Section 2 topics

**Introduction**, page 2-2

**Programming overview**, page 2-6

**Installing the TSP software**, page 2-10

**System connections**, page 2-10

**Using Test Script Builder**, page 2-12

Starting Test Script Builder, page 2-14

Opening communications, page 2-15

Creating and modifying a script, page 2-17

Script launch configuration, page 2-22

Launching a script, page 2-25

Running a TSP file, page 2-26

Retrieving scripts from the Model 260x, page 2-26

Instrument Console, page 2-28

File management tasks, page 2-34

**Sending commands and statements**, page 2-38

Source-measure voltage and current, page 2-38

Read and write to Digital I/O port, page 2-38

Display user-defined messages, page 2-38

**User scripts**, page 2-39

Script examples, page 2-39

Creating a user script, page 2-42

Saving a user script, page 2-43

Running a user script, page 2-43

Modifying a user script, page 2-46

Script management, page 2-46

**Factory scripts**, page 2-48

Running a factory script, page 2-48

Modifying a factory script, page 2-48

Differences: remote vs. local state, page 2-49

Memory considerations, page 2-51

**Test Script Language (TSL) reference**, page 2-52

Reserved words, page 2-52

Variables and types, page 2-52

Operators, page 2-53

Tables/arrays, page 2-53

Functions, page 2-54

Precedence, page 2-55

Logical operators, page 2-55

Concatenation, page 2-56

Branching, page 2-57

Loop control, page 2-58

Standard libraries, page 2-59

# Introduction

Conventional instrumentation responds to command messages sent to the instrument. Each command message contains one or more commands. The instrument executes these commands in order.

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.

Keithley's Test Script Processor based instruments can operate as conventional instruments by responding to a sequence of command messages sent by a controller. They are also capable of much more.

## Test Script Processor (TSP)

**Scripting** To orchestrate a sequence of actions.  
**Scripting Language** A programming language used for scripting.

The Test Script Processor (TSP) is a scripting engine that runs inside the instrument. It is capable of running code written in a scripting language called Lua ([www.lua.org](http://www.lua.org)). We will refer to Lua as the Test Script Language (TSL). The TSP runs portions of TSL code formally known as chunks. Most messages sent to the instrument are directly executed by the TSP as TSL chunks. The simplest messages sent to the instrument would be individual instrument control commands. Even though these messages are executed as TSP chunks, using them is no different than using a conventional instrument. The user sends a command message and the instrument executes that command. When sending individual command messages, it is irrelevant that the TSP is executing the message as a chunk.

Instrument control commands are implemented as a library within the TSP. The total command set for a TSP-enabled instrument is the Instrument Control Library (ICL) for that instrument. Each TSP-enabled instrument will have its own ICL. Although each TSP-enabled instrument runs the same TSL, different instruments respond to different commands and the ICL for each instrument may be different.

ICL commands are very similar to the commands sent to a conventional instrument but ICL commands look like function calls or assignment statements. For example the command to set the output voltage level to one volt on channel A is `smua.source.levelv = 1`. Similarly, the command to turn the Channel A output on is `smua.source.output = smua.OUTPUT_ON`. These commands, when sent individually as separate messages, are each a TSL chunk.

Commands do not need to be sent as separate messages. The two commands from above can be combined into one message, and thereby one chunk, by con-

concatenating the two commands together with a space separating them. The resulting chunk would be as follows:

```
smua.source.levelv = 1 smua.source.output = smua.OUTPUT_ON
```

## Run-time environment

A feature of all scripting environments is the run-time environment. In the TSP, the runtime environment is simply a collection of global variables. A global variable can be used to remember a value as long as the unit is powered on and the variable is not assigned a new value. The command `x = smua.measure.v()` instructs the instrument to measure voltage and store the result in a global variable named “x.”

A global variable can be removed from the environment by assigning it the nil value. For example, the command “`x = nil`” will remove the global variable `x` from the run-time environment. When the unit is turned off, the entire run-time environment will be lost. Note that SMU non-volatile reading buffers are not lost.

## Queries

TSP-enabled instruments do not have inherent query commands. Like any other scripting environment the print command and other related print commands are used to generate output. The print command will create one response message.

An example of generating an output message is the following chunk (two commands) that takes a measurement and returns its value:

```
x = smua.measure.v() print(x)
```

Note that the measurement value is stored in the global variable `x` between the two commands.

## Scripts

When taking advantage of the TSP to perform more complicated sequences of commands, especially sequences utilizing advance scripting features such as looping and branching, sending the entire sequence in one message is very cumbersome. Two special messages can be used to collect a sequence of command message together into one chunk.

The `loadscript` message will instruct the TSP-enabled instrument to begin collecting all subsequent messages rather than executing them immediately. After sending the sequence of command messages, the `endscript` message is used to instruct the TSP-enabled instrument to compile the test sequence and make it available to run in a subsequent message. This chunk is called the “active script.”

The active script can be run at any time by sending the command `script.run()`. The active script can be run many times without needing to re-send it. Each time the `script.run()` command is given, the active script will be executed.

Sending a new script using the `loadscript` and `endscript` messages will instruct the TSP-enabled instrument to replace the active script with the new script. While creating and using scripts this way is a very powerful feature of TSP-enabled instruments, only being able to access one script at a time in this way would be very limited. The “[Named scripts](#)” topic describes how to use named scripts to store many scripts in the instrument at one time.

## Named scripts

The `loadscript` message can also be used to create named scripts. When the `loadscript` message is used to create a named script the active script is not replaced with the named script. Instead, a global variable in the run-time environment is created to store the script. Because the script is stored in a global variable, the name of the script must be a legal TSL variable name.

The name of the script is specified in the `loadscript` message by appending it to the message and separating it from the `loadscript` keyword with a space character. The message `loadscript MyScript` will instruct the TSP-enabled instrument to begin gathering command messages that will be used to create a script named `MyScript`. After sending the command messages the `endscript` message is still used to indicate the end of the script. Upon receipt of the `endscript` message the instrument will compile the script. If there are no errors, the script will be made available as the global variable `MyScript` because that is the name we used in the `loadscript MyScript` message. After a named script has been successfully sent to the instrument, it can be run at any time by sending the `MyScript()` message. If the name given to the script were different, that name would be used instead.

If a new script is sent with the same name, it will replace the old one. Sending new scripts with different names will not remove any previously sent scripts. By using named scripts, any number of scripts can be made available simultaneously within the limits of the memory available to the run-time environment.

Named scripts are stored as global variables in the run-time environment. Like all other global variables, when the unit is powered off, they are lost. There is non-volatile storage on the instrument that can be used to store downloaded scripts across power cycles. See “[Saving a user script](#)” on [page 2-43](#) for more information.

## Functions

As previously explained, named scripts behave just like TSL functions. Executing a script is just like executing a function with the same name as the script. Scripts, like functions, may return values. Unlike functions, scripts may not take any parameters. In order to pass parameters to a chunk, you must make a TSL function.

Functions are created with a message in one of the following forms:

```
MyFunction = function (parameter1, parameter2) function body end  
or  
function MyFunction(parameter1, parameter2) function body end
```

Where `function body` is a TSP chunk that will be executed when the function is called. The above function can be executed by sending the following message:

```
MyFunction(value for parameter1, value for parameter2)
```

Where `value for parameterN` are the values to be passed to the function call for the given parameters. Note that when a function is defined, it is just another global variable in the run-time environment. Just like all global variables, functions will persist until it is removed from the run-time environment, overwritten, or the unit is turned off.

## Scripts that create functions

It is inconvenient in most cases to define a function in one message. The solution is to create a script that defines a function. The script will be like any other script. It will not cause any action to be performed on the instrument until it is executed. Remember that creating a function is just creating a global variable that is a function. That global variable will not exist until the chunk that creates it is executed. In this case the chunk that creates it is a script. Therefore the function will not exist until the script that creates it is executed. This is often confusing to first time users.

**Example:** Create the function `MyFunction` with a script named `MakeMyFunction`. The sequence of messages to do this is shown as follows:

```
loadscript MakeMyFunction  
    MyFunction = function (who)  
        print("Hello " .. who)  
    end  
endscript
```

After this sequence of messages is sent, the `MakeMyFunction` script exists on the instrument in a global variable named `MakeMyFunction`. The `MyFunction` function however does not yet exist because we have not executed the `MakeMyFunction` script. Let us now send the message `MakeMyFunction()`. That message instructs the instrument to run the `MakeMyFunction` script which then creates the `MyFunction` global variable that happens to be a function.

If we now send the message `MyFunction("world")` the instrument will execute the `MyFunction` function which causes the instrument to generate a response message with the text "Hello world" in it.

# Programming overview

## What is a chunk?

A chunk is a single programming statement, or a sequence of statements that are executed sequentially. There are non-scripted chunks and scripted chunks.

**Single statement chunk** – The following programming statement is a chunk:

```
print ("This is a chunk")
```

When the above chunk is executed, it returns the following string:

```
This is a chunk
```

**Multiple statement chunk** – A chunk can also contain multiple statements. Each statement in the line of code is to be separated by whitespace. The following chunk contains two statements:

```
print ("This is a chunk") print ("that has two statements")
```

When the above chunk is executed, the two statements are executed sequentially and the following strings are returned:

```
This is a chunk  
that has two statements
```

**Multiple chunks** – The following two lines of code are two chunks. The first chunk sets the source level of SMU A to 1V and the second chunk turns the output on.

```
smua.source.levelv = 1  
smua.source.output = smua.OUTPUT_ON
```

**Scripted chunk** – In a script environment, the chunk is the entire listing of test programming code. If the two statements in the above example were created as a script, then those two lines of code would be considered one chunk. See [“What is a script?”](#).

## What is a script?

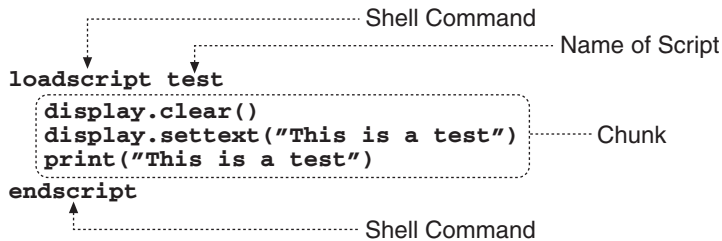
The Model 260x utilizes a Test Script Processor (TSP) to process and run programs called scripts. A script is a collection of instrument control commands and programming statements. [Figure 2-1](#) shows an example of how to create (and load) a script named “test”. When this script is run, the message “This is a test” will be displayed on the Model 260x and sent to the PC.

As shown, a script is made up of a chunk of programming code that is framed by shell commands. The first shell command in [Figure 2-1](#) loads the script named “test”. The last shell command marks the end of the script.

The chunk in [Figure 2-1](#) consists of three lines of code. When the chunk is executed, the test messages will be sent and displayed. The following command executes the chunk: `test()`

**NOTE** It is common practice to say that a script is run. In actuality, it is the chunk in the script that is being run (executed).

Figure 2-1

**Script example**

A script is loaded into the Model 260x SourceMeter where it can be run. Running a script at the SourceMeter is faster than running a test program from the PC. The piece-meal transmission process from PC to SourceMeter is eliminated by the use of a script.

Program statements control script execution and provide facilities such as variables, functions, branching, and loop control. Because scripts are programs, they are written using a programming language. This language is called the Test Script Language or TSL. TSL is derived from the Lua scripting language. For details, see [“Test Script Language \(TSL\) reference”](#) on page 2-52.

There are two types of scripts: Factory scripts and user scripts. A factory script was created by Keithley at the factory and stored in non-volatile memory of the Model 260x SourceMeter. A user script is created using your own program or the Test Script Builder Integrated Development Environment (IDE), which is a supplied software tool (see [“Using Test Script Builder”](#) on page 2-12). The user script is loaded into the Model 260x SourceMeter and can be saved in non-volatile memory.

## Run-time environment

The run-time environment is a collection of global variables (scripts) the user has created. After scripts are placed into the run-time environment, they are then ready to be run and/or managed. Scripts are placed in the run-time environment as follows:

- Scripts saved in [“Non-volatile memory”](#) of the Model 260x are automatically recalled into the run-time environment when the instrument is turned on.
- Named scripts created and loaded by the user are also placed in the run-time environment.
- An unnamed script created and loaded by the user is also placed in the run-time environment. Keep in mind that only one unnamed script, referred to as the active-script, can be in the run-time environment. If another



unnamed script is created and loaded, it will replace the old unnamed script in the run-time environment.

## Non-volatile memory

After a new or modified user script is loaded into the Model 260x, it resides in the run-time environment and will be lost when the unit is turned off. To save a script after power-down, the script must be saved in the non-volatile memory. When the Model 260x is turned back on, all saved scripts will load into the “[Run-time environment](#)”.

Do not confuse the run-time environment with the non-volatile memory of the Model 260x. Making changes to a script in the run-time environment does not affect the stored version of that script. After making changes, saving the script will overwrite the old version of the script in non-volatile memory.

## TSP programming levels

Instrument Control Library (ICL) commands and TSL programming statements are used to program and control the SourceMeters in the test system. There are three levels of programming:

- [Sending commands and statements \(page 2-38\)](#) – Non-scripted chunks are executed one line at a time by the PC.
- [User scripts \(page 2-39\)](#) – A program script is created and loaded into the Model 260x SourceMeter where it is then run.
- [Interactive script \(page 2-40\)](#) – This type of script interacts with the operator. It will provide user-defined messages on the SourceMeter display to prompt the operator to enter parameters from the front panel.

## Programming model for scripts

The fundamental programming model for scripts is shown in [Figure 2-1](#). Factory scripts (created by Keithley at the factory) are permanently stored in non-volatile memory of the Model 260x. User created scripts can also be stored in non-volatile memory.

When the Model 260x is turned on, all user scripts and factory script functions are recalled into the run-time environment from non-volatile memory. If any user scripts have been programmed to run automatically, they will run after all the scripts are loaded. Any script in the run-time environment can be run from the Test Script Builder or the users own program. Test data (e.g., readings) is returned from the Model 260x to the PC.

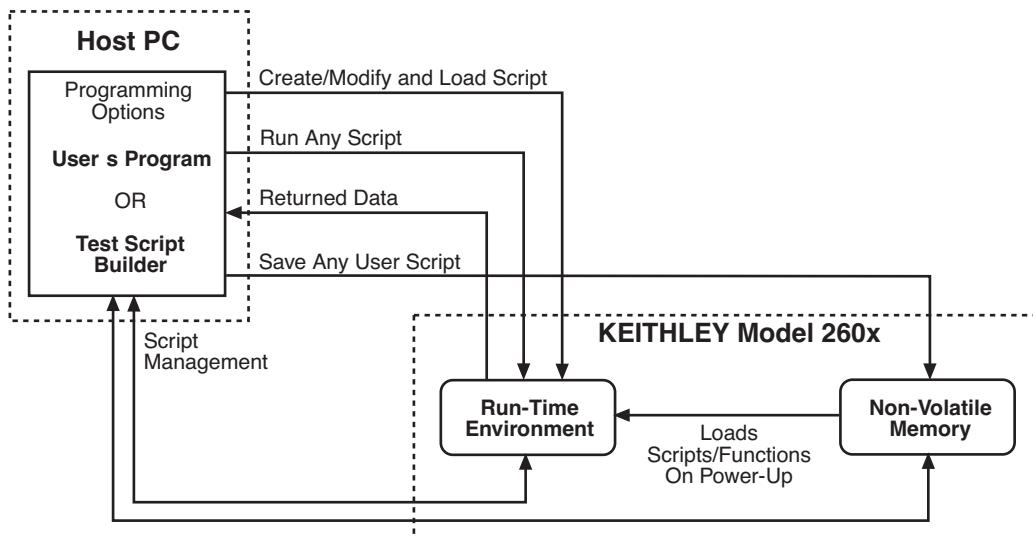
A user script can be created using the Test Script Builder or the users own program. Once the user script is loaded into the run-time environment, it is ready to be run. A user script can be saved in the non-volatile memory of the Model 260x. If it is not saved, the Model 260x will lose the script when it is turned off.

Script management includes commands for the following operations:

- Retrieve scripts from non-volatile memory so they can be modified.
- Delete user scripts from non-volatile memory.
- Restore scripts in the run-time environment from non-volatile memory.

Figure 2-2

### Programming model for scripts



## Installing the TSP software

To install the TSP software, close all programs, place the CD into your CD-ROM drive and follow the on-screen instructions. If the software install does not start automatically, click Start > select Run > click Browse and locate the CD in your CD-ROM > double-click the `setup.exe` file.

## System connections

Up to 64 Model 260x instruments can be used in a test system. The host interface for the test system can be the GPIB or the RS-232. For the GPIB, an IEEE-488 cable is used to connect the PC to one of the Model 260x instruments. For the RS-232, a straight-through RS-232 cable terminated with DB-9 connectors is used to connect the PC to the one of the Model 260x instruments.

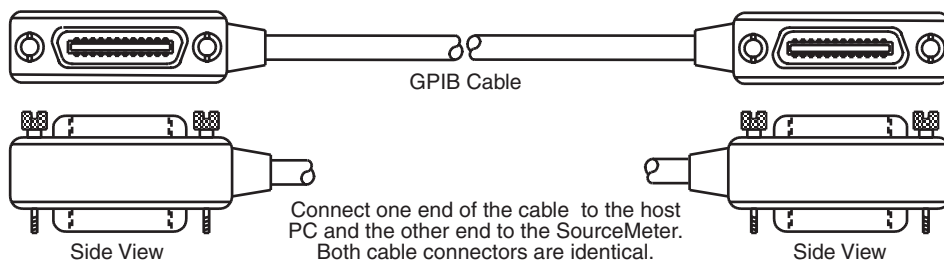
Keep in mind that the GPIB or RS-232 cable is only required to be connected to one of the Model 260x instruments. Communication to the other Model 260x instruments can be accomplished via the TSP-Link (see [Section 9](#)).

### GPIB

#### GPIB connections

To connect the Model 260x to the GPIB bus, use a cable equipped with standard IEEE-488 connectors as shown in [Figure 2-3](#). The IEEE-488 connector is located on the rear panel of the SourceMeter. When connecting the cable, make sure to tighten the captive screws.

Figure 2-3  
GPIB cable



**NOTE** To minimize interference caused by electromagnetic radiation, use only shielded GPIB cables. Available shielded cables from Keithley are the Model 7006 and Model 7007.

The GPIB cable connectors are stackable. For additional non-Model 260x GPIB instruments in the test system, daisy-chain a GPIB cable from one instrument to another.

## GPIB address

At the factory, the GPIB is set for address value 26. The address value can be set to any address value between 0 and 30. However, the address cannot conflict with the address assigned to other instruments in the system.

The GPIB address can be changed from the communications menu. To access the menu, press the MENU key, select COMMUNICATIONS and then select GPIB.

The GPIB address is saved in non-volatile memory. The address value will not change when power is cycled, or a reset command (`reset` or `*RST`) is sent.

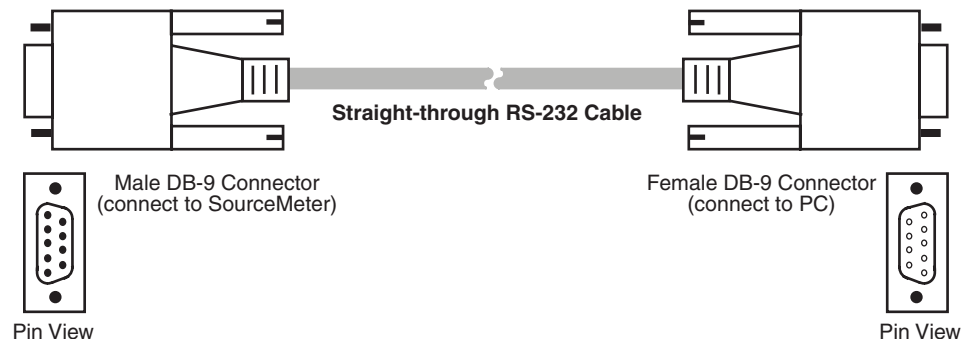
## RS-232

### RS-232 connections

To connect the Model 260x to the RS-232 interface, use a straight-through RS-232 cable (see [Figure 2-4](#)). [Figure 1-3](#) shows the location of the RS-232 connector on the SourceMeter. When connecting the cable, make sure to tighten the captive screws.

Figure 2-4

### RS-232 cable (straight-through)



### RS-232 settings

At the factory, the RS-232 is configured as follows:

- Baud rate: 9600
- Data bits: 8
- Parity: None
- Flow Control: None

The RS-232 settings can be changed from the communications menu. To access the menu, press the MENU key, select COMMUNICATIONS and then select RS-232.

The RS-232 settings are stored in non-volatile memory. The settings will not change when power is cycled, or a reset command (`reset` or `*RST`) is sent.

## Using Test Script Builder

Test Script Builder is a supplied software tool that can be used to perform the following operations:

- Send ICL commands and TSL statements
- Receive responses (data) to commands and scripts
- Run factory scripts
- Create and run user scripts

[Figure 2-5](#) shows an example of the Test Script Builder. As shown, the Workspace is divided into three window panes:

### Project Navigator

The window pane on the left side of the Workspace is where the Project Navigator resides. The navigator consists of created project folders and the script files (.tsp) created for each project. Each project folder can have one or more script files.

The navigator shown in [Figure 2-5](#) has two projects; one named “BeeperTest” and one named “SourceMeasure”. As shown, the “BeeperTest” project has one script file, and the “SourceMeasure” project has three script files.

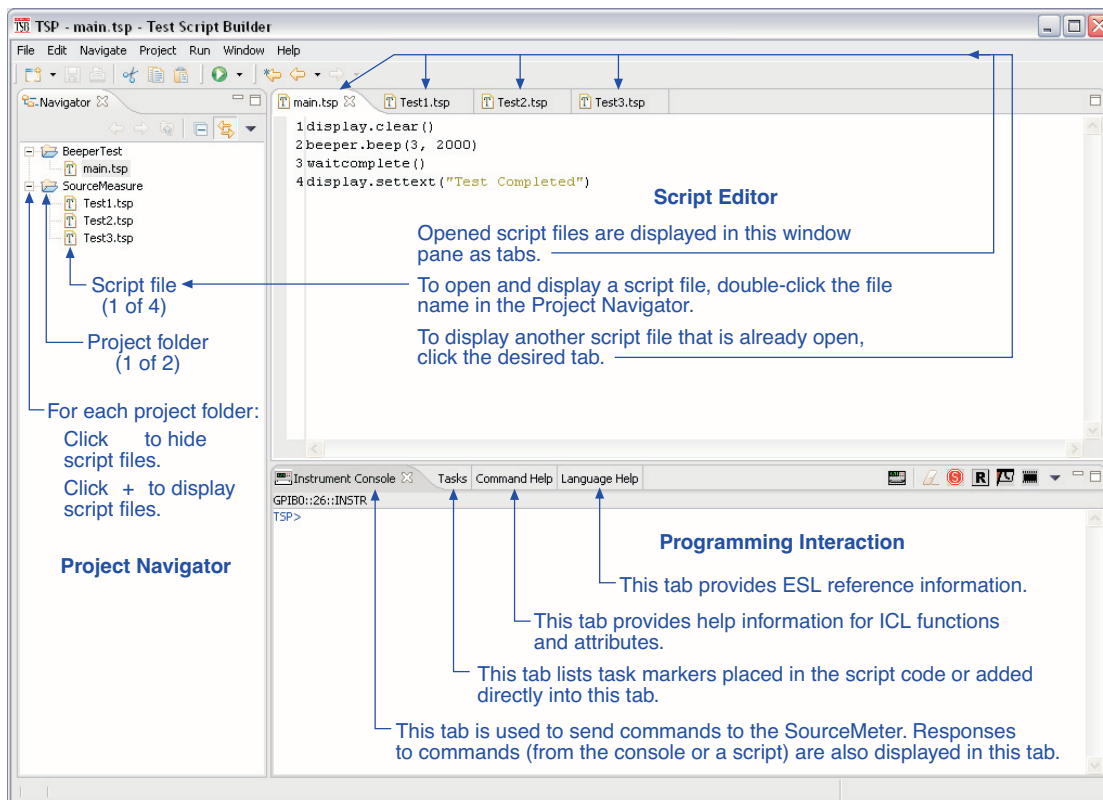
### Script Editor

The script chunk is written and/or modified in the Script Editor. Notice that there is a tab available for each opened script file. A script project is then downloaded to the SourceMeter where it can be run.

### Programming Interaction

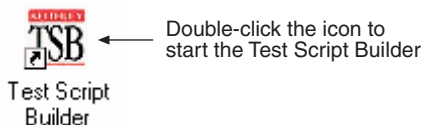
Up to seven tabs can be displayed in the lower window pane of the Workspace to provide programming interaction between the Test Script Builder and the SourceMeter. The Instrument Console (shown open in [Figure 2-5](#)) is used to send commands to the connected SourceMeter. Retrieved data (e.g., readings) from commands and scripts appear in the Instrument Console. See [“Programming interaction tabs”](#) on [page 2-30](#) for details on using the other tabs.

Figure 2-5  
**Test Script Builder (example)**



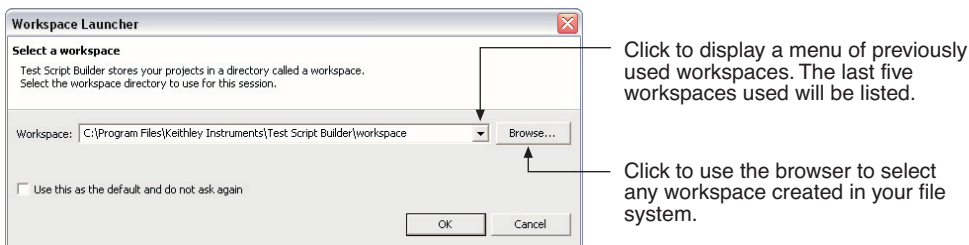
## Starting Test Script Builder

Make sure the SourceMeter is properly connected to the PC (see “[System connections](#)” on [page 2-10](#)) and it is turned on. On the PC desktop, double-click the Test Script Builder icon to begin:



**NOTE** The Test Script Builder can also be started from the Windows Start button on the task bar. For a default installation, follow this menu path to start the Test Script Builder:  
**Start > Programs > Keithley Instruments > Test Script Builder**

**Workspace Launcher** – During the initial start-up of TSB, the Workspace Launcher window will be displayed as shown below. This window will indicate the directory path for the workspace. This is where projects and script files will be stored. If you do not wish to see this window on subsequent power-ups, select Use this as the default and do not ask again. Click OK to continue start-up.



Note: See “[Creating a new workspace](#)” on [page 2-34](#) to create additional workspaces.

**Communications** – When Test Script Builder opens, communications to the SourceMeter will be closed. With communications closed, commands cannot be sent to the SourceMeter. A script can be written using the Test Script Builder, but it cannot be run. Communications with the SourceMeter are established by “[Opening communications](#)”.

## Opening communications

In order to activate communications between Test Script Builder and the SourceMeter, an instrument must be opened. The tool bar on the Instrument Console tab is used to open or close communications.

[Figure 2-6](#) illustrates how to open and close communications. The following details supplement the information in the drawing:

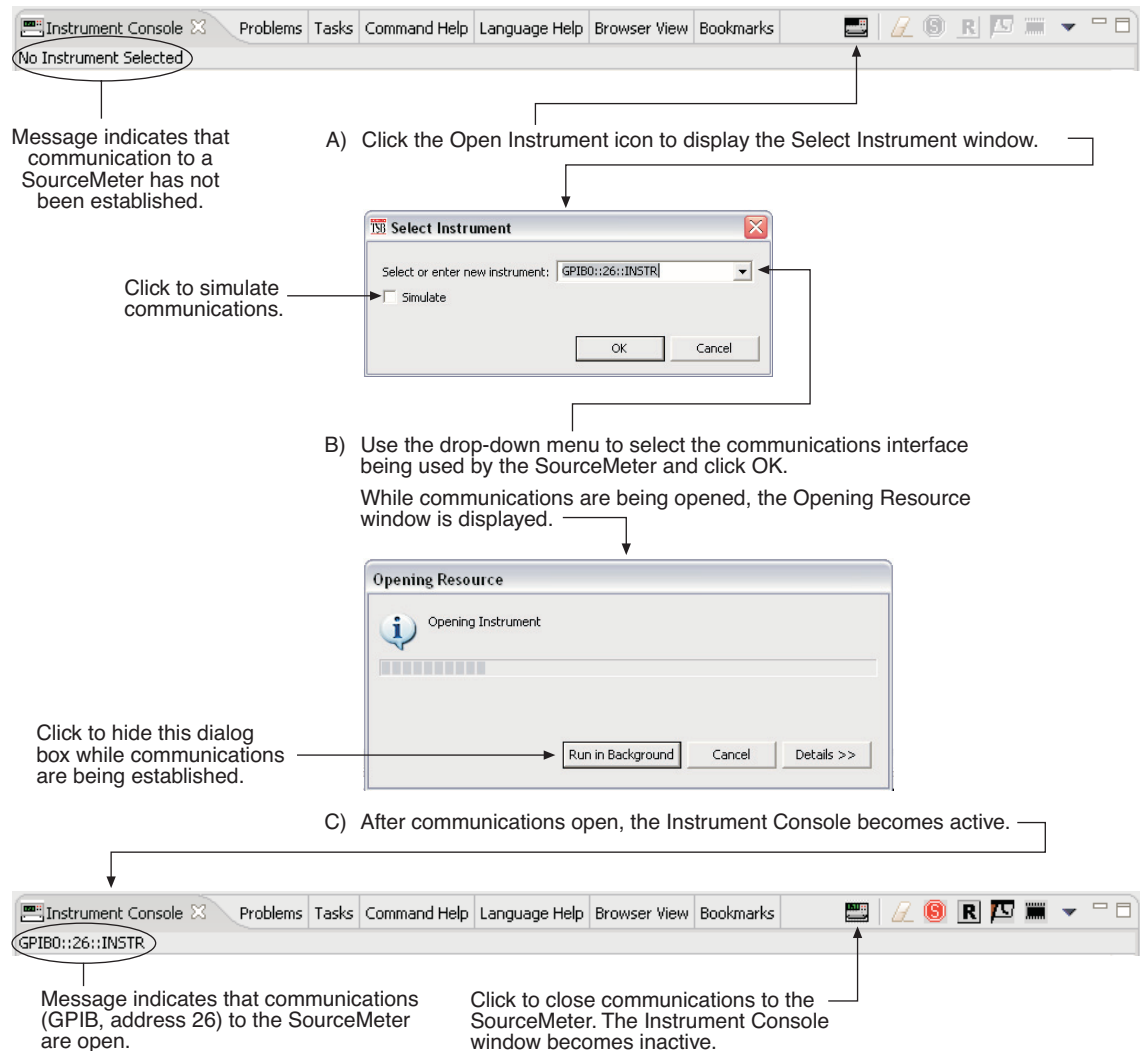
The Select Instrument window has a drop-down menu to select the GPIB or RS-232 interface being used by the Model 260x.

Simulate communications - If you select the Simulate option in the Select Instrument window, the Instrument Console will become active even though there will be no actual communication with the SourceMeter. You can simulate running a script or sending a command, but the SourceMeter will not respond.

**NOTE** The drop-down menu for the Menu icon can also be used to open or close communications between TSB and the SourceMeter. See [“Instrument Console icons”](#) on [page 2-28](#) for details on using the Menu icon.



Figure 2-6  
Opening and closing communications

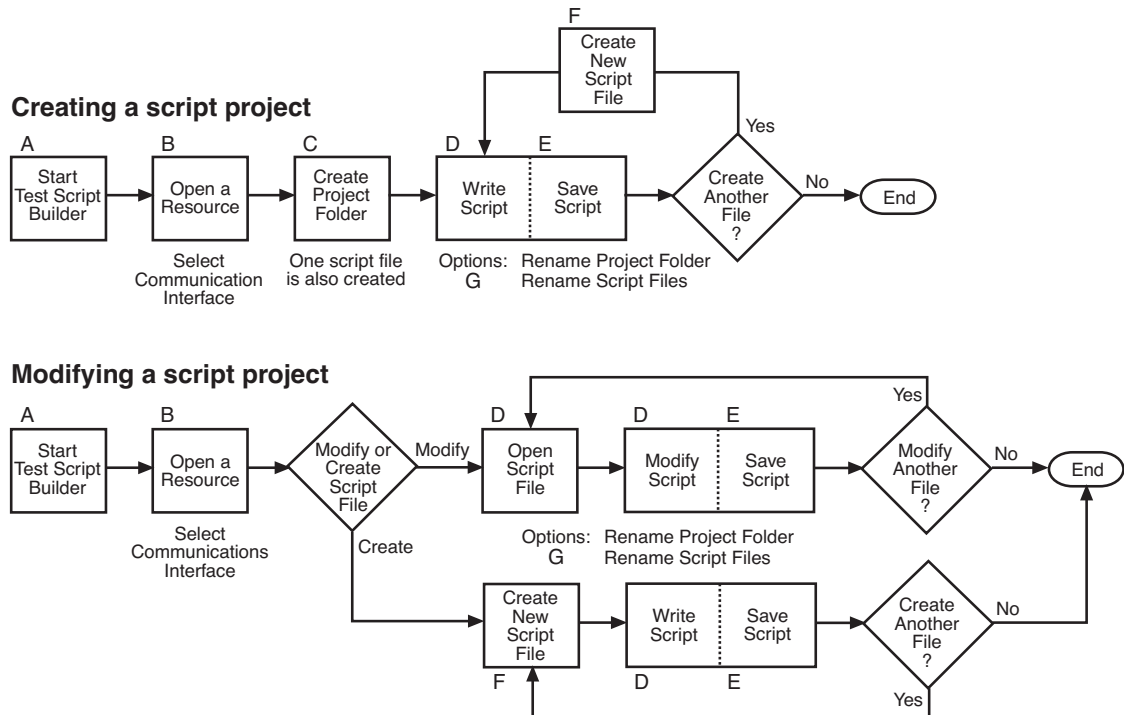


## Creating and modifying a script

The flowcharts in [Figure 2-7](#) show the basic processes to create and modify a script using the Test Script Builder. The labels (A through G) are used to identify reference links provided after the illustration.

Figure 2-7

### Creating and modifying a script using the Test Script Builder



Reference links for labels A through G shown in [Figure 2-7](#):

- A [“Starting Test Script Builder”](#) – Details on [page 2-14](#)
- B [“Opening communications”](#) – Details on [page 2-15](#)
- C [“Creating a project folder”](#) – Details on [page 2-18](#)
- D [“Writing or modifying a script”](#) – Details on [page 2-19](#)
- E [“Saving a script”](#) – Details on [page 2-19](#)
- F [“Creating new script files”](#) – Details on [page 2-20](#)
- G [“Renaming a project folder and/or script file”](#) – Details on [page 2-21](#)

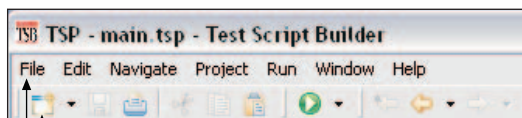
## Creating a project folder

When a project folder is created, the following actions occur:

- The project folder is added to the Project Navigator.
- A script file (named “main”) is created and placed in the project folder.
- The script file (which has no code) is opened and displayed in the Script Development area of the Test Script Builder.

The toolbar at the top of the Test Script Builder is used to create a project folder. [Figure 2-8](#) explains how to create a project folder.

Figure 2-8  
**Creating a project folder**

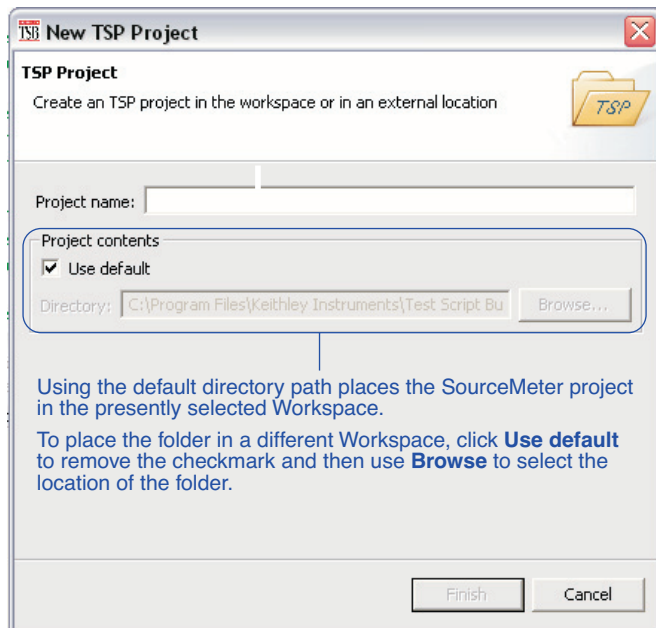


A) Open the New TSP Project dialog box as follows:  
Click the folder icon to display the New project wizard.  
In the wizard, select TSP Project and click Next.

OR

Click FILE to display the drop-down file menu. From the menu, click New and then click TSP Project.

B) In the New TSP Project window, type in a Project name (e.g., SourceMeasure) and click Finish:



## Writing or modifying a script

A script is a list of ICL commands and TSL statements. [Figure 2-5](#) shows a simple example of a script. When this script is run, it performs a beeper test. After sounding the beeper for three seconds at 1kHz, the message “Test Completed” is displayed on the Model 260x. See [page 2-39](#) for details on “User scripts”.

When a project or script file is created, the script file opens and is displayed in the Script Development area of the Test Script Builder. This is where a script can be written.

To modify an existing script file, it must be open. Open script files are presented as tabs in the Script Editor. To open and display a script file, click the file name in the Project Navigator. To display a different script file that is already open, click the appropriate tab at the top of the Script Editor.

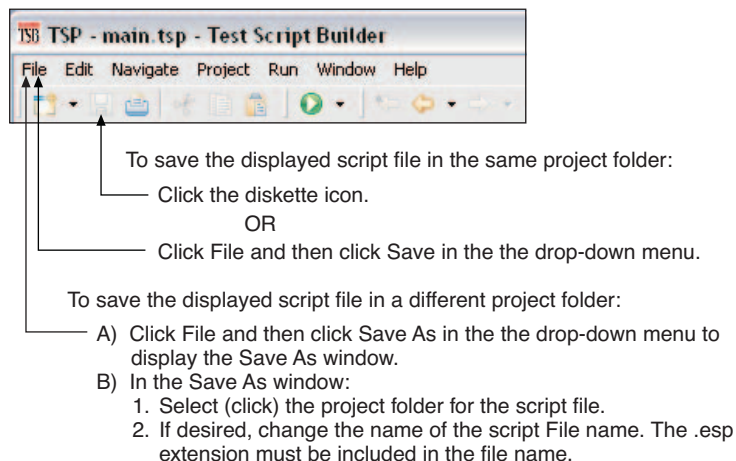
## Saving a script

It is good practice to routinely save a script file as lines of code are written or modified. The save operation performs error checking for the script. If an error occurs, an “X” will appear near the corrupt line of code, and the **Problems** tab will open to provide an explanation of the error. “X”s will also appear in the Project Navigator to indicate which project folder and which script file has the error.

The toolbar at the top of the Test Script Builder is used to save the displayed script file. As explained in [Figure 2-9](#), the script file can be saved in the same folder and/or saved in a different folder.

Figure 2-9

### Saving a script in the Test Script Builder

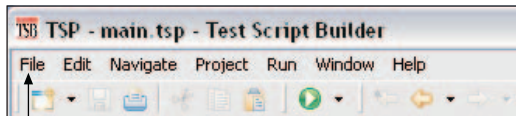


## Creating new script files

A script project can be made up of one or more script files. [Figure 2-10](#) shows how to add a script file to a project folder.

Figure 2-10

### Creating a new script file



A) Open the New TSP File window as follows:

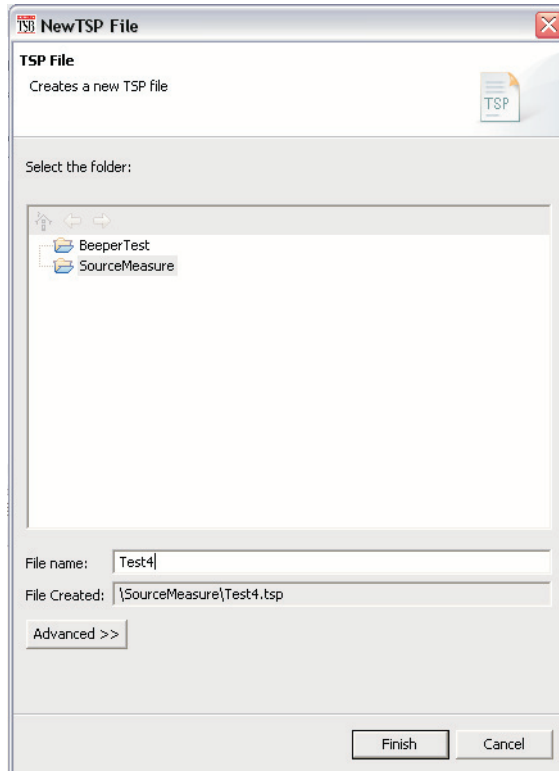
Click FILE to display the drop-down file menu. From the menu, click New and then click TSP File.

OR

In the Project Navigator, right-click the project folder for the script file. From the drop-down menu, click New and then click TSP File.

B) In the New TSP File window, make sure the desired project folder is selected. A folder is selected by clicking it.

C) Type in a file name (e.g., Test4) and click Finish:

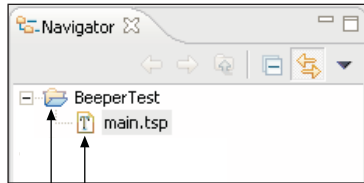


## Renaming a project folder and/or script file

When a new project is created, a script file (named “main”) is also created and placed in the Folder. [Figure 2-11](#) shows a project folder and script file that has been created and added to the Project Navigator. As shown, the project folder name and a script file name can be changed.

Figure 2-11

### Renaming a project folder and/or script file



#### To change the name of a script file:

- A) Right-click the script file, and click Rename in the drop-down menu.
- B) Type in the new name, making sure to include the .tsp extension, and then press the Enter key.

#### To change the name of a project folder:

- A) Right-click the project folder, and click Rename in the drop-down menu.
- B) Type in the new name, and then press the Enter key.

## Script launch configuration

A script is to be loaded into the Model 260x where it will be executed (run). The launch configuration options include the following:

- Select which script files will be included in the launch.
- Set the launch order for the selected script files.
- Set the script launch to load-only, or to load-and-execute (run).
- Set script storage for the Model 260x; volatile or non-volatile. A script stored in volatile memory will be lost when the SourceMeter power is turned off. A script stored in non-volatile memory will not be lost after power is turned off.

When a script project is created, the launch is configured initially as follows:

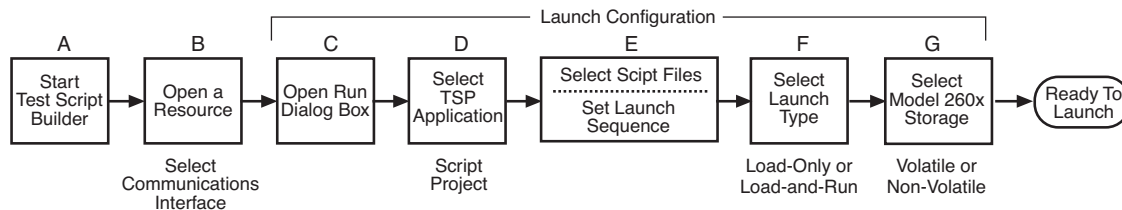
- Only the first script file (“main”) is selected to be included in the launch.
- The launch type is set to load-and-execute (run).
- The script project is set to be stored in the volatile memory of the Model 260x. The script will be lost when the Model 260x power is turned off.

**NOTE** If the initial launch configuration meets your requirements, the script is ready to be launched and is explained in [“Launching a script”](#) on page 2-25.

The flowchart in [Figure 2-12](#) shows the basic process to change the launch configuration for a script. The labels (A through G) are used to identify reference links which follow the illustration.

Figure 2-12

### Changing a launch configuration



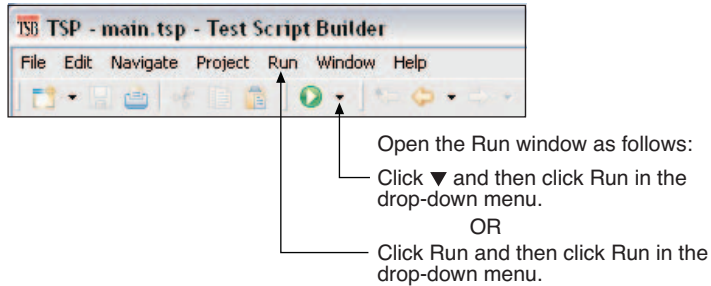
Reference links for labels A through G shown in [Figure 2-12](#):

- A [“Starting Test Script Builder”](#) – Details on [page 2-14](#)
- B [“Opening communications”](#) – Details on [page 2-15](#)
- C [“Displaying the launch configuration window”](#) – Details on [page 2-23](#)
- D [“Selecting a configuration”](#) – Details on [page 2-24](#)
- E [“Selecting script files and launch order”](#) – Details on [page 2-24](#)
- F [“Selecting the type of launch”](#) – Details on [page 2-24](#)
- G [“Storing the script”](#) – Details on [page 2-24](#)

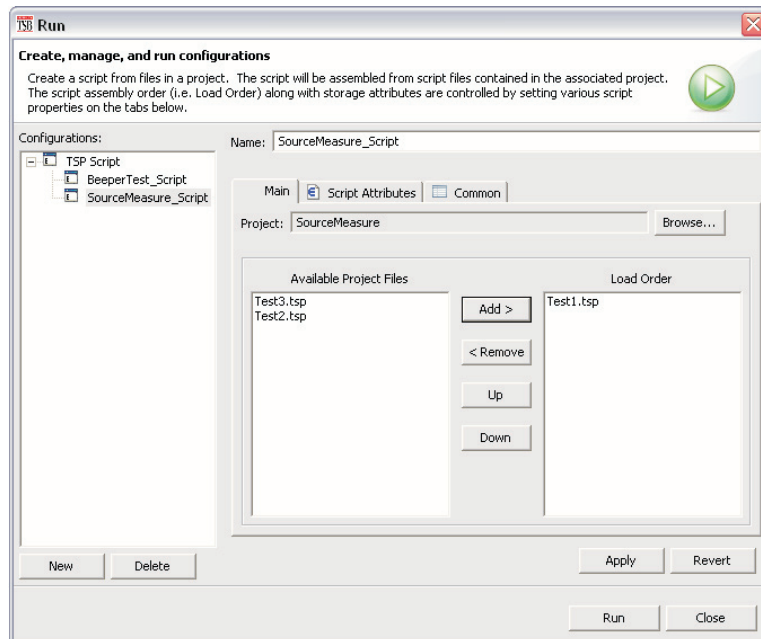
## Displaying the launch configuration window

A launch is configured from the **Run** dialog box. As shown in [Figure 2-13](#), use the tool bar at the top of the Test Script Builder to open the launch configuration window.

Figure 2-13  
Opening the Run dialog box (launch configuration)



Launch configuration - Main tab shown:





## Selecting a configuration

When a project is created using the Test Script Builder, a Configuration name for the launch is also created. The project name is altered to append “\_Script” to it. For example, for a project named “SourceMeasure”, the configuration will be named “SourceMeasure\_Script”.

In the Run window, the Configurations area lists the TSP Scripts. To view the launch configuration for a script, click the Configurations name. [Figure 2-13](#) shows the Main tab for “SourceMeter\_Script”.

## Selecting script files and launch order

As shown in [Figure 2-13](#), script files for the project are shown in the Main tab of the configuration window. Script files listed on the Available Project Files side of the tab are not selected to launch. Script files on the Load Order side are selected to launch in the order that they are listed.

Make configuration changes in the Main tab as follows:

- To move a script file to the Load Order side, click the file name and then click the Add > button.
- To move a file to the Available Project Files side, click the file name then click the < Remove button.
- For script files on the Load Order side, use the Up and Down buttons in a similar manner to change the launch sequence.
- After making changes in the Main tab, click the Apply button.

## Selecting the type of launch

There are two options for the launch process:

- **Load** – The script will load into the run-time environment of the Model 260x, but will not run. The script can be run later.
- **Load and Execute** – The script will load into the run-time environment. After the load process is completed, the script will run.
- **Auto Run** – With Load and Execute selected, Auto Run can be enabled. When enabled, the script will automatically run whenever the Model 260x is powered on.

## Storing the script

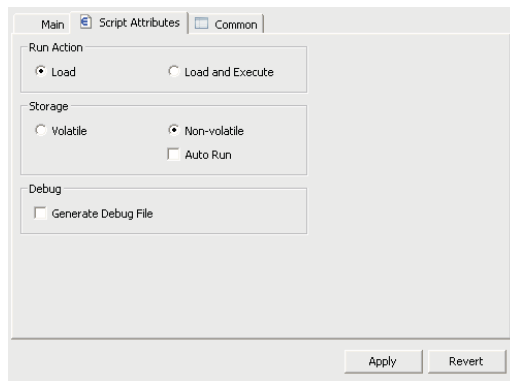
When a script is launched it can be stored in the volatile or non-volatile memory of the Model 260x. If stored in volatile memory, it will be lost when the SourceMeter power is turned off. If stored in non-volatile memory, it will not be lost when the power is turned off.

Script storage is set from the Script Attributes tab of the Run window and is shown in [Figure 2-14](#). In the Script Attributes tab, click Volatile or Non-volatile. After selecting non-volatile memory, Auto Run can be enabled (✓) to automatically run the script whenever the SourceMeter is turned on.

Debug - Click Generate Debug File to generate a read-only copy of the script. A folder named “Debug” and the debug file (.DBG) is added to the project.

After changing the storage configuration, click Apply.

Figure 2-14  
**Run dialog box (Script Attributes tab)**

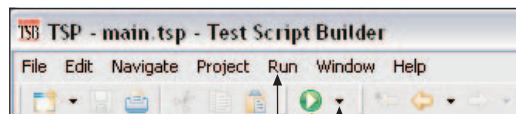


## Launching a script

After checking and/or changing a launch configuration, the script is launched from the Run dialog box by clicking the Run button shown in [Figure 2-13](#).

A script can be re-launched directly from the toolbar located at the top of the Test Script Builder. [Figure 2-15](#) explains how to re-launch a script from the toolbar.

Figure 2-15  
**Re-launching a script from the Test Script Builder toolbar**



Click Run and click Run Last Launched in the drop-down menu.

OR

Click ▶

OR

Click ▼ and then click the script in the drop-down menu.

## Running a TSP file

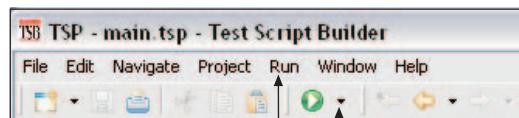
A TSP (.tsp) file does not have to be launched (loaded) into the Model 260x in order to be run. The code for a TSP file can simply be sent to the Model 260x and executed. The TSP file will not reside in the Model 260x (it is not saved in volatile or non-volatile memory). A TSP file can be run from the Project Navigator or from the tool bar at the top of Test Script Builder.

To run a TSP file from the Project Navigator, right-click the .tsp file name (e.g., main.tsp), select Run in the mouse menu, and then click Run As TSP File in the submenu.

A TSP file can also be run from the TSB tool bar as explained in [Figure 2-16](#).

Figure 2-16

### Re-launching a script from the Test Script Builder toolbar



Click Run or ▼, select Run As in the drop-down menu, then click 1 TSP File in the submenu.

A TSP file can also be run from the Menu icon on the Instrument Console tool bar. For details, see "[Instrument Console icons](#)" on [page 2-28](#).

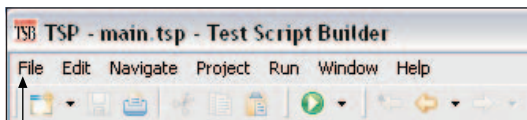
## Retrieving scripts from the Model 260x

A user script or factory script can be retrieved from memory of the Model 260x. The retrieved script folder will be placed in the Project Navigator with its script files opened.

[Figure 2-17](#) explains how to import a script from the Model 260x. It assumes that communications with the SourceMeter are already open. If communications are closed, a window will appear to open communications during the import process.

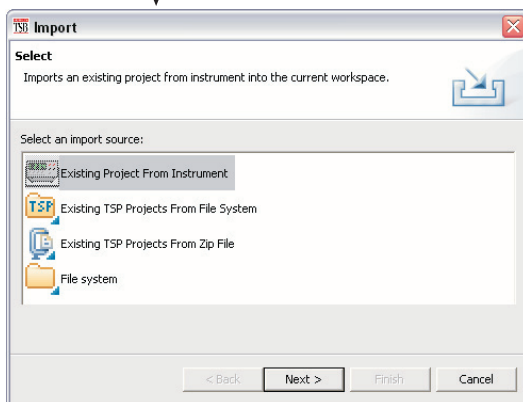
A modified script can be loaded back into the Model 260x as a user script using the same name or a new name. An imported factory script can only be loaded back into the Model 260x as a user script.

Figure 2-17  
**Importing a script (e.g., KIGeneral\_Script)  
from memory of the Model 260x**

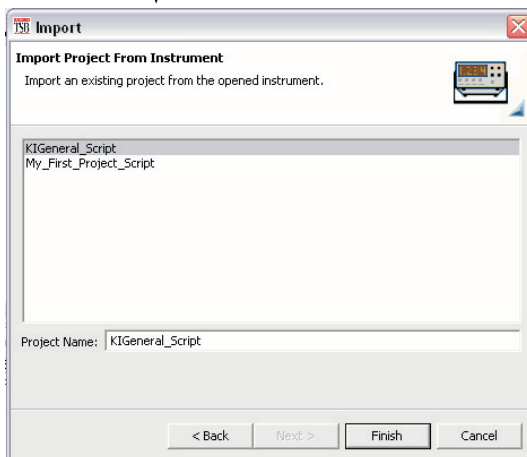


A) Click File to display the drop-down file menu and click Import to open the Import wizard.

B) In the Import Select box, click Existing Project From Instrument and then click Next.



C) In the Import Project From Instrument box, click the KIGeneral\_Script project, and then click Finish.



## Instrument Console

With communications established with the SourceMeter, the Instrument Console is used for the following operations:

- Execute chunks, which are individual ICL commands and TSL programming statements.
- Display returned data (readings and messages).
- Display error messages caused by erroneous code sent from the Instrument Console.

The instrument console is opened by clicking the Instrument Console tab in the lower window pane of the Test Script Builder (see [Figure 2-5](#)).

An active Instrument Console displays the TSP> prompt. Type in a command after the prompt and press Enter to execute it. For example, type in the following command:

```
TSP>reset ()
```

After pressing Enter, the SourceMeter will reset to its default settings.

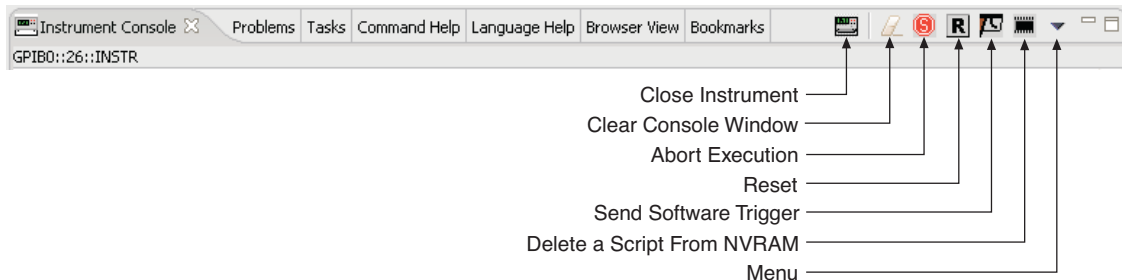
**NOTE** See “[Sending commands and statements](#)” for example code that can be sent from the Instrument Console.

Code and messages in the Instrument Console can be cleared by clicking the Clear Console Window icon shown in [Figure 2-18](#). It can also be cleared from the mouse menu as follows: Position the mouse pointer in the console window, right-click the mouse and then select Clear Console Window from the mouse menu.

### Instrument Console icons

After communications with the SourceMeter are open, all of the icons on the Instrument Console toolbar (shown in [Figure 2-18](#)) will be active.

Figure 2-18  
Instrument Console icons



The Instrument Console icons are explained as follows:

**Close Instrument** – With communications open, clicking this icon closes (disables) communications with the SourceMeter.

**Clear Console Window** – Clicking this icon removes all code and response messages from the Instrument Console window. There are two other ways to clear the Instrument Console window:

- Place the cursor in the console window, right-click the mouse, and then select Clear Console Window from the mouse menu.
- Click the Menu icon shown in [Figure 2-18](#) and click the Clear Console Window item in the menu.

**Abort Execution** – Clicking this icon aborts execution of a command sent from the Instrument Console.

**Reset** – Clicking this icon resets the SourceMeter. It is the same as sending the `reset()` command.

**Send Software Trigger** – Clicking this icon sends a software trigger to the SourceMeter. See “[Triggering](#)” in [Section 4](#) of this manual.

**Delete a Script From NVRAM** - Use this icon to delete a script from the non-volatile memory of the SourceMeter. After clicking this icon, select the script to be deleted from the displayed list, and click Delete.

**Menu** – Clicking this icon opens a menu with the following menu items:

- **Clear Console Window** – Click this menu item to clear the console window. Other ways to clear the console are explained above for the Clear Console Window icon.
- **Instrument** – Clicking this menu item opens a submenu to select items that perform the same operations as some of the other toolbar icons. Also included in the menu is the Flash item. The Keithley Flash Programmer is used to download firmware upgrades into the Model 260x. See “[Flash programmer](#)” on [page 2-34](#) for details on using the flash programmer.
- **Save Console** – The contents (code and response messages) of the Instrument Console window can be saved as a text (.txt) file. After clicking this menu item, a browser will open to allow you to save the log. Use any text editor, such as WordPad, to open the saved text file and view the log.
- **Run** – This menu item is used to run any TSP (.tsp) file that resides in the Project Navigator or elsewhere in your computer or network (see “[Running a TSP file](#)” on [page 2-26](#)). After selecting Run, a submenu will open with items to select Editor or Script File. Items for projects in the Project Navigator will also be listed in this submenu:
  - **Editor** – Selecting this item will open another submenu that will list all the TSP files that reside in the Project Navigator. Click a script file to run the script.

- **Script File** – Selecting this item will open a browser that allows you to locate a TSP file stored in your computer or network. With the File Name displayed in the browser, click Open to run the TSP file.
- **Projects** – The Run menu lists the projects that are in the Project Navigator. Select a project to display the TSP files for that project. Click a TSP file name to run the file.

The Menu icon is also displayed when the Problems, Tasks or Bookmarks tab is opened (displayed).

## Programming interaction tabs

Up to seven tabs can be displayed in the lower window pane of the Workspace to provide programming interaction between the Test Script Builder and the SourceMeter.

The tabs that can be placed in the Workspace include the following: Instrument Console, Problems, Tasks, Command Help, Language Help, Browser View and Bookmarks. Tabs not presently located in the Workspace can be added by selecting them from the Window option on the toolbar at the top of the Workspace as follows:

Click Window > Select Show View > Click the tab to be viewed

A tab in the Workspace can be opened (viewed) by clicking the tab name. When a tab is opened, an “X” will appear to the right of the tab name. Clicking this “X” removes the tab from the Workspace.

## Instrument Console tab

This tab (shown in [Figure 2-5](#)) is used to send commands to the connected SourceMeter. Retrieved data (e.g., readings) from commands and scripts appear in the Instrument Console.

**NOTE** [Figure 2-19](#) and [Figure 2-20](#) show partial screenshots of the following tabs.

## Problems tab

When a script file is saved, error checking is performed. If a script error is detected, an “X” will appear in the left-hand margin of the Script Editor at or near the corrupt line of code. The Problems tab will open automatically and provide a description of the error.

If you click the problem in the Problems tab, the line code that has the “X” will be highlighted in the Script Editor. After fixing the erroneous code, the problem will clear when the script file is saved.

## Tasks tab

This tab displays user-defined tasks associated with specific files, specific lines in specific files, as well as generic tasks that are not associated with any specific file.

A task marker (√) can be inserted for a line of code in the left-hand margin of the Script Editor. Right-click the line number for the code and select Add Task from the mouse menu. In the New Task window, type in a description of the task and click OK. The task will be added to the Task tab. If you click the task in the Tasks tab, the line of code that has the task marker will be highlighted in the Script Editor. A task can be cleared from the Script Editor by right-clicking the task marker and selecting Remove Task.

A task that is not linked to any code or file can be added to the Tasks tab. Place the mouse cursor in the Tasks tab, right-click the mouse, and then select Add Task to enter a description of the task.

### **Command Help tab**

This tab provided details on ICL functions and attributes. It has the same content as [Section 12](#) of the Reference Manual. The first page of Command Help provides links to the major topics of the help file. Click ICL commands list to display the list of functions and attributes. Click a function or attribute to display the details.

### **Language Help tab**

This tab provided details on the Test Script Language (TSL). It has the same content as “[Test Script Language \(TSL\) reference](#)” on [page 2-52](#) of this section. The first page of Language Help provides links to the major topics of the help file.

### **Browser View tab**

When on-line to the internet, this tab serves as a browser for the Keithley web site ([www.keithley.com](http://www.keithley.com)).

### **Bookmarks tab**

#### **Tasks tab**

This tab displays bookmarks that are placed in the Script Editor by the user. A bookmark is placed for a line of code in the left-hand margin of the Script Editor. Right-click the line number for the code and select Add Bookmark from the mouse menu. In the Add Bookmark window, type in a bookmark name and click OK. The bookmark name will be added to the Bookmarks tab.

In the Bookmarks tab, clicking a bookmark displays and highlights the line of code that has the bookmark. A bookmark can be removed from the Script Editor by right-clicking the bookmark and selecting Remove Bookmark.

The Bookmarks tab in [Figure 2-20](#) shows an example of using bookmarks. Each bookmark in the tab is linked to a function for a script file that exists in the Project Navigator. When a bookmark is clicked, the first line for that function will be displayed and highlighted in the Script Editor.



Figure 2-19

**Programming interaction tabs: Problems, Tasks and Command Help**

Problems tab:

Instrument Console Problems Tasks Command Help Language Help Bookmarks Browser View			
1 error, 0 warnings, 0 infos			
Description	Resource	In Folder	Location
`do' expected near `fo'	main.tsp	KIGeneral	line 47

Tasks tab:

Instrument Console Problems Tasks Command Help Language Help Bookmarks Browser View			
1 items			
Description	Resource	In Folder	Location
<input type="checkbox"/> -- TODO insert your code here.	main.tsp	My_First_Project	line 27

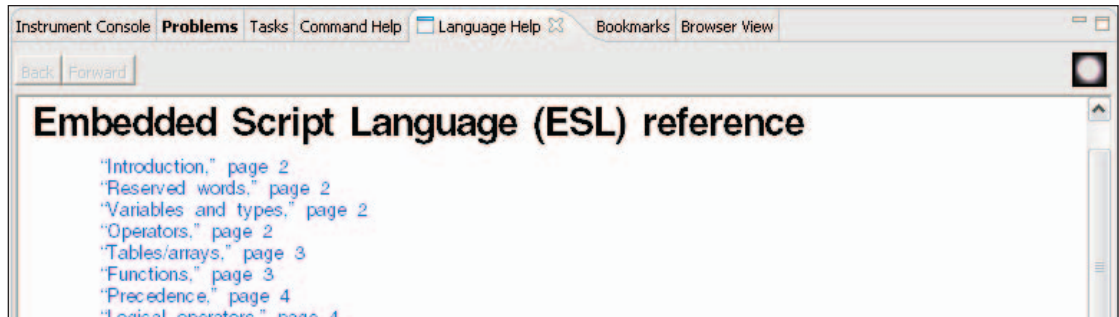
Command Help tab:

Instrument Console Problems Tasks Command Help Language Help Bookmarks Browser View	
Back	Forward
<h1>Instrument Control Library</h1>	
<p><b>Section 13 topics</b></p> <ul style="list-style-type: none"> <li>Command programming notes, <a href="#">page 13-2</a></li> <li>Conventions, <a href="#">page 13-2</a></li> <li>Instrument command types, <a href="#">page 13-3</a></li> <li>TSPlink nodes, <a href="#">page 13-4</a></li> <li>Logical instruments, <a href="#">page 13-4</a></li> <li>Reading buffers, <a href="#">page 13-5</a></li> <li>Time and date values, <a href="#">page 13-7</a></li> <li>ICL commands list, <a href="#">page 13-8</a></li> </ul>	

Figure 2-20

**Programming interaction tabs: Language Help, Bookmarks, Browser View**

Language Help tab:



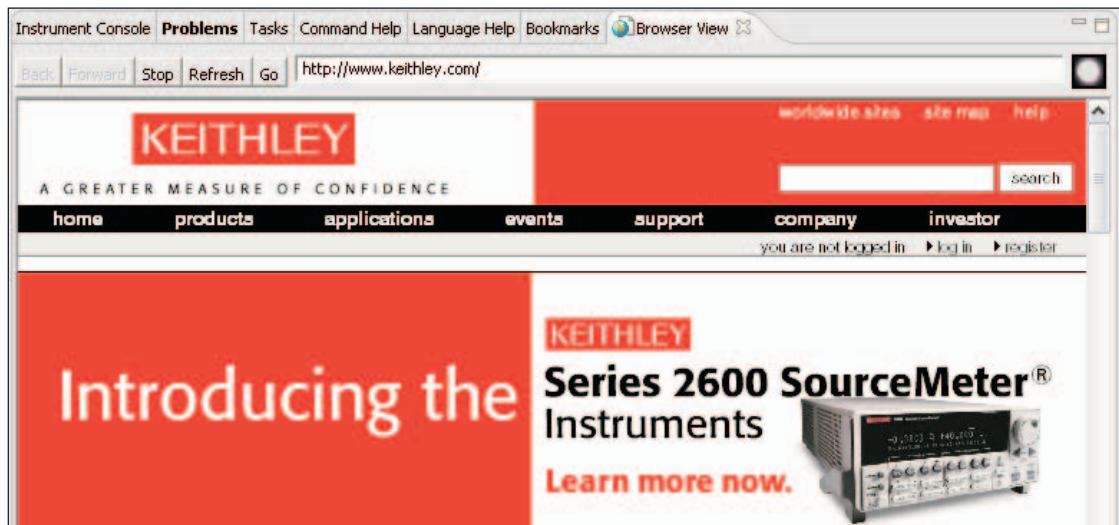
Bookmarks tab:

Instrument Console **Problems** Tasks Command Help Language Help Bookmarks Browser View

8 items

Description	Resource	In Folder	Location
PulseIMeasureV	main.tsp	KIGeneral	line 2
PulseVMeasureI	main.tsp	KIGeneral	line 74
SweepILinMeasureV	main.tsp	KIGeneral	line 147
SweepWLinMeasureI	main.tsp	KIGeneral	line 212
SweepILogMeasureV	main.tsp	KIGeneral	line 277
SweepVLogMeasureI	main.tsp	KIGeneral	line 342
SweepIListMeasureV	main.tsp	KIGeneral	line 407
SweepVListMeasureI	main.tsp	KIGeneral	line 467

Browser View tab:



## Flash programmer

When a firmware upgrade for the Model 260x becomes available, it can be downloaded from the Keithley website ([www.keithley.com](http://www.keithley.com)). New or enhanced factory scripts may be included in the upgrade. The file for the firmware upgrade can then be installed in the Model 260x using the flash programmer.

**CAUTION** External circuitry connected to input/output terminals while attempting a flash upgrade may cause instrument and/or DUT damage. Disconnect input/output terminals before performing a flash upgrade.

With communications between the TSB and the Sourcemeter opened, the flash programmer can be accessed using the Menu icon (see [Figure 2-18](#)) as follows:

Click Menu icon > Select Instrument > Click Flash

Use the displayed browser to select the downloaded file and click Open to start the upgrade. See “[Flash firmware upgrade](#)” on [page 13-13](#) for details.

## File management tasks

A project, along with its associated files (e.g., script files), resides in a workspace folder. Typical file management tasks include the creation of new projects and script files. (See “[Creating and modifying a script](#)” on [page 2-17](#) for details on file management tasks.) A script project can also be imported from a Model 260x into Test Script Builder where it can be modified (for details, see “[Retrieving scripts from the Model 260x](#)” on [page 2-26](#)).

Other typical file management tasks include “[Creating a new workspace](#)”, “[Importing a project from another workspace](#)”, “[Switching workspaces](#)”, and “[Deleting projects and/or script files](#)”. These file management tasks are explained as follows:

### Creating a new workspace

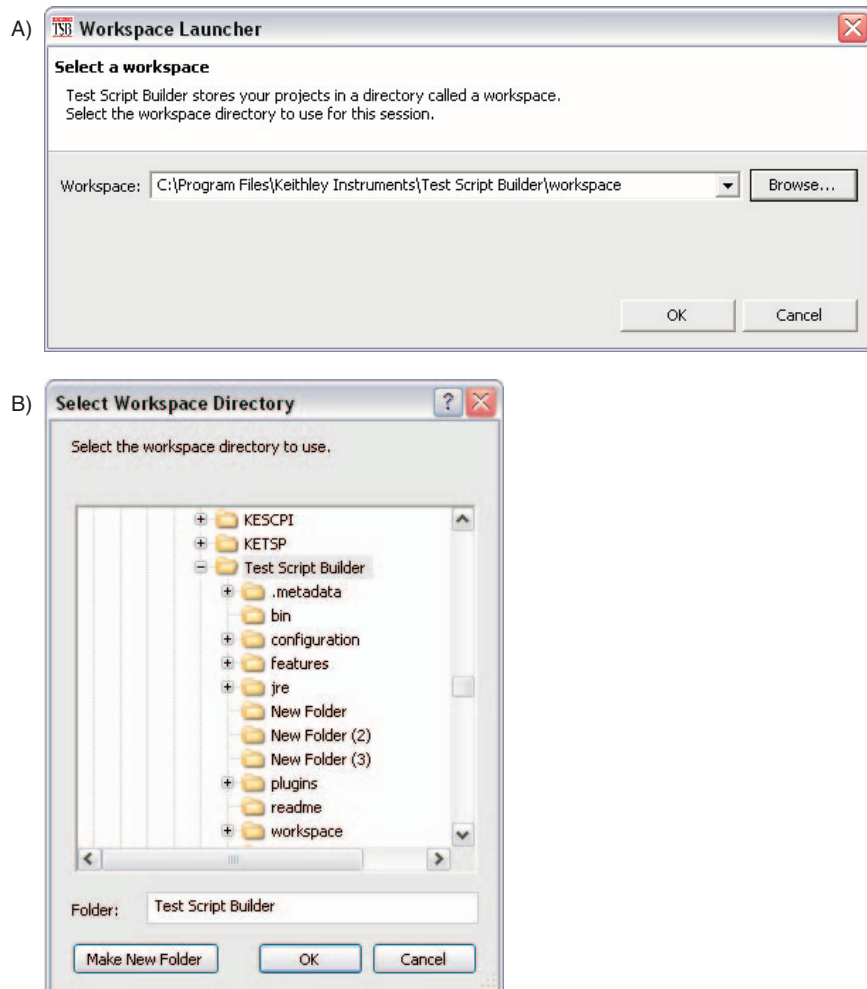
Additional workspaces can be created anywhere in your file system. A new workspace is simply a new folder for project files. A new folder for a workspace can be made from TSB as follows:

1. At the top of TSB, click File on the toolbar to open the file menu and then click Switch Workspace to open the Workspace Launcher ([Figure 2-21A](#)).
2. Click the Browse button to open the Select Workspace Directory browser and select the location for the new folder. [Figure 2-21B](#) shows the Test Script Builder folder selected as the location for the new workspace folder. Keep in mind that the workspace folder can be located anywhere in your file system.
3. In the Select Workspace Directory, click the Make New Folder button. A folder named New Folder will be inserted at the selected location.
4. In the browser, right-click New Folder and click Rename in the mouse menu.

5. Type in a name for the new workspace folder (e.g., workspace2) and press Enter.
6. In the browser, click OK, and then click OK in the Workspace Launcher. Test Script Builder will close and then re-open using the new workspace.

There will not be any projects residing in the Project Navigator for the new workspace. New projects and script files can be created as explained in “[Creating and modifying a script](#)” on [page 2-17](#). A project (along with its script files) can be imported into the new workspace from another workspace folder. See “[Importing a project from another workspace](#)” on [page 2-36](#).

Figure 2-21  
**Workspace Launcher and Select Workspace Directory**

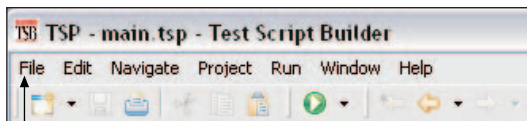


## Importing a project from another workspace

A project (along with its script files) can be imported from another workspace folder that resides in your file system. This is explained in [Figure 2-22](#), which imports a project named K12602Demo\_ASimpleTest. In Step C, use the Browser to locate the project that you wish to import.

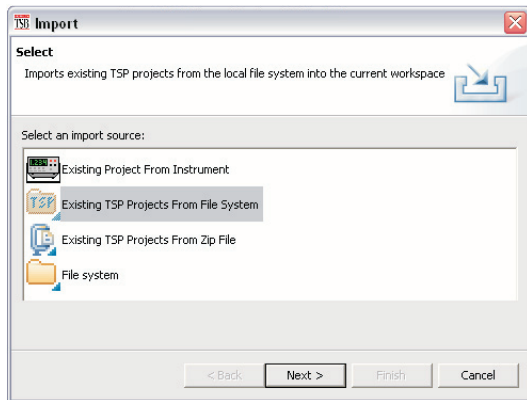
After clicking Finish in the Import window, the project will appear in the Project Navigator of the Test Script Builder.

Figure 2-22  
Importing a project from another workspace folder

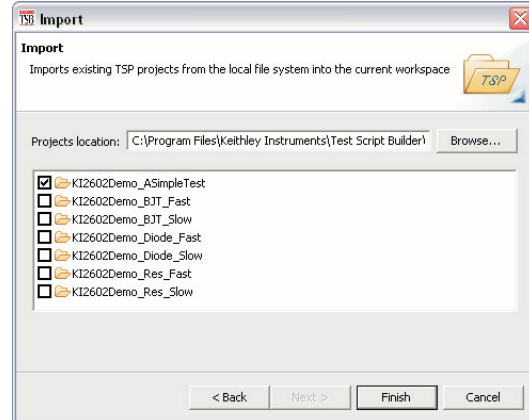


A) Click File to display the drop-down file menu and click Import to open the Import wizard.

B) In the Import Select box, click Existing TSP Project From File System and then click Next.



C) In the Import box, select ( ) the project to be imported, and then click Finish.



## Switching workspaces

Perform the following steps to switch to another workspace:

1. At the top of TSB, click File on the toolbar to open the file menu and then click Switch Workspace to open the Workspace Launcher ([Figure 2-21A](#)).
2. Click the Browse button to open the Select Workspace Directory browser ([Figure 2-21B](#)) and select the workspace folder. TSB will shut down and then re-open using the selected workspace.

## Deleting projects and/or script files

### Deleting a project

To delete a project, right-click the project in the Project Navigator and then click Delete in the mouse menu to display the Confirm Project Delete window (see [Figure 2-23](#)).

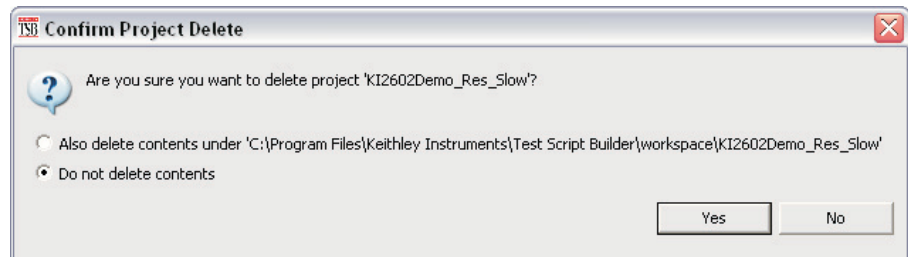
There are two project delete options:

- **Also delete contents under ...** (directory path for project) – This option deletes the project from the Project Navigator and also deletes the project from the workspace folder in your file system.
- **Do not delete contents** – This option deletes the project from the Project Navigator, but does not delete it from the workspace folder. The project can later be imported back into the Project Navigator (see [“Importing a project from another workspace”](#) on [page 2-36](#)).

After selecting the delete option, click Yes in the delete window to perform the deletion.

Figure 2-23

### Deleting a project



The script file will be deleted from the Project Navigator and will also be deleted from the workspace folder for the project.

### Deleting a script file

To delete a script file from a project, right-click the script file in the Project Navigator and then click Delete in the mouse menu. The script file will be deleted from the Project Navigator and will also be deleted from the workspace folder for the project.

## Sending commands and statements

Using your own program or the Test Script Builder, non-scripted chunks can be executed one line at a time. Responses (e.g., readings) are then transmitted back to the PC.

### Source-measure voltage and current

The primary function of an SMU is to source voltage or current, and measure current, voltage, resistance and/or power.

The following code fragments program `smua` to source-measure voltage. The measured current and voltage readings are then sent back to the PC.

Source V, and Measure I and V:

```
reset()                -- Returns SourceMeter to default settings.
smua.source.levelv = 1  -- Sets SMU A V-source level to 1V.
smua.source.output = smua.OUTPUT_ON -- Turns output on.
reading = smua.measure.iv() -- Performs I and V measurements.
print(reading)         -- PC displays I-measure reading.
smua.source.output = smua.OUTPUT_OFF -- Turns output off.
```

### Read and write to Digital I/O port

The Digital I/O port of the SourceMeter is used to control external circuitry (such as a component handler for binning operations). The I/O port has 14 input/output bits (lines) that can be at TTL logic state 1 (high) or 0 (low). The pinout for the Digital I/O port is shown in [Figure 10-1](#).

There are ICL commands to read and/or write to each individual bit, and commands to read and write to the entire port.

Use the following code fragment to write to one bit of the Digital I/O port. The I/O bit is then read and the state is returned to the PC where it is displayed.

```
digio.writebit(4,0)    -- Writes a "0" to I/O bit 4.
data=digio.readbit(4)  -- Reads I/O bit 4.
print(data)            -- PC displays state of I/O bit 4.
```

### Display user-defined messages

The operator can define and display messages on the front panel display of the SourceMeter. The following code fragment displays the "Test in Process" message on the SourceMeter display:

```
display.clear          -- Clears display of messages.
display.settext("Test in Process") -- Displays message.
```

Displayed messages and input prompts are used in scripts to prompt the operator to enter parameter values from the front panel. See “[Interactive script](#)” (on [page 2-40](#)) for more information.

## User scripts

User scripts can be written using your own program or the Test Script Builder. User scripts are loaded into the Model 260x and can be saved in non-volatile memory. Scripts not saved in non-volatile memory will be lost when the Model 260x is turned off.

### Script examples

#### Script using commands and statements only

The script in [Table 2-1](#) sweeps voltage (1V to 5V) and measures current at each step. The five current readings are returned to the host computer:

Table 2-1

Example script to sweep V and measure I

Test Script Builder	User’s Program Script
<pre>current = {} smua.source.output = smua.OUTPUT_ON for j = 1, 5 do     smua.source.levelv = j     current[j] = smua.measure.i()     print(current[j]) end smua.source.output = smua.OUTPUT_OFF</pre>	<pre>loadscript     current = {}     smua.source.output = smua.OUTPUT_ON     for j = 1, 5 do         smua.source.levelv = j         current[j] = smua.measure.i()         print(current[j])     end     smua.source.output = smua.OUTPUT_OFF endscript</pre>

#### NOTE

When creating a script using the Test Script Builder, only the chunk is typed in as shown above. See “[Using Test Script Builder](#)” on [page 2-12](#) for details on creating, loading and running the script.

When creating a script using a programming language, shell commands must be included to manage interactions between the host computer and TSP. The `loadscript` command loads the script into the Model 260x and `endscript` signifies the end of the script.



## Script using a function

TSL facilitates grouping commands and statements using the `function` keyword. Therefore, a script can also consist of one or more functions. Once a script has been RUN, the host computer can then call a function in the script directly.

The script in [Table 2-2](#) contains an ICL command to set measurement speed (NPLC) and a function (named `sourcev`). When this script is run, the measurement speed will set to 0.5 PLC and make the `sourcev` function available for calling.

Table 2-2

### Example script using a function

Test Script Builder	User's Program Script
<pre>smua.measure.nplc = 0.5 function sourcev(v)     smua.source.levelv = v     i = smua.measure.i()     print(i)     return(i) end</pre>	<pre>loadscript     smua.measure.nplc = 0.5     function sourcev(v)         smua.source.levelv = v         i = smua.measure.i()         print(i)         return(i)     end endscript</pre>

When calling the function, you must specify the source voltage in the argument for the function. For example, to set the source to 2V, call the function as follows:

```
sourcev(2)
```

Assuming SMU A output is on, it will output 2V and measure the current. The current reading is sent to the host PC and displayed.

## Interactive script

An interactive script prompts the operator (via the SourceMeter display) to input test parameters (via the SourceMeter front panel). The chunk fragment in [Table 2-3](#) uses display messages to prompt the operator to select an SMU Channel (A or B), a source function (I or V) and to input the source level. When an input prompt is displayed, the script will wait until the operator inputs the parameter and/or presses the ENTER key.

The `display.prompt` command, in the following script, prompts the user to input a source level. If a value is not entered, the default level (1mA or 1V) will be set when ENTER is pressed. The operator will not be able to input values that are not within the minimum (0.5mA or 0.1V) and maximum (3mA or 10V) limits.

Table 2-3

**Example interactive chunk fragment for a script**

<b>Script Chunk Fragment (Test Script Builder or User's Program)</b>
<pre>--Prompt operator to select channel:   chan = display.menu ("Select Channel", "smua smub")   if (chan == "smua") then     chan = smua   end   if (chan == "smub") then     chan = smub   end  --Prompt operator to select (input) the source function:   func = display.menu("Select Function", "amps volts")   if (func == "amps") then     chan.source.func = chan.OUTPUT_DCAMPS   else chan.source.func = chan.OUTPUT_DCVOLTS   end  --Prompt operator to set (input) source level:   if (func == "amps") then     level = display.prompt("0.0E+00", " mA", "Enter I level",       1E-3, 0.5E-3, 5E-3)   else     level = display.prompt("00.0", " V", "Enter V level",       1, 0.1, 10)   end  --Wait for operator to set source level:   if (func == "amps") then     chan.source.leveli = level   else chan.source.levelv = level   end</pre>

## Creating a user script

To create a script and load it, the test program (chunk) must be framed by the following shell commands: `loadscript` or `loadandruncscript`, and `endscript`.

**Load only** – The following scripts will load only into the run-time environment of the Model 260x. The script on the left is unnamed, while the one on the right is named (where `name` is the user-defined name):

```
loadscript          loadscript name
(chunk)            (chunk)
endscript          endscript
```

**Load and run** – The following scripts will load into the run-time environment and then run. Keep in mind that when a script is run, only the chunk is executed. The script on the left is unnamed, while the one on the right is named (where `name` is the user-defined name):

```
loadandruncscript  loadandruncscript name
(chunk)            (chunk)
endscript          endscript
```

Details on `loadscript` and `loadandruncscript` are provided as follows:

### **loadscript**

**loadscript** `name`

where: `name` is the user assigned name for the script.

The `loadscript` shell command loads the script into the run-time environment. The script can be assigned a name or it can be left nameless. If assigning a name that already exists for another loaded script, the old script will be overwritten with the new script.

If a script is not named when it is loaded into the run-time environment, it will be lost when another script is loaded or when the Model 260x is turned off. After loading the unnamed script, use the `run()` or `script.run()` command to run it.

A special `name` for a script is `autoexec`. After an `autoexec` script is saved in non-volatile memory, the script will automatically run after the Model 260x is powered on and all autorun scripts have been executed. For details, see [“Autoexec script” on page 2-45](#) and [“Autorun scripts” on page 2-44](#).

### **loadandruncscript**

**loadandruncscript** `name`

where: `name` is the user assigned name for the script.

These commands are similar to the `loadscript` commands except that the script will execute (run) after it is loaded into the run-time environment. Also, the `autorun` attribute for a named script will be set to “yes” (see [“myscript.autorun” on page 2-44](#)).

## Saving a user script

A created and loaded script does not have to be saved in non-volatile memory of the Model 260x before it can be run. However, an unsaved script will be lost when the Model 260x is turned off.

### Saving a named script

Only a named script can be saved in non-volatile memory of the Model 260x. After creating and loading a named script, use one of the following commands to save it.

```
myscript.save()  
myscript.save("name")
```

where: `myscript` is the user-defined name of the script.  
`name` is a new name for the script that is assigned by the user. Using this function is equivalent to the “Save As” file menu item in the Test Script Builder.

Either of the above save commands will save the script in non-volatile memory. If a script is not saved in non-volatile memory, the script will be lost when the Model 260x is turned off.

The `myscript.save()` command saves the script under the name that it was originally created and loaded. The `myscript.save("name")` shell command is used to save the script under a different name. If you save the script to a name that already exists in non-volatile memory, it will be overwritten.

Examples:

1. Assume a script named “test1” has been created and loaded. The following command saves the script in non-volatile memory:

```
test1.save()
```

2. To save the script named “test1” under a new name (“test2”) in non-volatile memory, send the following command:

```
test1.save(test2)
```

## Running a user script

### Running an unnamed script

There can only be one unnamed script in the run-time environment. If another unnamed script is created and loaded, the previous unnamed script will be removed from the run-time environment. Use one of the following commands to execute the chunk of the last loaded unnamed script. Both commands perform the same operation.

```
run()  
script.run()
```

## Running a named script

Any named script that is in the run-time environment can be run using one of the following commands. Both commands perform the same operation.

```
myscript ()  
myscript.run ()
```

where: `myscript` is the user-defined name of the script.

Example:

Assume a script named “test3” has been loaded into the run-time environment. The following command executes the chunk of the script.

```
test3 ()
```

## Running scripts automatically

Scripts can be set to run automatically when the Model 260x is turned on. One or more scripts can be set to autorun, and one script can be set to autoexec.

### Autorun scripts

When a saved script is set to autorun, it will automatically load and run when the Model 260x is turned on. Any number of scripts can be set for autorun. The run order for these scripts is arbitrary, so make sure the run order is not important.

To set a script for autorun, set the following autorun attribute to “yes”. Setting it to “no” disables autorun.

```
myscript.autorun
```

where: `myscript` is the user-defined name of the script.

Make sure to save the script in non-volatile memory after setting the autorun attribute.

Example:

Assume a script named “test5” is in the run-time environment. The script can be set to autorun as follows:

```
test5.autorun = "yes"  
test5.save ()
```

The next time the Model 260x is turned on, the “test5” script will automatically load and run.

<p><b>NOTE</b>    <b>The <code>loadandrunscript name</code> command sets the autorun attribute for that script to “yes”. To cancel autorun, set the autorun attribute to “no” and save the script.</b></p>
--

## Autoexec script

One script can be designated as the autoexec script. When the Model 260x is turned on, the autoexec script will start after all the autorun scripts have run.

```
loadscript autoexec
loadandruncscript autoexec
```

An autoexec script can be formed by creating a new script and naming it `autoexec` (as shown above using `loadscript` or `loadandruncscript`). After loading the new script, send the `autoexec.save()` command to save it in non-volatile memory. See “[Creating a user script](#)” (on [page 2-42](#)) for details on creating a script.

An autoexec script can also be created by changing the name of an existing script that is saved in non-volatile memory by using the following command:

```
myscript.save("autoexec")
```

where: `myscript` is the user-defined name of the script.

Example:

Assume a script named “test6” is saved in non-volatile memory. That script can be made into an autoexec script as follows:

```
test6.save("autoexec")
```

The next time the Model 260x is turned on, the “test6” script will automatically load and start after all of the autorun scripts have run.

## Running a user script from the Model 260x front panel controls

In order to run a user script from the front panel, an entry for the script needs to be added to the User menu for the LOAD key. The following commands are used to enter or delete a name into the User menu:

```
display.loadmenu.add(displayname, script)
display.loadmenu.delete(displayname)
```

where: `displayname` is the name to be added to (or deleted from) the User menu.

`script` is the name of the script.

It does not matter what order the items are added to the User menu. Menu items will be displayed in alphabetical order when the menu is selected.

Example:

Assume a user script named “Test9” has been loaded into the run-time environment. Add the name (“Test9”) to the User menu for the script as follows:

```
display.loadmenu.add(Test9, Test9)
```

After adding a name to the User menu, the script can then be run from the front panel as follows:

1. Press the LOAD key.
2. Select User.
3. Select the user script to run and press the RUN key.

## Modifying a user script

A user script stored in non-volatile memory can be modified by retrieving the script listing for the script. The retrieved script can then be modified, loaded and saved in non-volatile memory. See [“Retrieving a user script listing”](#) (on [page 2-46](#)) for details.

**NOTE** If using the Test Script Builder to modify a user script stored in non-volatile memory, the script listing should be retrieved from in the Project Navigator (see [“Retrieving scripts from the Model 260x”](#) on [page 2-26](#)).

## Script management

### Retrieving a user script listing

The listing for a user script can be retrieved from non-volatile memory. The listed script can then be modified and saved as a user script under the same name or a new name.

**NOTE** A modified user script can be loaded back into the Model 260x using the same name or a new name.

The following command returns a catalog listing of the user scripts stored in the Model 260x:

```
script.user.catalog()
```

Example:

Retrieve the catalog listing for user scripts:

```
for name in script.user.catalog() do
  print (name)
end
```

The following function retrieves a script listing. The script chunk is returned, along with the shell keywords (`loadscript` or `loadandrunscript`, and `endscript`):

```
myscript.list()
```

where: `myscript` is the user-defined name of the script.

Example:

Retrieve the listing for a saved script named "test7":

```
userscriptlist = test7.list()  
print (userscriptlist)
```

### Deleting a script from non-volatile memory

Replacing, changing or deleting a script from the run-time environment does not remove the script from non-volatile memory. A script can be permanently removed from non-volatile memory using either of the following commands:

```
script.delete("name")  
script.user.delete("name")
```

where: `name` is the user-defined name of the script.

Example:

Delete a user script named "test8" from non-volatile memory:

```
script.delete("test8")
```

### Restoring a script in the run-time environment

A script is inherently a global variable and can be replaced by assigning a new value or by loading a new script with the same name. It can also be removed from the run-time environment by assigning it the nil value. A script can be restored from non-volatile memory back into the run-time environment using either of the following commands:

```
script.restore("name")  
script.user.restore("name")
```

where: `name` is the user-defined name of the script to be restored.

Example:

Restore a user script named "test9" from non-volatile memory:

```
script.restore("test9")
```



## Factory scripts

A factory script is basically the same as a user script, except a factory script is created by Keithley at the factory and is permanently stored in non-volatile memory. Factory scripts are documented in [Section 13](#).

Most of the information for “[User scripts](#)” (page 2-39) also applies to factory scripts. The differences between a user script and a factory script include the following:

- A factory script cannot be deleted from non-volatile memory.
- The script listing for a factory script can be retrieved and modified, but it will then be treated as a user script. A user script cannot be saved as a factory script.

### Running a factory script

Use either of the following commands to run a factory script:

```
script.factory.scripts.name()  
script.factory.scripts.name.run()
```

where: `name` is the name of the factory script.

Example:

Run the factory script named “sourceMeasureDC”:

```
script.factory.scripts.sourceMeasureDC()
```

### Running a factory script from the Model 260x front panel controls

1. Press the LOAD key.
2. Select Factory.
3. Select the function to run and press the RUN key.

### Modifying a factory script

#### Retrieving a factory script listing

The script listing for a factory script can be retrieved and modified. However, it cannot be saved as a factory script. The modified script can be saved as a user script using the same name or a new name.

<b>NOTE</b>	<b>An imported factory script can only be loaded back into the Model 260x as a user script.</b>
-------------	---

The following command returns a catalog listing of the factory scripts stored in the Model 260x:

```
script.factory.catalog()
```

Example:

Retrieve the catalog listing for factory scripts:

```
for name in script.factory.catalog() do
    print (name)
end
```

The following function retrieves a script listing. The script chunk is returned, along with the shell keywords (`loadscript` or `loadandrunscript`, and `endscript`):

```
script.factory.scripts.name.list()
```

where: `name` is the name of the factory script.

Example:

Retrieve the script listing for a factory script named "sourceMeasureDC":

```
factoryscriptlist = script.factory.scripts.sourceMeasureDC()
print (factoryscriptlist)
```

## Differences: remote vs. local state

The Model 260x can be in either the local state or the remote state. When in the local state (REM annunciator off), the instrument is operated using the front panel controls. When in the remote state (REM annunciator on), instrument operation is being controlled by the PC. When the instrument is powered-on, it will be in the local state.

### Remote state

The following actions will place the instrument in the remote state:

- Sending a command from the PC to the instrument.
- Running a script (FACTORY or USER test) from the front panel. After the test is completed, the instrument will return to the local mode.
- Opening communications between the instrument and Test Script Builder.

While in the remote state, front panel controls are disabled. However, the LOCAL key will be active if it has not been locked out. When an interactive script is running, the front panel controls will be active to allow the operator to input parameter values.

## Local state

The following actions will cancel the remote state and return the instrument to the local state:

- Cycling power for the instrument.
- Pressing front panel LOCAL key (if it is not locked out).
- Sending the `abort` command from the PC.
- Clicking the Abort Execution icon on the toolbar of the Instrument Console for Test Script Builder.
- After a front panel script (FACTORY or USER test) is completed, the instrument will return to the local state.

## TSP-Link system

A test system can be expanded to include up to 64 TSP-Linked enabled instruments. The system can be stand-alone or PC-based. Details on system expansion using the TSP-Link are provided in [Section 9](#).

**Stand-alone system** – A script can be run from the front panel of any node (instrument) in the system. When a script is run, all nodes in the system go into remote operation (REM annunciators turn on). The node running the script becomes the Master and can control all of the other nodes, which become its Slaves. When the script is finished running, all the nodes in the system return to local operation (REM annunciators turn off), and the Master/Slave relationship between nodes is dissolved.

**PC-based system** – When using a PC, the GPIB or RS-232 interface to any single node becomes the interface to the entire system. When a command is sent via one of these interfaces, all nodes go into remote operation (REM annunciators turn on).

The node that receives the command becomes the Master and can control all of the other nodes, which become its Slaves. In a PC-based system, the Master/Slave relationship between nodes can only be dissolved by performing an abort.

# Memory considerations

## (run-time environment)

The Model 260x reserves 8MB of memory for dynamic run-time use. Of this memory, the firmware requires up to approximately 5MB for normal system operation. That leaves approximately 3MB of remaining memory that is available to the user. These numbers for memory are affected by the number of scripts loaded into non-volatile memory. The given memory values represent the memory available when the factory script as well as the user script non-volatile memory areas are filled to capacity.

The run-time environment and user created reading buffers must fit within the 3MB of memory loosely reserved. It is possible for memory used for these purposes to be greater than 3MB. When this occurs, there is a risk that memory allocation errors will be generated and commands will not be executed as expected.

If memory allocation errors are encountered, the state of the instrument cannot be guaranteed. After attempting to download any data from the instrument, it is recommended that power to the instrument be cycled to return it to a known state. Cycling power will reset the run-time environment and all user-created reading buffers. Any data not stored in the non-volatile reading buffers will be lost.

The amount of memory used by the run time environment can be checked using the `gcinfo` function. The first value returned by `gcinfo` is the number of kilobytes of memory being used by the run time-environment. This does not include the amount of memory used by reading buffers.

The amount of memory used by a reading buffer is approximately 15 bytes for each entry requested. There is a slight amount of overhead for a reading buffer but this can be ignored for memory utilization calculations. For example, assume two reading buffers were created. One of them was created to store up to 1,000 readings and the other 2,500. The memory reserved for the reading buffers is calculated as follows:

$$(1000 \times 15) + (2500 \times 15) = 52,500 \text{ bytes or } 52.5 \text{ kilobytes.}$$

Note that the non-volatile reading buffers do not consume memory needed by the run-time environment. Do not include them in your memory consumption calculations. Also, reading buffers for remote nodes consume memory on the remote node, not the local node. You should be sure the total reading buffer memory for any particular remote node does not exceed 3MB, but do not include that amount from your local memory consumption calculations.

# Test Script Language (TSL) reference

## Introduction

A script is a program that the Test Script Processor (TSP) executes. A script is written using the Test Script Language (TSL). TSL is an efficient language, with simple syntax and extensible semantics. TSL is derived from the Lua programming language, Copyright © 1994-2004 Tecgraf, PUC-Rio. See <http://www.lua.org>, the official web site for the Lua Programming Language, for more information. <http://lua-users.org> internet site is created for and by users of Lua programming language and is another source of useful information.

## Reserved words

and	function	return
elseif	nil	until
for	repeat	else
local	true	false
then	do	in
break	if	or
end	not	while

## Variables and types

TSL has six basic types; nil, boolean, number, string, function, and table. TSL is a dynamically typed language, which means variables do not need to be declared as a specific type. Instead, variables assume a type when a value is assigned to them. Therefore, each value carries its own type. If a variable has not been assigned a value, the variable defaults to the type nil. All numbers are real numbers. There is no distinction between integers and floating-point numbers in TSL.

```
var = nil                -- var is nil.
var = 1.0                -- var is now a number.
var = 0.3E-12           -- var is still a number.
var = 7                  -- var is still a number.
var = "Hello world!"    -- var is now a string.
var = "I said, Hello world!" -- var is still a string.
var = function(a, b) return(a+b) end -- var is now a function
-- that adds two numbers.
var = {1, 2., 3.00e0}    -- var is now a table (i.e.,
-- array) with three
-- initialized members.
```

Nil is a type with a single value, `nil`, whose main property is to be different from any other value. Global variables have a `nil` value by default, before a first assignment, and you can assign `nil` to a global variable to delete it. TSL uses `nil` as a kind of non-value, to represent the absence of a useful value.

## Operators

Arithmetic Operators:	Relational Operators:	Logical Operators:
+ (addition)	< (less than)	and
- (subtraction)	> (greater than)	or
* (multiplication)	<= (less than or equal)	not
/ (division)	>= (greater than or equal)	
- (negation)	~= (not equal)	
	== (equal)	

## Tables/arrays

TSL makes extensive use of the data type “table”, which is essentially a very flexible array like data type.

```
-- Define a table.
--
atable = {1, 2, 3, 4}    -- A table with four elements, which are numbers.
-- Let's print it:
i = 1                    -- Tables are indexed on one, NOT zero.
-- atable[index] is true if there is an element at that index.
-- nil is returned otherwise. 0 does NOT evaluate to false, only nil
-- does.
--
while atable[i] do
    print (atable[i])    -- Index into table using a number.
    i = i + 1
end
```

Output of code above:

```
1
2
3
4
```

## Functions

TSL allows you to define functions. A function can take a predefined number of parameters and return multiple parameters if desired.

```
-- Let's define a function and call it.
```

```
--
```

```
function add_two(parameter1, parameter2)
    return(parameter1 + parameter2)
end
```

```
print(add_two(3, 4))
```

```
-- Below is an alternate syntax for defining a function. Functions are
-- first-class values in TSL, which means functions can be stored in
-- variables, passed as arguments, and returned as results if desired.
```

```
--
```

```
add_three = function(parameter1, parameter2, parameter3)
    return(parameter1 + parameter2 + parameter3)
end
```

```
print(add_three(3, 4, 5))
```

```
-- A function that returns multiple parameters; sum, difference, and ratio
-- of the two numbers passed to it.
```

```
--
```

```
function sum_diff_ratio(parameter1, parameter2)
    psum = parameter1 + parameter2
    pdif = parameter1 - parameter2
    prat = parameter1 / parameter2
    return psum, pdif, prat
end
```

```
sum, diff, ratio = sum_diff_ratio(2,3)
print(sum)
print(diff)
print(ratio)
```

Output of code above:

```
7
12
5
-1
0.66666
```

## Precedence

Operator precedence in TSL follows the table below, from higher to lower priority:

```

^
not  - (unary)
*    /
+    -
.. (concatenation)
<    >    <=    >=    ~=    ==
and
or

```

All operators are left associative, except for '^' (exponentiation) and '..', which are right associative. Therefore, the following expressions on the left are equivalent to those on the right:

$a+i < b/2+1$	$(a+i) < ((b/2)+1)$
$5+x^2*8$	$5+((x^2)*8)$
$a < y \text{ and } y <= z$	$(a < y) \text{ and } (y <= z)$
$-x^2$	$-(x^2)$
$x^y^z$	$x^(y^z)$

## Logical operators

The logical operators are `and`, `or`, and `not`. Like control structures, all logical operators consider `false` and `nil` as false and anything else as true.

The operator `and` returns its first argument if it is false, otherwise it returns its second argument.

The operator `or` returns its first argument if it is not false; otherwise it returns its second argument:

```

print(4 and 5)
print(nil and 13)
print(false and 13)
print(4 or 5)
print(false or 5)

```

Output of code above:

```

5
nil
false
4
5

```

Both `and` and `or` use short-cut evaluation, that is, they evaluate their second operand only when necessary. A useful TSL construct is `x = x or v`, which is equivalent to:

```

if not x then x = v end

```



i.e., it sets `x` to a default value `v` when `x` is not set (provided that `x` is not set to `false`).

To select the maximum of two numbers `x` and `y`, use the following statement (note the `and` operator has a higher precedence than `or`):

```
max = (x > y) and x or y
```

When `x > y` is true, the first expression of the `and` is true, so the `and` results in its second argument `x` (which is also true, because it is a number), and then the `or` expression results in the value of its first expression, `x`. When `x > y` is false, the `and` expression is false and so the `or` results in its second expression, `y`.

The operator `not` always returns `true` or `false`:

```
print(not nil)
print(not false)
print(not 0)
print(not not nil)
```

Output of code above:

```
true
true
false
false
```

## Concatenation

TSL denotes the string concatenation operator by `..` (two dots). If any of its operands is a number, TSL converts that number to a string:

```
print("Hello " .. "World")
print(0 .. 1)
```

Output of code above:

```
Hello World
01
```

## Branching

TSL uses the “if” keyword to do conditional branching.

```
--
----- IF blocks -----
--
if 0 then                                -- Zero IS true! This is a contrast to C where
    print("Zero is true!")              -- 0 evaluates false. In TSL, “nil” is false
else                                      -- and everything else is true.
    print("Zero is false.")
end                                        -- if expression 1.

x = 1
y = 2
if (x and y) then
    print("' if ' expression 2 was not false.")
end                                        -- if expression 2.

if (x or y) then
    print("' if ' expression 3 was not false.")
end                                        -- if expression 3.

if (not x) then
    print("' if ' expression 4 was not false.")
else
    print("' if ' expression 4 was false.")
end                                        -- if expression 4.

if x == 10 then
    print("x = 10")
elseif y > 2 then
    print("y > 2")
else
    print("x is not equal to 10, and y is not less than 2.")
end                                        -- if expression 5.
```

Output of code above:

```
Zero is true!
' if ' expression 2 was not false.
' if ' expression 3 was not false.
' if ' expression 4 was false.
x is not equal to 10, and y is not less than 2.
```

## Loop control

TSL has familiar constructs for doing things repetitively and/or until an expression evaluates to false.

-- Something to iterate

```
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,")
```

```
print("in steps of two:")
```

```
for element = 1, 4, 2 do
```

```
    print(element, list[element])
```

```
end
```

```
--
```

```
----- WHILE loop -----
```

```
--
```

```
print("Count elements in list")
```

```
print("on numeric index")
```

```
element = 1
```

```
while list[element] do      -- Will exit when list[element] = nil
```

```
    print(element, list[element])
```

```
    element = element + 1
```

```
end
```

```
--
```

```
----- REPEAT loop -----
```

```
--
```

```
print("Count elements in list")
```

```
print("using repeat")
```

```
element = 1
```

```
repeat
```

```
    print(element, list[element])
```

```
    element = element + 1
```

```
until not list[element]
```

Output of code above:

```
Counting from one to three:
```

```
1 One
```

```
2 Two
```

```
3 Three
```

```

Counting from one to four,
in steps of two:
1 One
3 Three
Counting elements in list
on numeric index
1 One
2 Two
3 Three
4 Four
5 Five
6 Six
Counting elements in list
using repeat
1 One
2 Two
3 Three
4 Four
5 Five
6 Six

```

## Standard libraries

In addition to the standard programming constructs above, TSL includes standard libraries that contain useful functions for string manipulation, mathematics, etc. TSL also includes instrument control extension libraries. These libraries provide programming interfaces to the instrumentation accessible by the TSP. These libraries are automatically loaded when the TSP starts and do not need to be managed by the programmer.

### Base library functions

<code>print(x)</code>	Prints the argument <code>x</code> to the active host interface, using the <code>tostring()</code> function to convert <code>x</code> to a string.
<code>collectgarbage([limit])</code>	Sets the garbage-collection threshold to the given limit (in Kbytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, then TSL immediately runs the garbage collector. If the limit parameter is absent, it defaults to 0 (thus forcing a garbage-collection cycle). See <b>Note</b> for more information.
<code>gcinfo()</code>	Returns the number of Kbytes of dynamic memory that TSP is using.

<code>tonumber(x [,base])</code>	<p>Returns <code>x</code> converted to a number. If <code>x</code> is already a number, or a convertible string, then the number is returned; otherwise, it returns <code>nil</code>.</p> <p>An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter 'A' (in either upper or lower case) represents 10, 'B' represents 11, and so forth, with 'Z' representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.</p>
<code>tostring(x)</code>	Receives an argument of any type and converts it to a string in a reasonable format.
<code>type(v)</code>	Returns the type of its only argument, coded as a string. The possible results of this function are: "nil", "number", "boolean", "table", or "function".
<p><b>NOTE:</b> TSL does automatic memory management. That means that you do not have to worry about allocating memory for new objects and freeing it when the objects are no longer needed. TSL manages memory automatically by running a garbage collector from time to time to collect all dead objects (that is, those objects that are no longer accessible from TSL). All objects in TSL are subject to automatic management: tables, variables, functions, threads, and strings. TSL uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory TSL is using; the other is a threshold. When the number of bytes crosses the threshold, TSL runs the garbage collector, which reclaims the memory of all dead objects. The byte counter is adjusted, and then the threshold is reset to twice the new value of the byte counter.</p>	

## String library functions

This library provides generic functions for string manipulation, such as finding and extracting substrings. When indexing a string in TSL, the first character is at position 1 (not 0 as in ANSI C). Indices may be negative and are interpreted as indexing backwards, from the end of the string. Thus, the last character is at position 1, and so on.

<code>string.byte(s [,i])</code>	Returns the internal numerical code of the <code>i</code> -th character of string <code>s</code> , or <code>nil</code> if the index is out of range.
<code>string.char(i1, i1, ...)</code>	Receives 0 or more integers. Returns a string with length equal to the number of arguments, in which each character has the internal numerical code equal to its corresponding argument.
<code>string.format(fs, e1, e2, ...)</code>	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the <code>print</code> family of ANSI C functions. The only differences are that the options/modifiers <code>*</code> , <code>l</code> , <code>L</code> , <code>n</code> , <code>p</code> , and <code>h</code> are not supported. The options <code>c</code> , <code>d</code> , <code>E</code> , <code>e</code> , <code>f</code> , <code>g</code> , <code>G</code> , <code>l</code> , <code>o</code> , <code>u</code> , <code>X</code> , and <code>x</code> all expect a numeric argument, where <code>s</code> expects a string argument. String values to be formatted with <code>%s</code> cannot contain embedded zeros.
<code>string.len(s)</code>	Returns the length of the strings.

<code>string.lower(s)</code>	Returns a copy of the string <code>s</code> with all uppercase letters changed to lowercase.
<code>string.rep(s, n)</code>	Returns a string that is the concatenation of <code>n</code> copies of the string <code>s</code> .
<code>string.sub(s, i [,j])</code>	Returns the substring of <code>s</code> that starts at <code>i</code> and continues until <code>j</code> . <code>i</code> and <code>j</code> may be negative. If <code>j</code> is absent, then it is assumed to be equal to <code>-1</code> , which is the same as the string length. In particular, the call <code>string.sub(s, 1, j)</code> returns a prefix <code>s</code> with length <code>j</code> , and <code>string.sub(s, -i)</code> returns a suffix <code>s</code> with length <code>i</code> .
<code>string.upper(s)</code>	Returns a copy of the string <code>s</code> with all lowercase letters changed to uppercase.

## 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.

<code>math.abs(x)</code>	Returns the absolute value of the argument <code>x</code> .
<code>math.acos(x)</code>	Returns the principal value of the trigonometric arc cosine function of <code>x</code> .
<code>math.asin(x)</code>	Returns the principal value of the trigonometric arc sine function of <code>x</code> .
<code>math.atan(x)</code>	Returns the principal value of the trigonometric arc tangent function of <code>x</code> .
<code>math.atan2(y, x)</code>	Returns the principal value of the trigonometric arc tangent function of <code>y/x</code> .
<code>math.ceil(x)</code>	Returns the smallest floating-point number not less than <code>x</code> whose value is an exact mathematical integer.
<code>math.cos(x)</code>	Returns the trigonometric cosine function of <code>x</code> .
<code>math.deg(x)</code>	Returns the value of <code>x</code> in degrees, where <code>x</code> is in radians.
<code>math.exp(x)</code>	Returns the exponential function of <code>x</code> ; that is, $e^x$ , where $e$ is the base of the natural logarithms.
<code>math.floor(x)</code>	Returns the largest floating-point number not greater than <code>x</code> whose value is an exact mathematical integer.
<code>math.log(x)</code>	Returns the natural logarithm function of <code>x</code> .
<code>math.log10(x)</code>	Returns the base-10 logarithm function of <code>x</code> .
<code>math.max(x, y, ...)</code>	Returns the maximum value of its numeric argument(s).
<code>math.min(x, y, ...)</code>	Returns the minimum value of its argument(s).
<code>math.mod(x, y)</code>	Returns an approximation to the mathematical value $f$ such that $f$ has the same sign as <code>x</code> , the absolute value of $f$ is less than the absolute value of <code>y</code> , and there exists an integer $k$ such that $k*y+f = x$ .

<code>math.pi</code>	Variable containing the value of $\pi$ (3.141592654).
<code>math.pow(x, y)</code>	Returns $x^y$ .
<code>math.rad(x)</code>	Returns the value of $x$ in radians, where $x$ is in degrees.
<code>math.sin(x)</code>	Returns the trigonometric sine function of $x$ .
<code>math.sqrt(x)</code>	Returns the non-negative square root of $x$ .
<code>math.tan(x)</code>	Returns the trigonometric tangent function of $x$ .
<code>math.frexp()</code>	Splits $x$ into a fraction $f$ and exponent $n$ , such that $f$ is 0.0 or $0.5 \leq  f  \leq 1.0$ , and $f * 2^n$ is equal to $x$ . Both $f$ and $n$ are returned; $f, n = \text{math.frexp}(x)$ .
<code>math.ldexp(x, n)</code>	Returns the inverse of the <code>math.frexp()</code> function; it computes the value $x * 2^n$ .
<code>math.random([x], [y])</code>	When called without an argument, returns a pseudo-random real number in the range $[0, 1)$ . When called with number $x$ , returns a pseudo-random integer in the range $[1, n]$ . When called with two arguments, $x$ and $y$ , returns a pseudo-random integer in the range $[x, y]$ .
<code>math.randomseed(x)</code>	Sets a "seed" for the pseudo-random generator. Equal seeds produce equal sequences of numbers.

# 3

## DUT Test Connections

---

### Section 3 topics

**Input/Output terminal blocks**, page 3-2

**Input/Output LO and chassis ground**, page 3-3

**Sensing methods**, page 3-5

2-wire local sensing, page 3-5

4-wire remote sensing, page 3-5

Sense mode selection, page 3-6

**Multiple SMU connections**, page 3-7

**Guarding and shielding**, page 3-8

Guarding, page 3-8

Noise shield, page 3-9

Safety shield, page 3-10

Using shielding and guarding together, page 3-11

**Test fixture**, page 3-12

**Floating an SMU**, page 3-13

**Output-off states**, page 3-15



## Input/Output terminal blocks

The Model 260x uses screw terminal blocks for input and output connections to DUTs (Devices Under Test). The Model 2602 uses two terminal blocks as shown in [Figure 3-1](#) (one for each SMU channel). The Model 2601 has only one terminal block for the single SMU.

A terminal block can be removed from the rear panel by loosening the two captive retaining screws and pulling it off the rear panel. Each screw terminal can accommodate from 24 AWG (0.2mm<sup>2</sup>) to 12 AWG (2.5mm<sup>2</sup>) conductors.

After making the wire connections from a terminal block to a DUT, re-install the terminal block onto the rear panel connector and tighten the two captive screws.

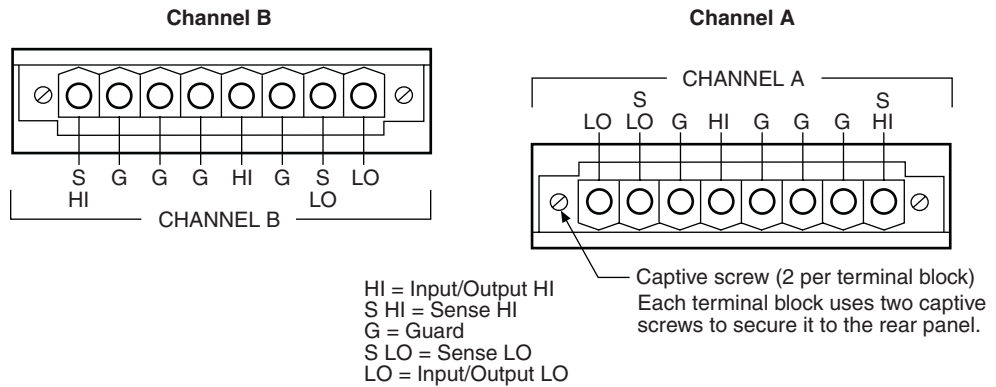
**WARNING** **Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, NEVER make or break connections to the Model 260x while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the SourceMeter before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.**

**Maximum floating (common mode) voltage for an SMU is 250V. Exceeding this level could damage the instrument and create a shock hazard. See “[Floating an SMU](#)” on [page 3-14](#) for details on floating the SMUs.**

**The Input/Output terminals of the SourceMeters are rated for connection to circuits rated Installation Category I only. Do not connect the SourceMeter terminals to CAT II, CAT III, or CAT IV circuits. Connections of the SourceMeter Input/Output terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.**

**To prevent electric shock and/or damage to the SourceMeter, when connecting to a source with a greater current capability than the Model 260x, a fuse should be provided in line with the SourceMeter Input/Output terminals rated no more than 3A.**

Figure 3-1  
Input/output terminal blocks



## Input/Output LO and chassis ground

As shown in [Figure 3-2](#), SMU input/output LOs are available at the rear panel terminal blocks. Input/Output LOs are not connected between channels and are electrically isolated from chassis ground.

As shown, there is a low-noise chassis ground banana jack that can be used as a common signal ground point for Input/Output LOs. This low-noise signal ground banana jack is connected to the chassis through a Frequency Variable Resistor (FVR).

The FVR (see [Figure 3-3](#)) is used to isolate the SMUs from high frequencies that may be present on the chassis of the Model 260x. As frequencies on the chassis increase, the resistance of the FVR increases to dampen its effects.

### NOTE

Keep in mind that the chassis should never be used as a ground point for signal connections. High frequencies present on the chassis of the Model 260x may result in higher noise. The chassis should only be used as a safety shield. Use the chassis screw for connections to the chassis of the Model 260x.

Figure 3-2  
**Input/Output LO and chassis ground terminals**

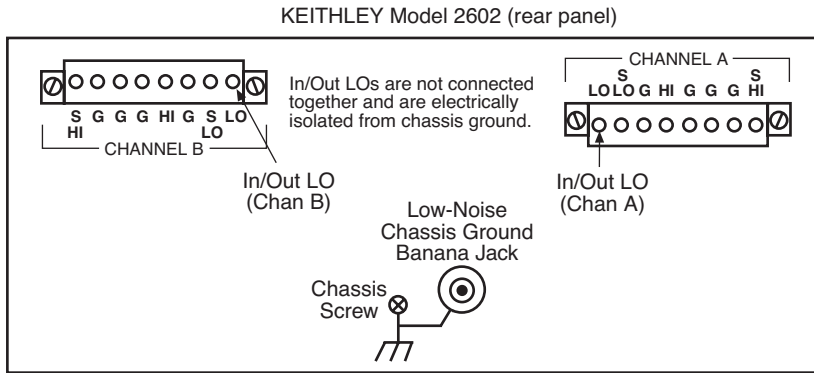
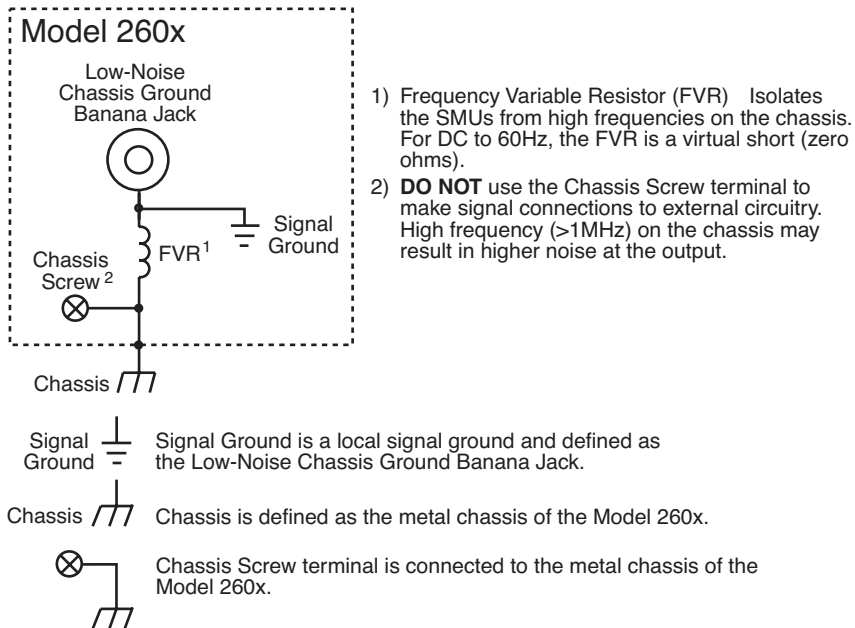


Figure 3-3  
**Low-Noise Chassis Ground Banana Jack and Chassis Screw**



## Sensing methods

Source-measure operations are performed using either 2-wire local sense connections or 4-wire remote sense connections.

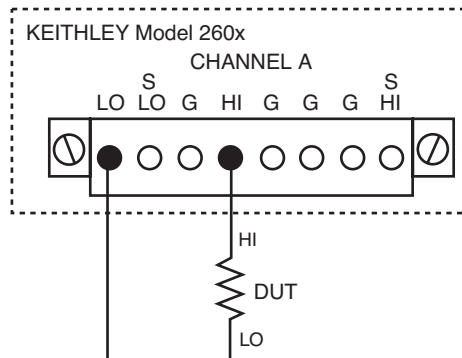
**NOTE** The default sense setting is 2-wire local. See “[Sense mode selection](#)” on [page 3-6](#) to check and or change the sense mode.

### 2-wire local sensing

Two-wire local sensing (as shown in [Figure 3-4](#)) can be used for the following source-measure conditions:

- Sourcing and measuring current.
- Sourcing and/or measuring voltage in high impedance ( $>1\text{k}\Omega$ ) test circuits.

Figure 3-4  
2-wire connections (local sense)



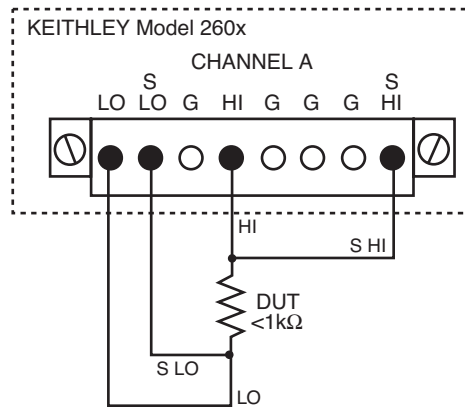
### 4-wire remote sensing

When sourcing and/or measuring voltage in a low-impedance test circuit (see [Figure 3-5](#)), there can be errors associated with IR drops in the test leads. Voltage source and measure accuracy are optimized by using 4-wire remote sense connections. When sourcing voltage, 4-wire remote sensing ensures that the programmed voltage is delivered to the DUT. When measuring voltage, only the voltage drop across the DUT is measured.

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

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

Figure 3-5  
**4-wire connections (remote sense)**



## Sense mode selection

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

### Front panel sense selection

Figure 3-1 summarizes the steps to check and/or change the sense mode front panel. When in the menu structure, use the Rotary Wheel (or **CURSOR** keys) to position the blinking cursor on the desired menu item, and press **ENTER** to select it. Use the **EXIT** key to back out of the menu structure.

Table 3-1

### Selecting the sense mode from the front panel

Model 2601	Model 2602
1) Press <b>CONFIG</b> key	1) Press <b>CONFIG</b> key
2) Select <b>SRC</b> or <b>MEAS</b> menu*	2) Select <b>CHANNEL-A</b> or <b>CHANNEL-B</b>
3) Select <b>V-SOURCE</b> menu	3) Select <b>SRC</b> or <b>MEAS</b> menu*
4) Select <b>SENSE-MODE</b> menu	4) Select <b>V-SOURCE</b> menu
5) Select <b>2-WIRE</b> or <b>4-WIRE</b>	5) Select <b>SENSE-MODE</b> menu
	6) Select <b>2-WIRE</b> or <b>4-WIRE</b>

\* The sense mode can be set from either the **SRC** or **MEAS** menu.

## Remote programming sense selection

Table 3-2 summarizes the commands to select the sense mode. See [Section 12](#) for details on using these commands.

Table 3-2  
Commands to select sense mode

Command*	Description
<code>smuX.source.output = smuX.OUTPUT_OFF</code>	Turns off the SMU output.
<code>smuX.sense = smuX.SENSE_LOCAL</code>	Selects local (2-wire) sense.
<code>smuX.sense = smuX.SENSE_REMOTE</code>	Selects remote (4-wire) sense.

\* Model 2601: smuX = smua    Model 2602: smuX = smua (Channel A) or smub (Channel B).

## Multiple SMU connections

Figure 3-6 shows how to use two SMUs to test a 3-terminal device, such as an N-channel JFET. A typical application is for SMU B to source a range of gate voltages, while SMU A sources voltage to power the device and measures current at each gate voltage.

Figure 3-6  
Two SMUs connected to a 3-terminal device (local sense)

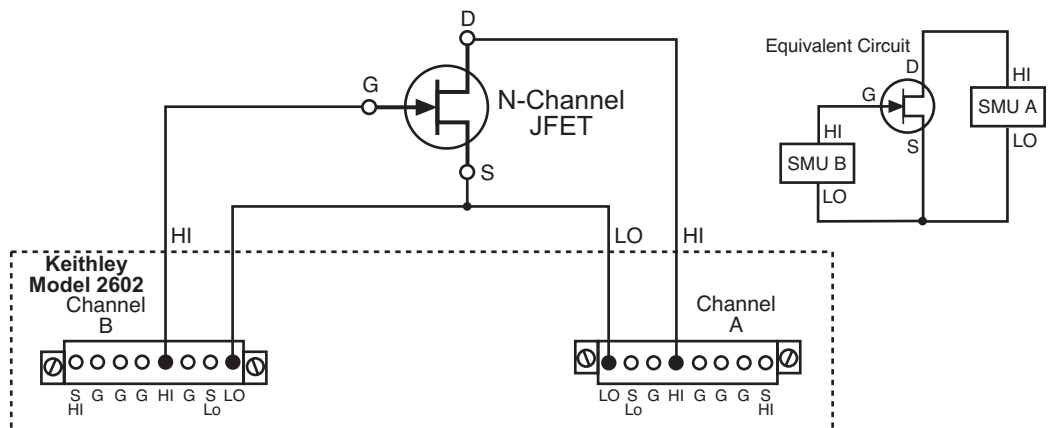
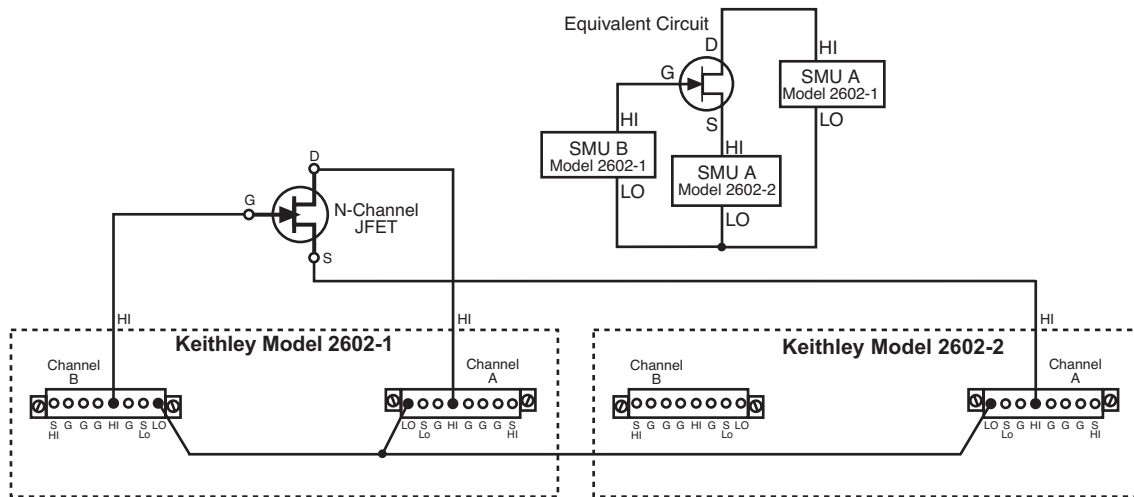


Figure 3-7 shows how to use three SMUs to test the same 3-terminal device. The third SMU is connected to the source (S) terminal of the JFET. This allows the source terminal to be biased above signal low. Setting this SMU to output 0V effectively connects the source terminal of the JFET to signal low.

Figure 3-7  
 Three SMUs connected to a 3-terminal device (local sense)



## Guarding and shielding

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

### Guarding

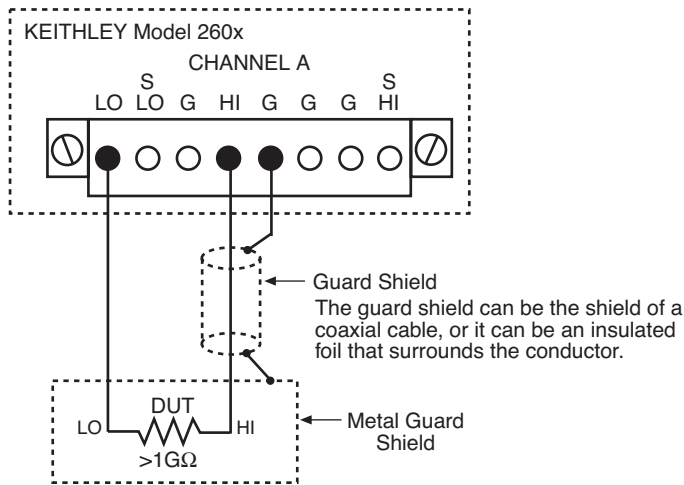
A driven guard is always enabled and provides a buffered voltage that is at the same level as the Input/Output HI voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between Input/Output high and low. Without guarding, leakage and capacitance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

Guarding (shown in [Figure 3-8](#)) should be used for the following source-measure condition:

- Test circuit impedance is  $>1\text{G}\Omega$

**NOTE** See “[Guard](#)” in [Section 8](#) for details on the principles of guarding.

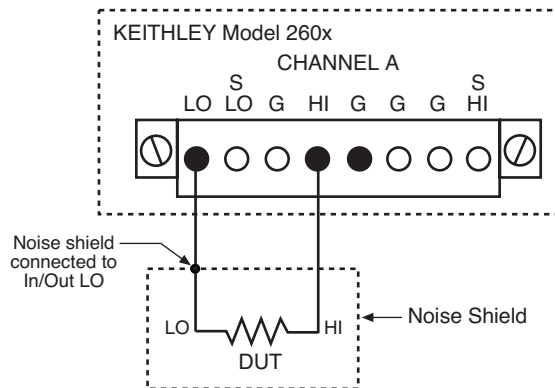
Figure 3-8  
**High-impedance guarding**



### Noise shield

A noise shield (see Figure 3-9) is used to prevent unwanted signals from being induced into the test circuit. Low level signals may benefit from effective shielding. The metal noise shield surrounds the test circuit and should be connected to SMU LO (as shown in Figure 3-9), and can also be connected to the chassis ground screw on the rear panel of the Model 260x.

Figure 3-9  
**Noise shield**





## Safety shield

A safety shield must be used whenever hazardous voltages ( $>30V_{rms}$ ,  $42V_{peak}$ ) will be present in the test circuit. The safety shield can be metallic or nonmetallic, and must completely surround the DUT test circuit. A metal safety must be connected to a known safety earth ground and chassis ground. (See “[Test fixture](#)” on [page 3-12](#) for important safety information on the use of a metal or nonmetallic enclosure.)

The maximum output voltage for a Model 260x SMU is 40V, which is considered a non-hazardous level. However, using two or more voltage sources in a series configuration can cause test circuit voltage to exceed 42V. For example, the SMUs of two Model 260x instruments can be connected in series to apply 80V to a DUT (see [Figure 3-10](#)).

The connections for the test configuration in [Figure 3-10](#) are shown in [Figure 3-11](#). Use #18AWG wire or larger for connections to safety earth ground and chassis.

**NOTE** Floating an SMU may also cause test circuit voltage to exceed 42V (see “[Floating an SMU](#)” on [page 3-14](#)).

Figure 3-10  
Safety shield for hazardous voltage ( $>42V$ )

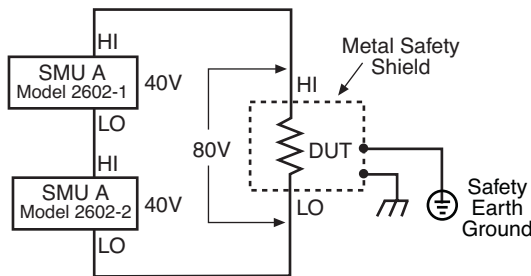
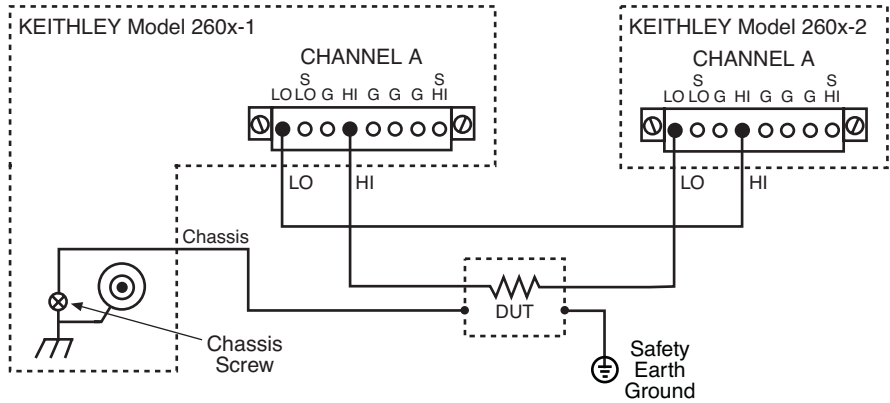


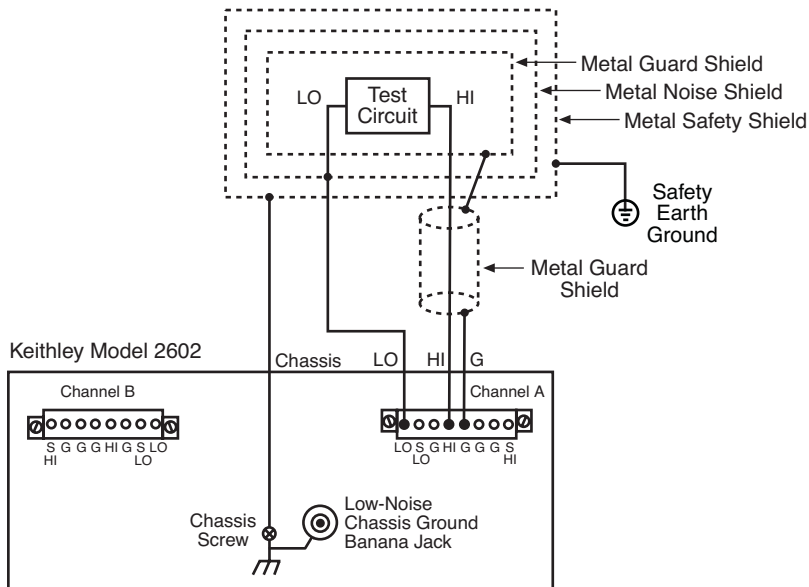
Figure 3-11  
**Connections for test circuit shown in Figure 3-10**



### Using shielding and guarding together

Figure 3-12 shows connections for a test system that uses a noise shield, a safety shield, and guarding. The guard shields are connected to the driven guard (G) of the SMU. The noise shield is connected to SMU LO. The safety shield is connected to the chassis and to a safety earth ground.

Figure 3-12  
**Connections for noise shield, safety shield and guarding**



## Test fixture

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

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

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

**For metallic enclosures, the test fixture chassis must be properly connected to safety earth ground. A grounding wire (#18 AWG or larger) must be attached securely to the test fixture at a screw terminal designed for safety grounding. The other end of the ground wire must be attached to a known safety Earth Ground.**

**Construction material** – A metal test fixture must be connected to a known safety Earth Ground as described in the above **WARNING**. A nonmetallic test fixture must be constructed of materials that are suitable for flammability, voltage and temperature conditions that may exist in the test circuit. The construction requirements for a nonmetallic enclosure are also described in the **WARNING**.

**Test circuit isolation** – With the lid closed, the test fixture must completely surround the test circuit. A metal test fixture must be electrically isolated from the test circuit. Input/output connectors mounted on a metal test fixture must also be isolated from the test fixture. Internally, teflon standoffs are typically used to insulate the internal pc-board or guard plate for the test circuit from a metal test fixture.

**Interlock switch** – The test fixture must have a normally-open interlock switch. The interlock switch must be installed so that when the lid of the test fixture is opened, the switch will open, and when the lid is closed, the switch will close.

**WARNING** When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages. The Digital I/O port of Model 2600 Series SourceMeter is not suitable for control of safety circuits and should not be used to control a safety interlock.

## Floating an SMU

Using an external source in the test system may require that a Model 260x SMU float off chassis earth ground. An example of such a test system is shown in [Figure 3-13](#), which includes an external voltage source. Notice that output low of the voltage source is connected to chassis earth ground.

For the test circuit shown in [Figure 3-13](#), the Model 260x must float off chassis earth ground. As shown, SMU LO of the Model 622x is floating +10V above chassis earth ground. If SMU LO of the Model 622x was instead connected to chassis ground, the external voltage source would be shorted through chassis ground.

The Model 260x connections for the floating configuration ([Figure 3-13](#)) are shown in [Figure 3-14](#). In order to float the SMU, Input/Output LO must be isolated from chassis ground. This is accomplished by NOT connecting Input/Output LO to chassis ground.

The external voltage source in [Figure 3-13](#) and [Figure 3-14](#) can instead be an SMU of a second Model 260x instrument. Keep in mind that if the combined outputs of the sources exceeds 42V, then a safety shield will be required for the DUT (see the following WARNINGS).

**WARNING** The maximum floating (common mode) voltage for an SMU is  $\pm 250\text{V}$ . Exceeding this level may cause damage to the instrument and create a shock hazard.

Using an external source to float an SMU could create a shock hazard in the test circuit. A shock hazard exists whenever  $>42\text{V}$  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  $>42\text{V}$  is present, the test circuit must be insulated for the voltage used or surrounded by a metal safety shield that is connected to a known safety earth ground and chassis ground (see “[Safety shield](#)” on [page 3-10](#)).

Figure 3-13  
**Floating the Model 260x**

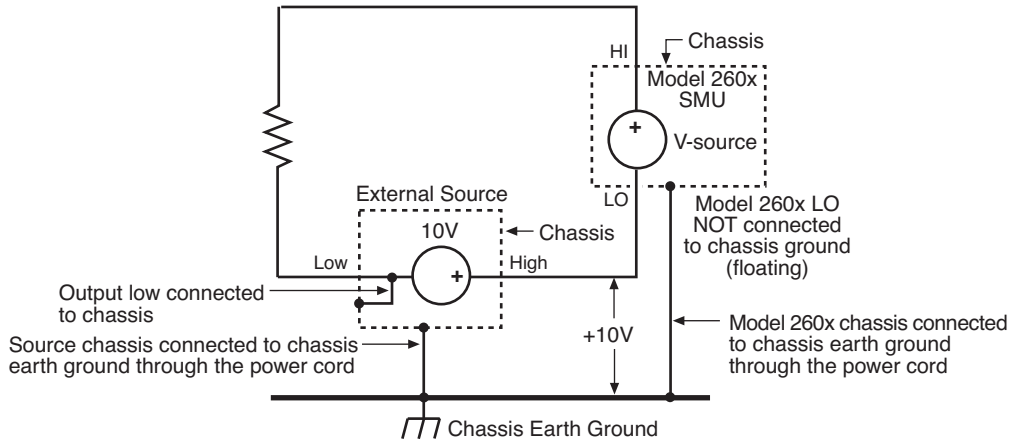
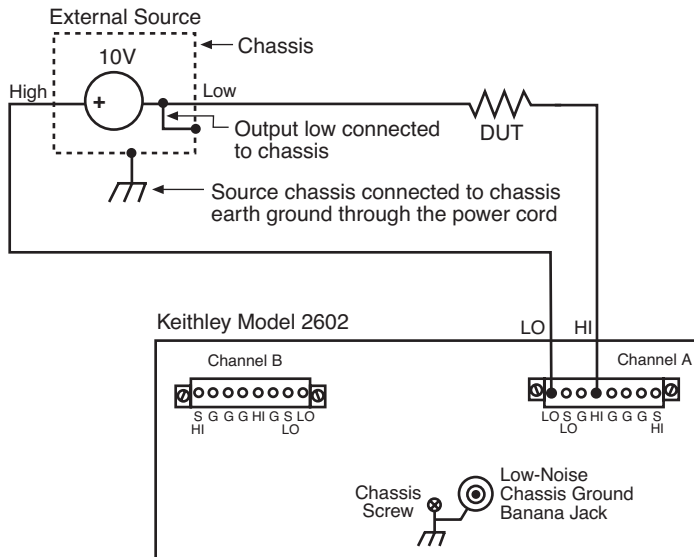


Figure 3-14  
**SMU connections for the floating configuration shown in Figure 3-13**



# Output-off states

When an SMU is turned off, it may not be completely isolated from the external circuit that is connected to. There are three output-off states for a Model 260x SMU: Normal, High Impedance or Zero. For the Model 2602, each SMU can have its own unique output-off state.

## Normal output-off state

For the normal output-off state (which is the default setting), the SMU will source 0V. Current compliance will set to 10% full scale of the present current range, or 100 $\mu$ A (whichever is smaller). In theory, with the SMU set to source 0V, the SMU will not source or sink power. In practice, the source value may not be exactly at zero. Therefore, the SMU may source or sink a very small amount of power. In most cases, this source or sink power level is not significant.

## High-impedance output-off state

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

## Zero output-off state

When in this output-off state, the SourceMeter is configured as follows:

When the V-Source is the selected source:

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

When the I-Source is the selected source:

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

While in the zero output-off state, the SourceMeter can be used as an I-Meter since it will output 0V, but measure current.

## Selecting the Output-off state

### Output-off state menu

The OUTPUT configuration menu can be accessed by pressing the CONFIG key and then the appropriate OUTPUT ON/OFF key. In the configuration menu, select OFF STATE to display the OUTPUT OFF STATE menu.

**NOTE**     **The OUTPUT OFF STATE menu can also be accessed by navigating the configuration menu that is displayed by pressing the CONFIG key.**

With the OUTPUT OFF STATE menu displayed, select the desired output-off state: HI-Z (high-impedance), NORMAL or ZERO.

### Remote programming

[Table 3-3](#) lists the commands to select the Output-off state.

Table 3-3

**Commands to select Output-off state**

Command*	Description
smuX.source.offmode = smuX.OUTPUT_NORMAL	Selects normal Output-off state.
smuX.source.offmode = smuX.OUTPUT_HIGH_Z	Selects high-impedance Output-off state.
smuX.source.offmode = smuX.OUTPUT_ZERO	Selects zero Output-off state.

\* Model 2601: smuX = smua     Model 2602: smuX = smua (Channel A) or smub (Channel B).

# 4

# Basic Operation

---

## Section 4 topics

**Overview**, page 4-2

**Operation overview**, page 4-2

Source-measure capabilities, page 4-2

Compliance limit, page 4-3

Setting the compliance limit, page 4-4

Basic circuit configurations, page 4-5

**Operation considerations**, page 4-7

Warm-up, page 4-7

Auto zero, page 4-7

NPLC caching, page 4-7

**Triggering**, page 4-9

Triggering types, page 4-9

Measurement triggering, page 4-10

Front panel triggering, page 4-11

Remote triggering, page 4-12

**Basic source-measure procedure**, page 4-13

Front panel source-measure procedure,  
page 4-13

Remote source-measure procedure, page 4-14

**Measure only**, page 4-16

**Sink operation**, page 4-17

**Ohms measurements**, page 4-18

Ohms calculations, page 4-18

Ohms ranging, page 4-18

Basic ohms measurement procedure, page 4-18

Ohms sensing, page 4-19

Sense selection, page 4-20

Remote ohms programming, page 4-21

**Power measurements**, page 4-22

Power calculations, page 4-22

Basic power measurement procedure, page 4-22

Remote power programming, page 4-23



# Overview

The documentation in this section provides detailed information on using the buffer to store data and includes the following:

- “Operation overview”, page 4-2
- “Operation considerations”, page 4-7
- “Triggering”, page 4-9
- “Measure only”, page 4-16
- “Sink operation”, page 4-17
- “Ohms measurements”, page 4-18
- “Power measurements”, page 4-22

## Operation overview

### Source-measure capabilities

From the front panel, the SourceMeter can be configured to perform the following operations:

- **Source voltage** — Display current and/or voltage measurement.
- **Source current** — Display voltage and/or current measurement.
- **Measure resistance** — Display resistance calculated from voltage and current components of measurement.
- **Measure power** — Display power calculated from voltage and current components of measurement.
- **Measure only (V or I)** — Display voltage or current measurement.

### Voltage and current

[Table 4-1](#) lists the source and measure limits for the voltage and current functions. The full range of operation is explained in “[Operating boundaries](#)” on [page 8-7](#).

**NOTE** See the specifications in [Appendix A](#) for important performance aspects that may affect measurements.

Table 4-1  
**Source-measure capabilities**

Range	Source	Measure
100mV	±101mV	±102mV
1V	±1.01V	±1.02V
6V	±6.06V	±6.12V
40V	±40.4V	±40.8V
100nA	±101nA	±102nA
1µA	±1.01µA	±1.02µA
10µA	±10.1µA	±10.2µA
100µA	±101µA	±102µA
1mA	±1.01mA	±1.02mA
10mA	±10.1mA	±10.2mA
100mA	±101mA	±102mA
1A	±1.01A	±1.02A
3A	±3.03A	±3.06A
Max Power = 40.4W per channel		

## Compliance limit

When sourcing voltage, the SourceMeter can be set to limit current. Conversely, when sourcing current, the SourceMeter can be set to limit voltage. The SourceMeter output will not exceed the compliance limit. The maximum compliance limit is the same as the maximum values listed in [Table 4-2](#). Note that the compliance value will take the same sign as the source value, and the maximum compliance limits are based on source range. See [“Compliance limit” on page 8-2](#) for more information. The only exception to the compliance limit not being exceeded is the VLIMIT when operating as an ISOURCE. To avoid excessive (and potentially destructive) currents from flowing, the VLIMIT will source or sink up to 102mA for ISOURCE ranges on or below 100mA. For the ranges 1A and above, the maximum current allowed is the current source setting.

Table 4-2  
**Maximum compliance values**

<b>Source range</b>	<b>Maximum compliance value</b>
100mV	3A
1V	3A
6V	3A
40V	1A
100nA	40V
1μA	40V
10μA	40V
100μA	40V
1mA	40V
10mA	40V
100mA	40V
1A	40V
3A	6V

## Setting the compliance limit

### Front panel compliance limit

Set the compliance limit from the front panel as follows:

1. For the Model 2601 or the Model 2602 single-channel display mode, press the LIMIT key to directly access compliance editing.
2. For the Model 2602 dual-channel display mode, press CONFIG then LIMIT, then select CURRENT or VOLTAGE as desired. Press ENTER or the Rotary Knob.
3. Press the Rotary Knob, set the compliance limit to the desired value, and then press ENTER or the Rotary Knob to complete editing.
4. Press EXIT to return to the normal display.

## Remote compliance limit

[Table 4-3](#) summarizes basic commands to program the compliance limit. See [Section 12](#) for more details on these commands. To program the compliance, simply send the command using the desired parameter. For example, the following commands set the current and voltage compliance to 50mA and 4V:

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

The following command will print the compliance state:

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

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

Table 4-3

### Compliance commands

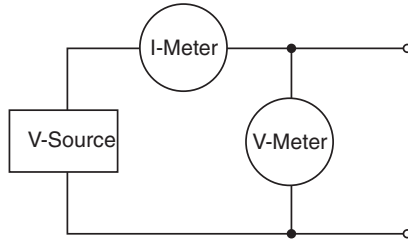
Command <sup>1</sup>	Description
smuX.source.limiti = limit smuX.source.limitv = limit compliance = smuX.source.compliance	Set current compliance limit. Set voltage compliance limit. Test if in compliance (1 = in compliance; 0 = not in compliance).

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

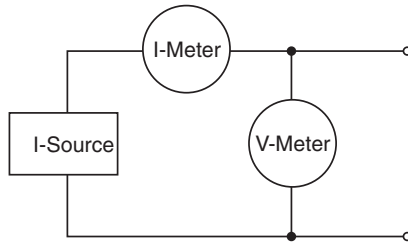
## Basic circuit configurations

The fundamental source-measure configurations for the SourceMeter are shown in [Figure 4-1](#). When sourcing voltage, you can measure current or voltage (configuration A). When sourcing current, you can measure voltage or current (configuration B). See [“Basic circuit configurations” on page 8-16](#) for more detailed information on these circuit configurations.

Figure 4-1  
**Fundamental source  
measure configuration**



**A. Source V**



**B. Source I**

# Operation considerations

The following paragraphs discuss the warm-up period and auto zero.

## Warm-up

The SourceMeter must be turned on and allowed to warm up for at least two hours to achieve rated accuracies. See Appendix A for specifications.

## Auto zero

The Model 260x SourceMeters use a ratiometric A/D conversion technique. To ensure accuracy of readings, the instrument must periodically obtain fresh measurements of its internal ground and voltage reference. The time interval between needing to update these reference measurements is determined by the integration aperture being used for measurements. Separate reference and zero measurements are used for each aperture.

There are three different settings for auto zero as summarized in [Table 4-4](#). By default, the instrument automatically checks these reference measurements whenever a signal measurement is made (AUTO). If the reference measurements are out of date when a signal measurement is made, the instrument will automatically take two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes longer than normal.

This extra time can cause problems in sweeps and other test sequences in which measurement timing is critical. To avoid the extra time for the reference measurements in these situations, the OFF selection can be used to disable the automatic reference measurements. Keep in mind that with automatic reference measurements disabled, the instrument may gradually drift out of specification.

To minimize the drift, a reference and zero measurement should be made just prior to the critical test sequence. The ONCE setting can be used to force a refresh of the reference and zero measurements used for the current aperture setting.

## NPLC caching

NPLC caching speeds up operation by caching A/D reference and zero values for up to the five most recent measurement function settings. Whenever the integration rate is changed via the SPEED key, or a user setup is recalled, NPLC caching will occur. If the integration rate is already stored in the cache, the stored reference and zero values are recalled and used. Otherwise, a reference and zero value are acquired and stored in the cache. If there are already five NPLC values stored, the oldest one will be overwritten by the newest one. When auto zero is off,

NPLC values stored in the cache will be used regardless of how old they are. If there are no entries in the cache for the aperture being used, the unit will acquire them when the first measurement is made.

Table 4-4

**Auto zero settings**

<b>Auto zero setting</b>	<b>Description</b>
OFF	Turns automatic reference measurements off.
ONCE	Turns automatic reference measurements on, forcing one reference and one zero measurement.
AUTO	Automatically takes new acquisitions when processor determines reference and zero values are out-of-date.

**Front panel auto zero**

Set the auto zero from the front panel as follows:

1. Press the CONFIG key, and select MEAS from the menu.
2. Select AUTO-ZERO, then press ENTER or the Rotary Knob.
3. Select the desired mode (OFF, ONCE, or AUTO), and then press ENTER or the Rotary Knob.
4. Press EXIT as necessary to return to the normal display.

**Remote command auto zero**

Use the auto zero command with the appropriate option shown in [Table 4-5](#) to set auto zero via remote. (See [Section 12](#) for more details). For example, send the following command to turn automatic reference measurements on:

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

Table 4-5

**Auto zero command and options**

Command <sup>1</sup>	Description
smuX.measure.autozero = smuX.AUTOZERO_OFF	Disable auto zero. <sup>2</sup>
smuX.measure.autozero = smuX.AUTOZERO_ONCE	Force one ref and zero.
smuX.measure.autozero = smuX.AUTOZERO_AUTO	Force ref and zero with each measurement.

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

2. Old NPLC cache values will be used when auto zero is disabled. See “NPLC caching” on page 4-7.

# Triggering

## Triggering types

A trigger initiates an event within the Model 260x SourceMeter. In general, there are three types of triggering:

- **Measurement triggering** — Used to initiate one or more measurements, to control the time interval between measurements or the trigger and measurement, and to set the number of measurements per trigger. See [Measurement triggering](#) below for details.
- **Digital I/O port triggering** — Used to trigger external devices with pulses from the Digital I/O port, and to trigger the Model 260x from external control pulses applied to the Digital I/O port (see [Section 10](#) for details).
- **Display triggering** — Used to trigger events with front panel keys (see [Section 14](#)).

**NOTE** It is not necessary to change any trigger settings to use the basic source and measurement procedures covered in this section. Simply make sure that the unit is reset to the factory default conditions by using the MENU > SAVE-SETUP > RECALL > FACTORY option before using those procedures.

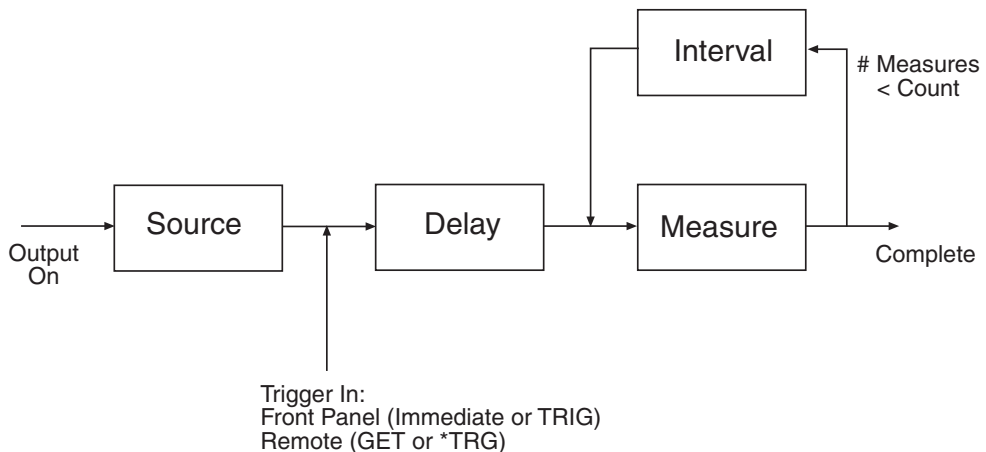


## Measurement triggering

Figure 4-2 shows the general sequence for measurement triggering. The basic sequence is as follows:

- When the output is turned on, the programmed source value is immediately applied to the DUT.
- For front panel operation, if the immediate trigger source is selected, a measurement will be triggered immediately. However, if the manual trigger source is selected, the front panel TRIG key must be pressed.
- For remote operation, the unit waits the programmed timeout period for a GET (GPIB only) or \*TRG (both interfaces) trigger command.
- The unit waits for the programmed delay period (if any).
- The instrument takes one measurement.
- If the number of measurements is less than the programmed trigger count, the unit cycles back to take another measurement (the measurement cycle will be repeated indefinitely if the infinite trigger count is selected).
- For multiple measurements, the unit waits for the programmed trigger interval (if any) before taking the next measurement.

Figure 4-2  
Measurement triggering sequence



## Front panel triggering

To control triggering from the front panel, press CONFIG followed by TRIG, then set up trigger parameters as described below:

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

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

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

- **FINITE:** The unit will cycle through measurement cycles for the programmed trigger count (1 to 99999).
- **INFINITE:** The unit will cycle through measurement cycles indefinitely until halted.

**INTERVAL** — Sets the time interval between measurements (0s to 999.999s) when the COUNT is greater than 1.

**DELAY** — Sets the delay period between the trigger and the start of measurement (0s to 999.999s).

### Front panel triggering example

As an example, assume you wish to set up triggering as follows:

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

Set up this example as follows:

1. Press CONFIG then TRIG.
2. Select TRIGGER-IN, then press ENTER or the Rotary Knob. Select MANUAL, then press ENTER or the Rotary Knob.
3. Choose COUNT, then select INFINITE, and press ENTER or the knob.
4. Select INTERVAL, set the interval to 1s, then press ENTER or the knob.
5. Choose DELAY, set the delay to 2s, then press ENTER.
6. Press EXIT to return to normal display.
7. Turn on the OUTPUT ON, then press TRIG. A two second delay will occur before the first measurement. The unit will cycle through measurements indefinitely with a 1s interval between measurements.
8. Turn the OUTPUT off to stop taking readings.

## Remote triggering

### Remote trigger commands

Trigger commands are listed in [Table 4-6](#). See Section 12 for more details.

The trigger event detector remembers if an event has been detected since the last `trigger.wait` call. The `trigger.clear()` command clears the trigger's event detector and discards the previous history of command interface trigger events.

The `trigger.wait` function will wait `timeout` seconds for a GPIB GET command (see [Section 11](#)) or a \*TRG message (see [Appendix C](#)) on the GPIB interface if that is the active command interface, or a \*TRG message on the command interface for all other interfaces. If one or more of these trigger events were previously detected, this function will return immediately. After waiting for a trigger with this function, the event detector will be automatically reset and re-armed regardless of the number of events detected.

Table 4-6

#### Trigger commands

Command	Description
<code>trigger.clear()</code>	Clear pending triggers.
<code>triggered = trigger.wait(timeout)</code>	Wait <code>timeout</code> seconds for GET or *TRG trigger.

### Remote trigger example

The example below clears triggers and then enables a 30 second timeout to wait for a GET or \*TRG trigger.

```
smua.reset()                --Restore 260x defaults.
triggered = trigger.wait(30) --Wait 30s for GET or *TRG.
smua.source.output =smua.OUTPUT_ON --Turn on output.
*TRG                        --Trigger reading.
reading = smua.measure.v()  --Get voltage reading.
```

# Basic source-measure procedure

## Front panel source-measure procedure

Use the following procedure to perform the basic source-measure operations of the Model 260x SourceMeter. The following procedure assumes that the SourceMeter is already connected to the DUT as explained in Section 3.

**WARNING** Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, NEVER make or break connections to the Model 260x while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the SourceMeter before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.

### Step 1: Select and set source level.

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

1. Press SRC as needed to select the V-Source or I-Source as indicated by the units in the source field on the display. The flashing digit (cursor) indicates which value is presently selected for editing.
2. Move the cursor to the digit to change, then press the Rotary Knob to enter the EDIT mode, as indicated by the EDIT annunciator.
3. Use the RANGE keys to select a range that will accommodate the value you want to set. (See [Section 6](#) for range information.) For best accuracy, use the lowest possible source range.
4. Enter the desired source value, then press ENTER or the Rotary Knob to complete editing.

### Step 2: Set compliance limit.

Perform the following steps to edit the compliance limit value:

1. For the Model 2601 or the Model 2602 single-channel display mode, press the LIMIT key.
2. For the Model 2602 dual-channel display mode, press CONFIG then LIMIT, then select CURRENT or VOLTAGE. Press ENTER or the Rotary Knob.
3. Move the cursor to the digit to change, then press the Rotary Knob to enter the EDIT mode, as indicated by the EDIT annunciator.
4. Enter the desired limit value, then press ENTER or the Rotary Knob to complete editing.

### **Step 3: Select measurement function and range.**

Select measurement function and range as follows:

1. Put the Model 2602 in the single-channel display mode, then select the desired measurement function by pressing MEAS or MODE.
2. Select the desired measurement range with the RANGE keys, or enable AUTO RANGE, keeping the following points in mind:
  - When measuring the source (i.e., Source V Measure V), you cannot select the measurement range using the RANGE keys. The selected source range determines the measurement range.
  - When not measuring the source (i.e., Source V Measure I), measurement range selection can be done manually or automatically. When using manual ranging, use the lowest possible range for best accuracy. In auto range, the SourceMeter automatically goes to the most sensitive range to make the measurement.

### **Step 4: Turn output on.**

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

### **Step 5: Observe readings on the display.**

Observe the readings on the display. Press TRIG if necessary to trigger the unit to begin taking readings. For the single-channel display mode, the readings will appear on the top line, while source and limit values are on the bottom line.

### **Step 6: Turn output off.**

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

## **Remote source-measure procedure**

Basic source-measurement procedures can also be performed via remote by sending appropriate commands in the right sequence. The following paragraphs summarize the basic commands and give a simple programming example.

## Basic source-measure commands

Table 4-7 summarizes basic source-measure commands. See [Section 12](#) for more information on using these commands.

Table 4-7

**Basic source-measure commands**

Command <sup>1</sup>	Description
smuX.measure.autorangei = smuX.AUTORANGE_ON smuX.measure.autorangev = smuX.AUTORANGE_ON smuX.measure.autorangei = smuX.AUTORANGE_OFF smuX.measure.autorangev = smuX.AUTORANGE_OFF smuX.measure.rangei = rangeval smuX.measure.rangev = rangeval reading = smuX.measure.i() reading = smuX.measure.v() reading = smuX.measure.iv() reading = smuX.measure.r() reading = smuX.measure.p()	Enable current measure auto range. Enable voltage measure auto range. Disable current measure auto range. Disable voltage measure auto range. Set current measure range. Set voltage measure range. Request a current reading. Request a voltage reading. Request a current and voltage reading. Request a resistance reading. Request a power reading.
smuX.source.autorangei = smuX.AUTORANGE_ON smuX.source.autorangev = smuX.AUTORANGE_ON smuX.source.autorangei = smuX.AUTORANGE_OFF smuX.source.autorangev = smuX.AUTORANGE_OFF smuX.source.func = smuX.OUTPUT_DCVOLTS smuX.source.func = smuX.OUTPUT_DCAMPS smuX.source.leveli = sourceval smuX.source.levelv = sourceval smuX.source.limiti = level smuX.source.limitv = level smuX.source.output = smuX.OUTPUT_ON smuX.source.output = smuX.OUTPUT_OFF smuX.source.rangei = rangeval smuX.source.rangev = rangeval smuX.sense = smuX.SENSE_LOCAL smuX.sense = smuX.SENSE_REMOTE	Enable current source auto range. Enable voltage source auto range. Disable current source auto range. Disable voltage source auto range. Select voltage source function. Select current source function. Set current source value. Set voltage source value. Set current limit. Set voltage limit. Turn on source output. Turn off source output. Set current source range. Set voltage source range. Local sense (2-wire). Remote sense (4-wire).

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

## Requesting readings

You can request readings by including the appropriate measurement command as the argument for the `print` command. For example, the following will request a channel A current reading:

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

## Source-measure programming example

The command sequence for a basic source-measure procedure is shown below. These commands set up the SourceMeter as follows:

- Source function and range: volts, 6V
- Source output level: 5V
- Current compliance: 10mA
- Measure function and range: current, 10mA

```
smua.reset()                --Restore Model 260x defaults.
smua.source.func = smua.OUTPUT_DCVOLTS--Select voltage source function.
smua.source.rangev = 6      --Set source range to 6V.
smua.source.levelv = 5     --Set voltage source to 5V.
smua.source.limiti = 10e-3  --Set current limit to 10mA.
smua.measure.rangei = 10e-3 --Set current range to 10mA.
smua.source.output =smua.OUTPUT_ON  --Turn on output.
print(smua.measure.i())    --Request current reading.
smua.source.output =smua.OUTPUT_OFF --Turn off output.
```

## Measure only

In addition to being used for conventional source-measure operations, the SourceMeter can also be used to measure only voltage or current. Perform the following steps to use the SourceMeter to measure voltage or current:

1. Select source-measure functions.
 

Measure voltage only (voltmeter) — Press SRC to select the I-Source, and press MEAS to select the voltage measurement function.

Measure current only (ammeter) — Press SRC to select the V-Source, and press MEAS to select the current measurement function.
2. Set source and compliance levels.
 

Use the editing procedure provided in steps 1 and 2 of the [“Front panel source-measure procedure”](#) on page 4-13 to edit the source and compliance levels as follows:

- a. Select the lowest source range and set the source level to zero (000.000nA or 000.000mV).
- b. Set compliance to a level that is higher than the expected measurement.

**CAUTION** When using the SourceMeter as a voltmeter, V-Compliance must be set higher than the voltage that is being measured. Failure to do this could result in excessive current flow into the SourceMeter (<150mA) and incorrect measurements.

3. Select range.  
Use the RANGE keys to select a fixed measurement range that will accommodate the expected reading. Use the lowest possible range for best accuracy.  
When measuring the function opposite from the source function, AUTO range can be used instead. The SourceMeter will automatically go to the most sensitive range.
4. Connect voltage or current to be measured. Connect the DUT to the SourceMeter using 2-wire connections (see [Section 3](#)).
5. Turn output on. Press the ON/OFF key to turn the output on.
6. Take reading from display (press TRIG if necessary). When finished, turn output off.

## Sink operation

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

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

**CAUTION** When using the I-Source as a sink, ALWAYS set V-Compliance to a level that is higher than the external voltage level. Failure to do so could result in excessive current flow into the SourceMeter (102mA) and incorrect measurements. See “[Compliance limit](#)” on page 4-3 for details.

The sink operating limits are shown in “[Operating boundaries](#)” on page 8-7.



# Ohms measurements

## Ohms calculations

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

$$R = V/I$$

Where: R is the calculated resistance  
V is the measured voltage  
I is the sourced current

## Ohms ranging

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

## Basic ohms measurement procedure

Perform the following steps to perform ohms measurements. The following procedure assumes that the SourceMeter is already connected to the DUT as explained in [Section 3](#).

**WARNING** Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, NEVER make or break connections to the Model 260x while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the SourceMeter before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.

1. For the Model 2602, press the DISPLAY key to select the single-channel display mode.
2. Press SRC to select the current source function, then set the output current to the desired value based on the expected resistance. See Step 1 of “[Front panel source-measure procedure](#)” on page 4-13.
3. Press the LIMIT key. Set the voltage limit high enough for the expected voltage across the resistance to be measured based on both the resistance value and programmed source current. See Step 2 of “[Front panel source-measure procedure](#)” on page 4-13.

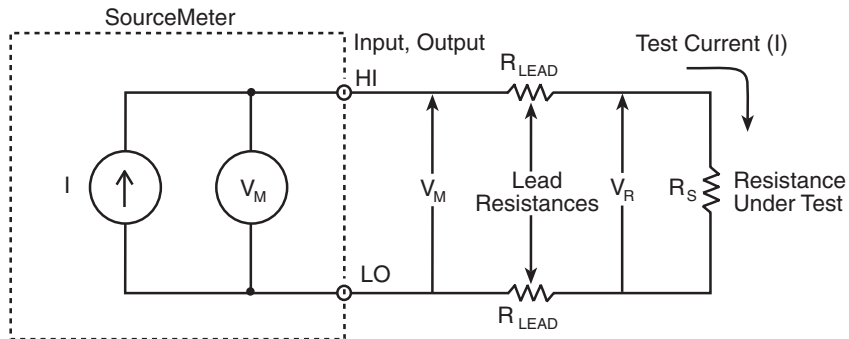
4. Press the MEAS or MODE key to display voltage, then make sure that AUTO measurement range is on.
5. Press the MEAS or MODE key to display ohms.
6. Turn on the output, then note the reading on the display. If necessary, press the TRIG key to display continuous readings. Turn off the output when finished.

## Ohms sensing

Ohms measurements can be made using either 2-wire or 4-wire sensing. (See Section 3 for information on connections and sensing methods.)

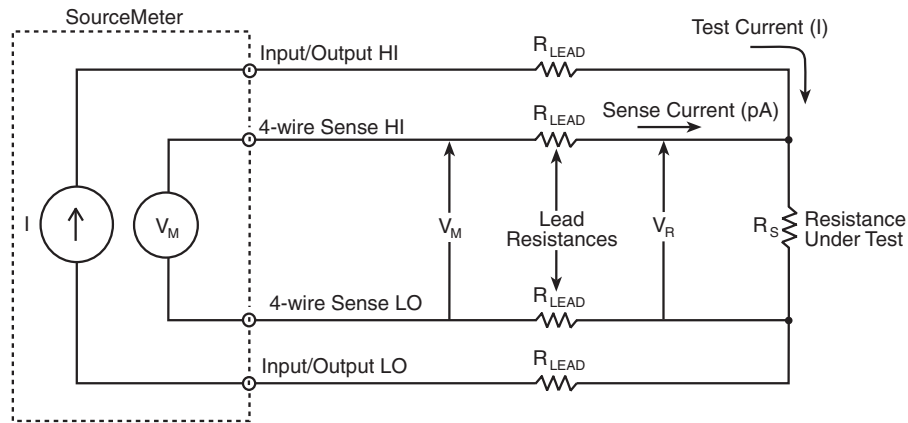
The 2-wire sensing method has the advantage of requiring only two test leads. However, as shown in Figure 4-3, test lead resistance can seriously affect the accuracy of 2-wire resistance measurements, particularly with lower resistance values. The 4-wire sensing method shown in Figure 4-4 minimizes or eliminates the effects of lead resistance by measuring the voltage across the resistor under test with a second set of test leads. Because of the high input impedance of the SourceMeter voltmeter, the current through the sense leads is negligible, and the measured voltage is essentially the same as the voltage across the resistor under test.

Figure 4-3  
**2-wire resistance sensing**



$$\begin{aligned}
 I &= \text{Current sourced by SourceMeter} \\
 V_M &= \text{Voltage measured by SourceMeter} \\
 V_R &= \text{Voltage across resistor} \\
 \text{Measured resistance} &= \frac{V_M}{I} = R_S + (2 \times R_{LEAD}) \\
 \text{Actual resistance} &= \frac{V_R}{I} = R_S
 \end{aligned}$$

Figure 4-4  
4-wire resistance sensing



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

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

## Sense selection

### Front panel sense selection

To select sensing mode:

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

### Remote sense selection

Use the `smuX.sense` command to control sense selection by remote. For example, send this command to enable 4-wire sensing:

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

See [Table 4-7](#) on [page 4-15](#) and [Section 12](#) for details.

## Remote ohms programming

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

### Remote ohms command

Use the following command to obtain a resistance reading:

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

See [Table 4-7 on page 4-15](#) for more commands necessary to set up source and measure functions and also [Section 12](#) for more details.

### Ohms programming example

The command sequence for a typical ohms measurement is shown below. These commands set up the SourceMeter as follows:

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

```
smua.reset() --Restore Model 260x defaults.
smua.source.func = smua.OUTPUT_DCAMPS --Select current source function.
smua.source.rangei = 10e-3 --Set source range to 10mA.
smua.source.leveli = 10e-3 --Set current source to 10mA.
smua.source.limitv = 10 --Set voltage limit to 10V.
smua.sense = smua.SENSE_REMOTE --Enable 4-wire ohms.
smua.measure.autorangev = smua.AUTORANGE_ON --Set voltage range to auto.
smua.source.output = smua.OUTPUT_ON --Turn on output.
print(smua.measure.r()) --Get resistance reading.
smua.source.output = smua.OUTPUT_OFF --Turn off output.
```

# Power measurements

## Power calculations

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

$$P = V \times I$$

Where: P is the calculated power  
V is the sourced or measured voltage  
I is the measured or sourced current

## Basic power measurement procedure

Perform the following steps to perform power measurements. The following procedure assumes that the SourceMeter is already connected to the DUT as explained in [Section 3](#).

**WARNING** Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, NEVER make or break connections to the Model 260x while the output is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the SourceMeter before handling cables connected to the outputs. Putting the equipment into standby does not guarantee the outputs are not powered if a hardware or software fault occurs.

1. For the Model 2602, press the DISPLAY key to select the single-channel display mode.
2. Set source function and value. Press SRC to select the voltage or current source function as required, then set the output voltage or current to the desired value. See Step 1 of [“Front panel source-measure procedure” on page 4-13](#).
3. Press the LIMIT key, and set the voltage or current limit high enough for the expected voltage or current across the DUT to be measured. See Step 2 of [“Front panel source-measure procedure” on page 4-13](#).
4. Press the MEAS or MODE key to display power.
5. Turn on the output, then note the reading on the display. If necessary, press the TRIG key to display continuous readings.
6. Turn off the output when finished.

## Remote power programming

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

### Remote power command

Use the following command to obtain a power reading:

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

See [Table 4-7 on page 4-15](#) for more commands necessary to set up source and measure functions and also Section 12 for more details.

### Power programming example

The command sequence for a typical power measurement is shown below. These commands set up the SourceMeter as follows:

- Source function: voltage, 6V range, 5V output
- Current measure function and range: current, auto
- Current compliance: 50mA

```
smua.reset()                --Restore Model 260x defaults.
smua.source.func = smua.OUTPUT_DCVOLTS --Select voltage source function.
smua.source.rangev = 6      --Set source range to 6V.
smua.source.levelv = 5     --Set voltage source to 5V.
smua.source.limiti = 50e-3  --Set current limit to 50mA.
smua.measure.autorangei = smua.AUTORANGE_ON--Set current range to auto.
smua.source.output = smua.OUTPUT_ON    --Turn on output.
print(smua.measure.p())                --Get power reading.
smua.source.output = smua.OUTPUT_OFF   --Turn off output.
```

# 5

# Sweep Operation

---

## **Section 5 topics**

### **Overview**, page 5-2

Section overview, page 5-2

Sweep overview, page 5-2

### **Sweep characteristics**, page 5-3

Linear staircase sweeps, page 5-3

Logarithmic staircase sweeps, page 5-4

Pulse sweeps, page 5-6

Custom (list) sweeps, page 5-7

Sweep measurement storage, page 5-8

### **Sweep functions**, page 5-8

Staircase sweep functions, page 5-9

Pulse sweep functions, page 5-10

Custom sweep functions, page 5-10

### **Running sweeps**, page 5-11

Front panel, page 5-11

Sweep programming examples, page 5-11

# Overview

## Section overview

Following a brief “[Sweep overview](#)” of the types of sweeps (linear staircase, logarithmic staircase, custom, and pulse), the documentation in this section provides detailed information on characteristics, commands, and programming for each type of sweep as follows:

- “[Sweep overview](#)”, page 5-2
- “[Sweep characteristics](#)”, page 5-3
- “[Sweep functions](#)”, page 5-8
- “[Running sweeps](#)”, page 5-11

## Sweep overview

As shown in [Figure 5-1](#), the Model 260x SourceMeter can generate several types of sweeps using the factory sweep scripts.

**Linear staircase sweep** – With this sweep type, the voltage or current increases or decreases in specific steps, beginning with a start current and ending with a stop current. [Figure 5-1A](#) shows an increasing linear staircase sweep.

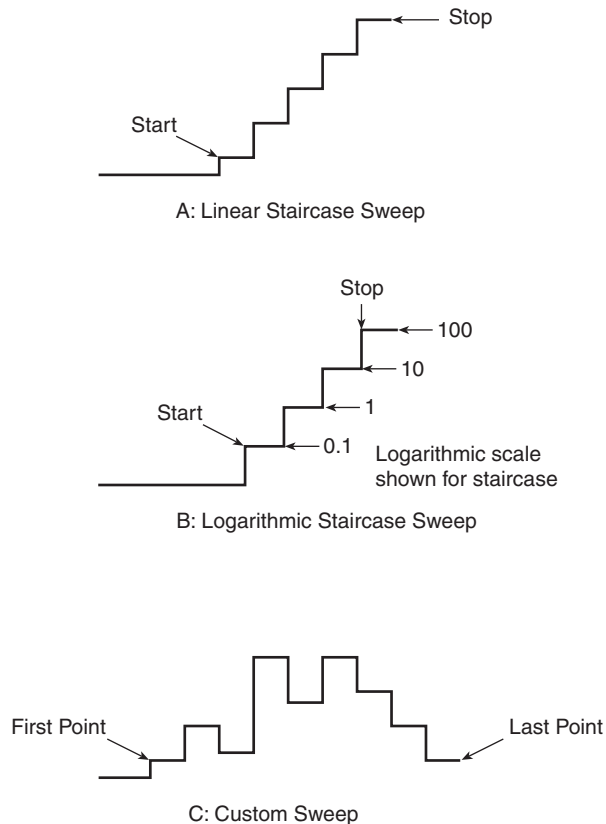
**Logarithmic staircase sweep** – In this case, the current or voltage increases or decreases logarithmically, beginning with a start voltage or current and ending with a stop voltage or current. [Figure 5-1B](#) shows an increasing logarithmic staircase sweep.

**Custom (list) sweep** – The custom sweep allows you to program arbitrary sweep steps anywhere within the output voltage or current range of the Model 260x. [Figure 5-1C](#) shows a typical custom sweep with arbitrary steps.

**Pulse sweeps** – Two sweeps can also be performed as a pulse sweep: fixed current pulse or fixed voltage pulse. These pulse sweeps are similar to those outlined above except that a pulse of a specific width is generated at each point instead of a constant level.



Figure 5-1  
**Comparison of staircase sweep types**



## Sweep characteristics

### Linear staircase sweeps

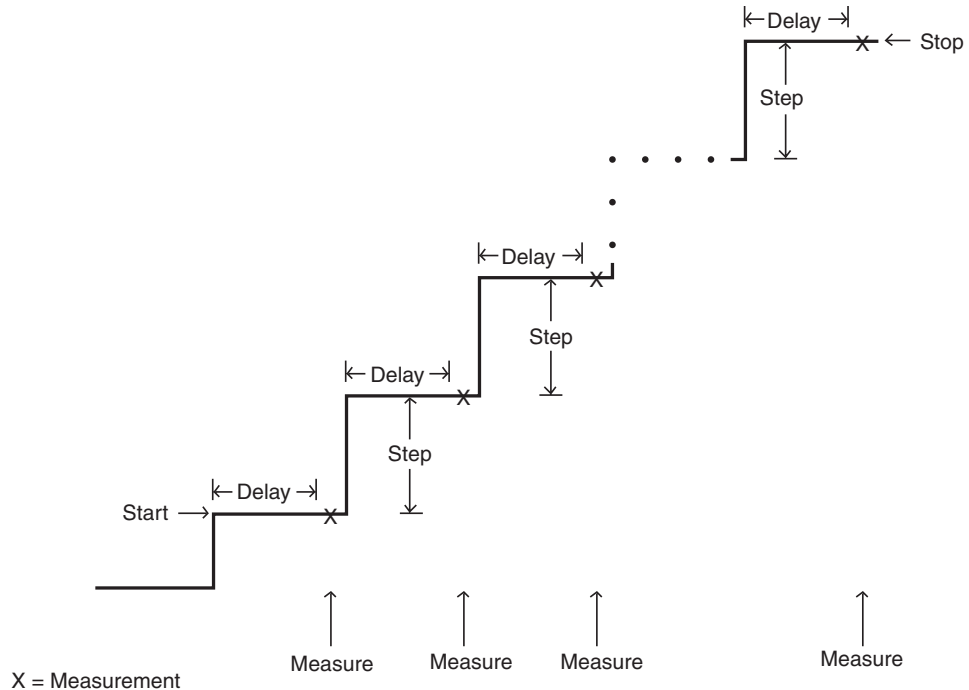
As shown in [Figure 5-2](#), this sweep type steps from a start voltage or current value to an ending (stop) value. A measurement is made at each step after a specified delay period (settling time). Programmable parameters include the source function, channel, start and stop levels, the number of sweep points, and the delay (the time between source and measure at each step). The step size is determined by the start and stop levels, and the number of sweep points:

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

The sweep can be either positive-going or negative-going, depending on the relative values of the start and stop parameters.

When this sweep starts, the output will go to the start source level. The output will then change in equal steps until the stop level is reached. The delay parameter determines the time duration before the measurement at each sweep step.

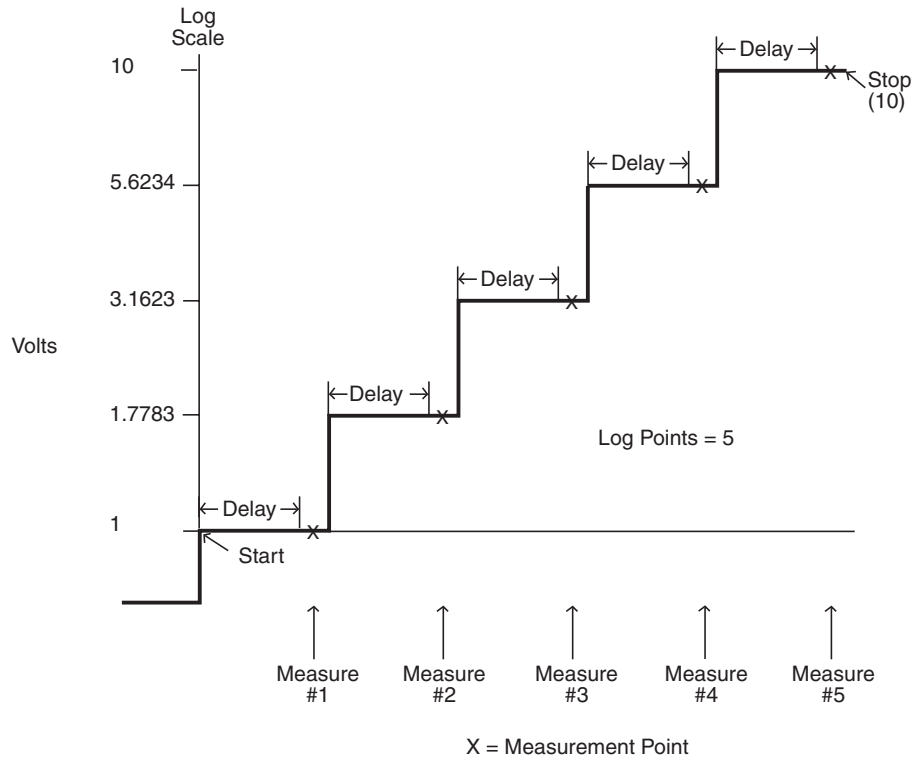
Figure 5-2  
**Linear staircase sweep**



## Logarithmic staircase sweeps

This sweep is similar to the linear staircase sweep. The steps, however, are done on a logarithmic scale as shown in the example sweep in [Figure 5-3](#). This example is a 5-point log sweep from 1V to 10V. The sweep can be either positive-going or negative-going depending on relative start and stop parameter values.

Figure 5-3  
**Logarithmic staircase sweep (1V to 10V, five steps)**



The programmable parameters for a log sweep include the source function, channel, start and stop levels, delay (settling time), and the number of measurement points for the sweep. The specified start, stop, and points parameters determine the logarithmic step size for the sweep. The delay parameter determines the time period before each measurement at each step.

Step size for the sweep in [Figure 5-3](#) is calculated as follows:

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

Thus, the five log steps for this sweep are 0, 0.25, 0.50, 0.75, and 1.00. The actual SourceMeter levels at these points are listed in [Table 5-1](#) (the voltage level is the anti-log of the log step).

Table 5-1  
**Logarithmic sweep points**

Measure point	Log step	Source level (V)
Point 1	0	1
Point 2	0.25	1.7783
Point 3	0.50	3.1623
Point 4	0.75	5.6234
Point 5	1.0	10

When this sweep starts, the output will go to the start level (1V) and sweep through the symmetrical log points. The time duration before each measurement at each step is determined by the measurement delay interval.

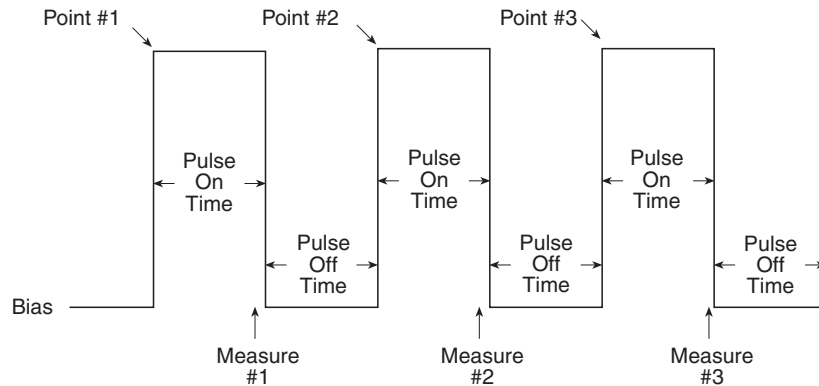
## Pulse sweeps

A fixed pulse sweep outputs fixed voltage or current pulses. Programmable parameters with this function include the sourcing function, the bias level, the pulse level, the number of pulses, the pulse on time, the pulse off time, and which SourceMeter channel is used (A or B).

When this sweep executes, the output will go from the bias level to the “on” level “of the first pulse. The time duration at each pulse level is determined by the programmed on time while the period between pulses is determined by the programmed off time.

An example of a simple three-point pulse sweep is shown in [Figure 5-4](#). Note that the programmed pulse on time determines the pulse width, and the pulse off time is the time between pulses. The level is the same for each pulse.

Figure 5-4  
Pulse sweep example



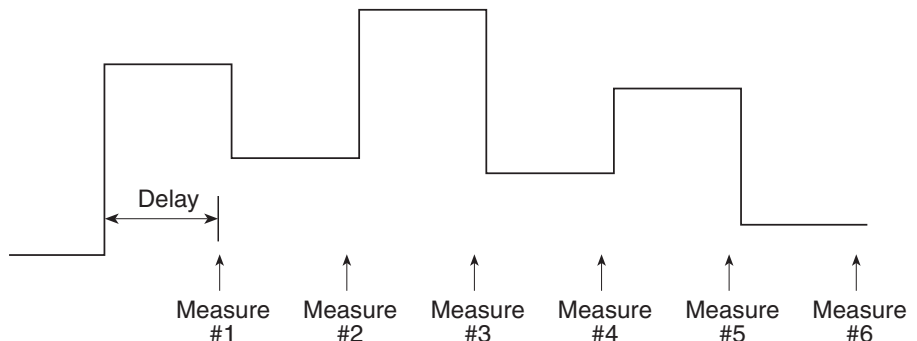
## Custom (list) sweeps

This sweep type lets you configure a customized sweep with arbitrary steps. Programmable parameters include the source function, channel, list of sweep values, delay (settling time), and the number of points.

When this sweep is started, the output level goes to the first point in the sweep. The sweep will continue through the steps in the order they were programmed and stop after the last step. The time duration before the measurement at each step is determined by the delay interval (settling time).

Figure 5-5 shows an example of a custom sweep with six measurement points. When the sweep starts, the current or voltage goes to the first point in the sweep. The unit cycles through the sweep points in the programmed order.

Figure 5-5  
Custom sweep example



## Sweep measurement storage

When sweeps are run, measurements are automatically stored in non-volatile memory Buffer 1 for later recall. Sweep data can be recalled as follows:

- **Front panel:** Press the RECALL key, select the channel and Buffer 1, then choose reading numbers to display with the Rotary Knob or cursor keys.
- **Remote:** Use the `printbuffer` command to request buffer readings from `smua.nvbuffer1` (channel A) or `smub.nvbuffer1` (channel B).

See [Section 7](#) for details on recalling data from the buffer.

## Sweep functions

Functions to perform staircase, pulse, and custom sweeps are discussed below. See [Section 13](#) of this manual for details on using factory scripts.

**NOTE** The functions shown indicate factory scripts at the time of this writing. See the release notes for any updates to these factory scripts that support sweeps.

Also visit [www.keithley.com](http://www.keithley.com) for additional available user scripts for various tests.

## Staircase sweep functions

Functions for linear and logarithmic staircase sweeps are listed in [Table 5-2](#).

Table 5-2

### Staircase sweep functions

Command	Description
SweepLinMeasureV(smu, starti, stopi, stime, points) smu start i stopi stime points	Define linear source current sweep:  Smu: smua for channel A or smub for channel B. Start current value in amps. Stop current value in amps. Settling time (source-measure delay in seconds). Number of points ( $\geq 2$ ).
SweepVLinMeasureI(smu, startv, stopv, stime, points) smu startv stopv stime points	Define linear source voltage sweep:  Smu: smua for channel A or smub for channel B. Start voltage value in volts. Stop voltage value in volts. Settling time (source-measure delay in seconds). Number of points ( $\geq 2$ ).
SweepLogMeasureV(smu, starti, stopi, stime, points) smu starti stopi stime points	Define log source current sweep:  Smu: smua for channel A or smub for channel B. Start current value in amps. Stop current value in amps. Settling time (source-measure delay in seconds). Number of points ( $\geq 2$ ).
SweepVLogMeasureI(smu, startv, stopv, stime, points) smu startv stopv stime points	Define log source voltage sweep:  Smu: smua for channel A or smub for channel B. Start voltage value in volts. Stop voltage value in volts. Settling time (source-measure delay in seconds). Number of points ( $\geq 2$ ).

## Pulse sweep functions

Functions for pulse sweeps are listed in [Table 5-3](#).

Table 5-3

### Pulse sweep functions

Function	Description
PulseIMeasureV(smu, bias, level, ton, toff, points) smu bias level ton toff points	Define fixed source current pulse sweep: Smu: smua for channel A or smub for channel B. DC bias current level in amps. On source value of the pulse in amps. Pulse on time in seconds. Pulse off time in seconds. Number of pulse-measure cycles.
PulseVMeasureI(smu, bias, level, ton, toff, points) channel bias level ton toff points	Define fixed source voltage pulse sweep: Smu: smua for channel A or smub for channel B. DC bias voltage level in volts. On source value of the pulse in volts. Pulse on time in seconds. Pulse off time in seconds. Number of pulse-measure cycles.

## Custom sweep functions

Functions for list (custom) sweeps are listed in [Table 5-4](#).

Table 5-4

### Custom sweep functions

Command	Description
SweepIListMeasureV(smu, ilist, stime, points) smu ilist stime points	Define current list sweep: Smu: smua for channel A or smub for channel B. List of current values in amps. Settling time (source-measure delay in seconds). Number of points ( $\geq 2$ ).
SweepVListMeasureI(smu, vlist, stime, points) smu vlist stime points	Define voltage list sweep: Smu: smua for channel A or smub for channel B. List of voltage values in volts. Settling time (source-measure delay in seconds). Number of points ( $\geq 2$ ).



# Running sweeps

## Front panel

To run a sweep, press the LOAD key, then select the test to run. Press the RUN key, then follow the display prompts to complete the test (refer to [Table 5-2](#) through [Table 5-4](#) for sweep parameters). See [Section 13](#) for more information on using factory scripts.

Press the RECALL key to access sweep data stored in Buffer 1. See [Section 7](#) for more details on the buffer.

## Sweep programming examples

Procedures for programming and running a sweep for three sweep types are given on the following pages. Each of these procedures includes commands for a typical sweep example. [Table 5-5](#) summarizes parameters for each of these examples.

**NOTE** These programming examples use factory sweep scripts as defined at the time of this writing. See the release notes for any updates.

Table 5-5  
Sweep example parameters

Sweep type	Parameters for sweep examples
Linear staircase sweep ( <a href="#">page 5-12</a> )	Start current: 1mA Stop current: 10mA # points: 10 Settling time: 0.1s
Pulse current sweep ( <a href="#">page 5-13</a> )	Bias current: 1mA On current: 10mA Pulse on time: 10ms Pulse off time: 50ms # points: 10
Custom (list) sweep ( <a href="#">page 5-14</a> )	Five points: 3V, 1V, 4V, 5V, 2V Settling time 0.1s

## Linear staircase sweep example

### 1. Configure source functions.

**Examples** – The following commands restore defaults and set the compliance to 1V:

```
smua.reset()                --Restore Model 260x defaults.  
smua.source.limitv = 1     --Set compliance to 1V.
```

### 2. Configure the sweep.

**Example** – The following parameters configure a linear staircase current sweep from 1mA to 10mA with 10 points and a 0.1 second settling time:

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

### 3. Run the sweep.

```
smua.source.output =        --Turn on output.  
smua.OUTPUT_ON  
waitcomplete()             --Wait for sweep to complete.
```

### 4. Turn off the output.

When the sweep is done, turn the source output off with this command:

```
smua.source.output = smua.OUTPUT_OFF
```

### 5. Request readings.

Request readings from Buffer 1 as follows:

```
printbuffer(1, 10, smua.nvbuffer1.readings)
```

## Pulse sweep example

### 1. Configure source functions

**Examples** – The following commands restore defaults and set the compliance to 10V:

```
smua.reset()                --Restore Model 260x defaults.
smua.source.limitv = 10     --Set compliance to 5V.
```

### 2. Configure the sweep.

**Example** – The following parameters configure a 10mA current pulse sweep with a 10ms pulse on time, a 50ms pulse off time, and 10 pulse-measure cycles:

```
PulseIMeasureV(smua, 1e-3,    --Pulse current sweep, channel A,
10e-3, 10e-3, 50e-3, 10)     1mA bias, 10mA level, 10ms pulse
                               on, 50ms pulse off, 10 cycles.
```

### 3. Run the sweep.

```
smua.source.output =        --Turn on output.
smua.OUTPUT_ON
waitcomplete()              --Wait for sweep to complete.
```

### 4. Turn off output.

When the sweep is done, turn the source output off with this command:

```
smua.source.output = smua.OUTPUT_OFF
```

### 5. Request readings.

Request readings from Buffer 1 as follows:

```
printbuffer(1, 10, smua.nvbuffer1.readings)
```

## Custom sweep example

### 1. Configure source functions

**Examples** – The following commands restore defaults and set the compliance to 10mA:

```
smua.reset()                --Restore Model 260x defaults.
smua.source.limiti = 10e-3  --Set compliance to 10mA.
```

### 2. Configure the sweep.

**Example** – The following parameters configure a list sweep with 3V, 1V, 4V, 5V, 2V points using a 0.1s settling time:

```
vlist = {5, 3, 1, 4, 5, 2}  --Define voltage list.
SweepVListMeasureI(smua, vlist, --List sweep, channel A, 3V, 1V, 4V,
0.1, 5)                    5V, 2V steps, 0.1s delay, 5 points.
```

### 3. Run the sweep.

```
smua.source.output =        --Turn on output.
smua.OUTPUT_ON
waitcomplete()              --Wait for sweep to complete.
```

### 4. Turn off output.

When the sweep is done, turn the source output off with this command:

```
smua.source.output = smua.OUTPUT_OFF
```

### 5. Request readings.

Request readings from Buffer 1 as follows:

```
printbuffer(1, 5, smua.nvbuffer1.readings)
```

# 6

## Range, Digits, Speed, Rel, and Filters

---

### Section 6 topics

**Overview**, page 6-2

**Range**, page 6-2

Available ranges, page 6-2

Maximum source values and readings, page 6-3

Ranging limitations, page 6-3

Manual ranging, page 6-3

Auto ranging, page 6-3

Low range limits, page 6-4

Range considerations, page 6-4

Range programming, page 6-5

**Digits**, page 6-7

Setting display resolution, page 6-7

Remote digits programming, page 6-7

**Speed**, page 6-8

Setting speed, page 6-9

Remote speed programming, page 6-9

**Rel**, page 6-10

Front panel rel, page 6-10

Remote rel programming, page 6-11

**Filters**, page 6-12

Filter types, page 6-12

Response time considerations, page 6-12

Front panel filter control, page 6-12

Remote filter programming, page 6-15

## Overview

The documentation in this section provides detailed information on characteristics and script programming for each of the following functions:

- “Range”, page 6-2
- “Digits”, page 6-7
- “Speed”, page 6-8
- “Rel”, page 6-10
- “Filters”, page 6-12

## Range

The selected measurement range affects the accuracy of the measurements as well as the maximum signal that can be measured. Note that dashed lines are displayed (i.e., --.----  $\mu\text{A}$ ), to indicate that the previous measurement is not recent. This usually happens when a change occurs such as selecting a different range.

### Available ranges

Table 6-1 lists the available source and measurement ranges for the Model 260x SourceMeter.

Table 6-1  
Source and measurement ranges

Voltage Ranges	Current Ranges
100mV	100nA
1V	1 $\mu\text{A}$
6V	10 $\mu\text{A}$
40V	100 $\mu\text{A}$
	1mA
	10mA
	100mA
	1A
	3A

## Maximum source values and readings

The full scale output for each voltage and current source range is 101% of the selected range, while the full scale measurement is 102% of the range. For example,  $\pm 1.01\text{V}$  is the full scale source value for the 1V range, and  $\pm 102\text{mA}$  is the full scale reading for the 100mA measurement range. Input levels that exceed the maximum levels cause the overflow message to be displayed. Note, however, that the instrument will auto range at 100% of the range.

## Ranging limitations

- With the 40V V-Source range selected, the highest current measurement range is 1A. With the 3A I-Source range selected, the highest voltage measurement range is 6V.
- For Source V Measure I or Source I Measure V, you can set source and measure ranges separately. If both source and measure functions are the same, the measure range is locked to the source range.

## Manual ranging

The RANGE  $\triangle$  and  $\nabla$  keys are used to select a fixed range:

- To set source range, press SRC, then use the RANGE keys to set the range.
- To set measure range, select the single-channel display mode (Model 2602 only), press MEAS, then set the range with the RANGE keys (Source V Measure I, or Source I Measure V).

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

The SourceMeter allows you to send ICL command values that may be out of range when auto range is off. An example is sending 1V on the 100mV range, in which case the unit does not error check until the output is turned on. In this situation, the display will show a series of question marks: ????.???

## Auto ranging

To use auto source ranging, press SRC then AUTO RANGE. To use auto measure ranging, select the Model 2602 single-channel display mode, then press MEAS followed by AUTO RANGE. The AUTO annunciator turns on when source or measure auto ranging is selected. With auto ranging selected, the instrument automatically chooses the best range to source or measure the applied signal. The instrument will auto range at 100% of range.

Note that source auto ranging will turn off when editing the source value.

## Low range limits

The low range limits set the lowest range the Model 260x will use when auto ranging is enabled. This feature is useful for minimizing auto range settling times when numerous range changes are involved.

Low range limits can be individually set for Source V, Source I, Measure V, and Measure I as follows:

1. Press the CONFIG key, then press either SRC for source or MEAS for measure.
2. Choose voltage or current source, or measure as appropriate, and then press ENTER or the Rotary Knob.
3. Choose LOWRANGE, then press ENTER or the Rotary Knob.
4. Set the low range to the desired setting, and then press ENTER or the Rotary Knob.
5. Use EXIT to back out of the menu structure.

## Range considerations

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

If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is locked to be the same as the source range. However, the setting for the voltage measure range is retained and used when the source function is changed to current, and the present voltage measurement range will be used.

Example:

```
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.rangev = 1
smua.measure.rangev = 6
print(smua.measure.rangev)      -- will print 1, to match
                                source range
smua.source.func = smua.OUTPUT_DCAMPS
print(smua.measure.rangev)      -- will print 6, the user's
                                range
```

Explicitly setting either a source or measurement range for a function will disable auto ranging for that function. Autoranging is controlled separately for each source and measurement function: source voltage, source current, measure voltage and measure current. Auto ranging is enabled for all four by default.

Changing the range while the output is off will not update the hardware settings, but querying will return the range setting that will be used once the output is turned on. Setting a range while the output is on will take effect immediately.



With source auto ranging enabled, the output level controls the range. Querying the range after the level is set will return the range the unit chose as appropriate for that source level.

The SourceMeter allows you to send ICL command values that may be out of range when auto range is off. An example is sending 1V on the 100mV range. The unit does not error check until the output is turned on. In this situation, the display will show a series of question marks:???.???

With measure auto ranging enabled, the range will be changed only when a measurement is taken. Querying the range after a measurement will return the range selected for that measurement.

## Range programming

### Range commands

[Table 6-2](#) summarizes the commands necessary to control measure and source ranges. See [Section 12](#) for more details on these commands.

Table 6-2

**Range commands**

Commands <sup>1</sup>	Description
<p><b>Measure range commands:</b> <sup>2</sup></p> <p>smuX.measure.autorangei = smuX.AUTORANGE_ON  smuX.measure.autorangei = smuX.AUTORANGE_OFF  smuX.measure.autorangev = smuX.AUTORANGE_ON  smuX.measure.autorangev = smuX.AUTORANGE_OFF  smuX.measure.lowrangei = lowrange  smuX.measure.lowrangev = lowrange  smuX.measure.rangei = rangeval  smuX.measure.rangev = rangeval</p> <p><b>Source range commands:</b> <sup>2</sup></p> <p>smuX.source.autorangei = smuX.AUTORANGE_ON  smuX.source.autorangei = smuX.AUTORANGE_OFF  smuX.source.autorangev = smuX.AUTORANGE_ON  smuX.source.autorangev = smuX.AUTORANGE_OFF  smuX.source.limiti = level  smuX.source.limitv = level  smuX.source.lowrangei = lowrange  smuX.source.lowrangev = lowrange  smuX.source.rangei = rangeval  smuX.source.rangev = rangeval</p>	<p>Enable current measure auto range.  Disable current measure auto range.  Enable voltage measure auto range.  Disable voltage measure auto range.  Set lowest I measure range for auto range.  Set lowest V measure range for auto range.  Select manual current measure range.  Select manual voltage measure range.</p> <p>Enable current source auto range.  Disable current source auto range.  Enable voltage source auto range.  Disable voltage source auto range.  Set voltage source current limit.  Set current source voltage limit.  Set lowest I source range for auto range.  Set lowest V source range for auto range.  Select manual current source range.  Select manual voltage source range.</p>

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

2. See [Table 6-1 on page 6-2](#) for ranges.

**Range programming example**

The listing below shows a programming example for controlling both source and measure ranges. The SourceMeter is set up as follows:

- Voltage source range: 10V
- Current measure range: auto
- Measure low range: = 10 $\mu$ A
- Voltage source current limit: 10mA

```
smua.reset()                --Restore Model 260x defaults.
smua.source.rangev = 10     --Set V source range to 10V.
smua.measure.autorangei =  --Enable I measure auto range.
smua.AUTORANGE_ON
smua.measure.lowrangei = 1e-5 --Set lowest range to 10µA.
smua.source.limiti = 1e-2   --Set limit level to 10mA.
```

## Digits

The display resolution of the measured reading depends on the DIGITS setting. This setting is global, which means the digits setting selects display resolution for all measurement functions.

The DIGITS setting has no effect on the remote reading format. The number of displayed digits does not affect accuracy or speed. Those parameters are controlled by the SPEED setting (see “[Speed](#)” on page 6-8).

### Setting display resolution

To set display resolution, press the DIGITS key until the desired number of digits is displayed. The display resolution will cycle through 4.5, 5.5, and 6.5 digits.

**NOTE** For the Model 2602 dual-channel display mode, the maximum display resolution is 4.5 digits. For the Model 2602 single-channel display mode, pressing the DIGITS key for the channel not being displayed will have no effect, but the unit will display a message advising you to change to the indicated channel.

### Remote digits programming

#### Digits commands

[Table 6-3](#) summarized digits commands. See [Section 12](#) for more information.

Table 6-3  
**Digits commands**

Command <sup>1</sup>	Description
display.smuX.digits = display.DIGITS_4_5	Set display to 4.5 digits.
display.smuX.digits = display.DIGITS_5_5	Set display to 5.5 digits.
display.smuX.digits = display.DIGITS_6_5	Set display to 6.5 digits.

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

### Digits programming example

```
smua.reset ()                                --Restore Model 260x defaults.
display.smua.digits = display.DIGITS_5_5 --Select 5.5 digits.
```

## Speed

The SPEED key is used to set the integration time, or measurement aperture, of the A/D converter (period of time the input signal is measured). The integration time affects the usable digits, the amount of reading noise, and the ultimate reading rate of the instrument. The integration time is specified in parameters based on the Number of Power Line Cycles (NPLC), where 1 PLC for 60Hz is 16.67msec (1/60) and 1 PLC for 50Hz is 20msec (1/50).

In general, the fastest integration time (FAST; 0.01 PLC) results in the fastest reading rate, but at the expense of increased reading noise and fewer usable digits. The slowest integration time (25 PLC) provides the best common-mode and normal-mode noise rejection, but has the slowest reading rate. In-between settings are a compromise between speed and noise. The default power-on speed setting is NORMAL (1 PLC).

## Setting speed

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

### SPEED configuration menu

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

- **FAST** — Sets speed to 0.01 PLC and sets display resolution to 4-1/2 digits.
- **MED** — Sets speed to 0.10 PLC and sets display resolution to 5-1/2 digits.
- **NORMAL** — Sets speed to 1.00 PLC and sets display resolution to 6-1/2 digits.
- **HI ACCURACY** — Sets speed to 10.00 PLC and sets display resolution to 6-1/2 digits.
- **OTHER** — Used to set speed to any PLC value from 0.001 to 25. Display resolution is not changed when speed is set with this option.

**NOTE** The SPEED setting affects all measurement functions. After setting speed, display resolution can be changed using the DIGITS key. For the Model 2602 single-channel display mode, pressing the SPEED key for the channel not being displayed will result in a display message to change to the other channel before setting speed.

## Remote speed programming

### Speed command

Table 6-4 summarizes commands to control speed. See [Section 12](#) for more information.

Table 6-4

**Speed command**

Command <sup>1</sup>	Description
smuX.measure.nplc = nplc	Set speed (nplc = 0.001 to 25). <sup>2</sup>

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

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

## Speed programming example

Use the NPLC command to set the speed. For example, send the following parameter to set the speed to 10 PLC:

```
smua.reset ()                --Restore Model 260x defaults.  
smua.measure.nplc = 10      --Set NPLC to 10.
```

# Rel

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

Displayed Reading = Actual Input - Rel Value

Once a rel value is established for a measurement function, the value is the same for all ranges. For example, if 5V is set as a rel value on the 6V range, the rel value is also 5V on the 1V and 100mV ranges.

Selecting a range that cannot accommodate the rel value does not cause an overflow condition, but it also does not increase the maximum allowable input for that range. For example, on 6V range, the SourceMeter still overflows for a >6.06V input.

**NOTE** When rel is enabled, the REL annunciator turns on. Changing measurement functions disables rel.

## Front panel rel

### Enabling and disabling rel

Rel can be used to null out zero offsets or to establish a zero baseline by pressing the REL key. The reading (which becomes the rel value) is subtracted from itself. As a result, a zero reading is displayed. Pressing REL a second time disables rel.

### Defining a rel value

A unique rel value can be established for the selected measurement function from the front panel as follows:

1. Press CONFIG then REL.
2. Choose the measurement function (CURRENT, VOLTAGE, OHMS, or WATTS), then press ENTER or the Rotary Knob.

3. The present rel value will be displayed.
4. Set the desired rel value.
5. With the desired rel value displayed, press ENTER or the Rotary Knob, and then use EXIT to back out of the menu structure.

## Remote rel programming

### Rel commands

Rel commands are summarized in [Table 6-5](#).

Table 6-5

#### Rel commands

Command <sup>1</sup>	Description
<b>To set rel values:</b> smuX.measure.rel.leveli = relval smuX.measure.rel.levelp = relval smuX.measure.rel.levelr = relval smuX.measure.rel.levelv = relval	Set current rel value. Set power rel value. Set resistance rel value. Set voltage rel value.
<b>To enable/disable rel:</b> smuX.measure.rel.enablei = smuX.REL_OFF smuX.measure.rel.enablep = smuX.REL_OFF smuX.measure.rel.enabler = smuX.REL_OFF smuX.measure.rel.enablev = smuX.REL_OFF smuX.measure.rel.enablei = smuX.REL_ON smuX.measure.rel.enablep = smuX.REL_ON smuX.measure.rel.enabler = smuX.REL_ON smuX.measure.rel.enablev = smuX.REL_ON	Disable current rel. Disable power rel. Disable resistance rel. Disable voltage rel. Enable current rel. Enable power rel. Enable resistance rel. Enable voltage rel.

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

### Rel programming example

```

smua.reset ()           --Restore Model 260x defaults.
smua.measure.rel.leveli = 0.1  --Set current rel to 100mA.
smua.measure.rel.enablei =    --Enable current rel.
smua.REL_ON

```

# Filters

Filter lets you set the filter response to stabilize noisy measurements. The SourceMeter uses a digital filter, which is based on reading conversions. The displayed, stored, or transmitted reading is an average of many reading conversions (from 1 to 100).

## Filter types

There are two averaging filter types to choose from: repeating and moving (Figure 6-1). For the repeating filter (which is the power-on default), the stack (filter count) is filled, and the conversions are averaged to yield a reading. The stack is then cleared, and the process starts over.

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

When the filter is first enabled, the stack is empty. Keep in mind that a filtered reading is not yielded until the stack is full. The first reading conversion is placed in the stack and is then copied to the other stack locations in order to fill it. Thus, the first filtered reading is the same as the first reading conversion. Now the normal moving average filter process can continue. Note that a true average is not yielded until the stack is filled with new reading conversions (no copies in stack). For example, in Figure 6-1A, it takes ten filtered readings to fill the stack with new reading conversions. The first nine filtered readings are calculated using copied reading conversions.

## Response time considerations

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

## Front panel filter control

### Configuring filter

Filter type and count is configured from the filter configuration menu. The configured filter is the same for all measurement functions.



## Filter configuration menu

Press CONFIG and then FILTER to display the filter configuration menu:

- AVERAGE-TYPE — Use this menu item to select filter type (MOVING or REPEAT).
- AVERAGE-COUNT — Use this menu item to specify filter count (1 to 100 readings).

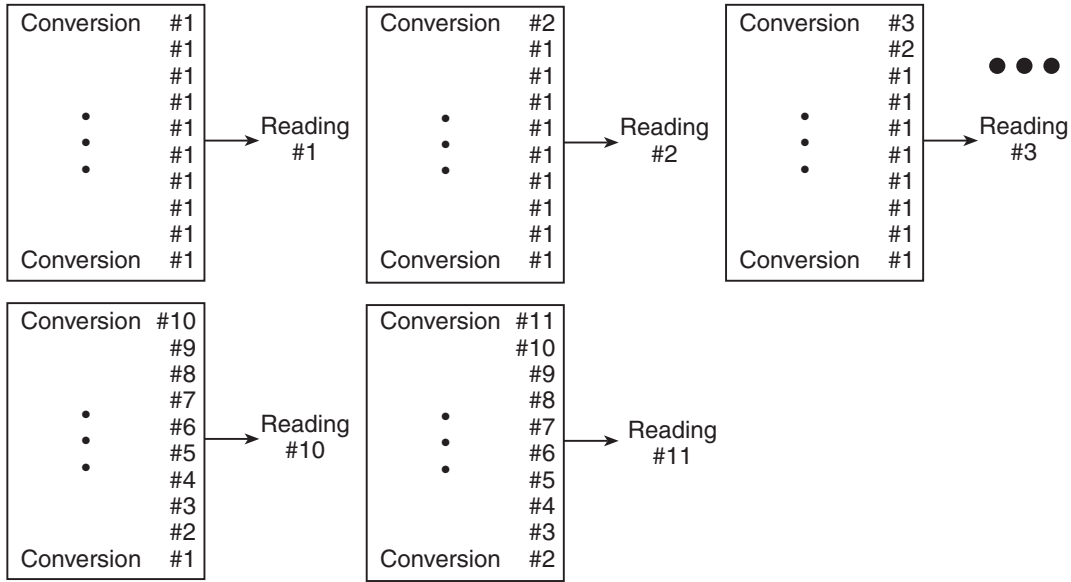
## Enabling filter

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

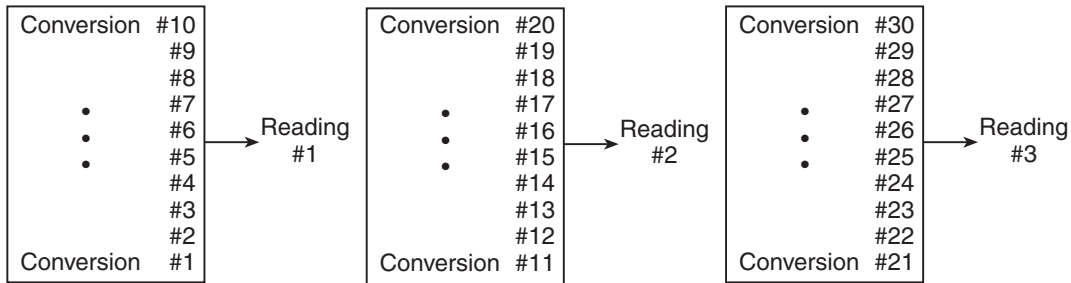
## Response time

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

Figure 6-1  
**Moving average and repeating filters**



**A. Type - Moving Average, Readings = 10**



**B. Type - Repeating, Readings = 10**

## Remote filter programming

### Filter commands

Table 6-6 summarizes filter commands. See Section 12 for more details.

Table 6-6

#### Filter commands

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

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

### Filter programming example

The example below programs filter aspects as follows:

- Filter type: moving average
- Filter count: 10
- Filter state: enabled

```
smua.reset()                --Restore Model 260x defaults.
smua.measure.filter.count = 10  --Program count to 10.
smua.measure.filter.type = smua.FILTER_MOVING_AVG --Moving average filter type
smua.measure.filter.enable = smua.FILTER_ON      --Enable filter.
```

# 7

## Buffer (Data Store)

---

### **Section 7 topics**

**Overview**, page 7-2

**Data store overview**, page 7-2

**Front panel data store**, page 7-2

Buffer configuration, page 7-2

Buffer configuration menu, page 7-3

Storing readings, page 7-3

Recalling readings, page 7-4

**Remote data store**, page 7-5

Data store commands, page 7-5

Reading buffers, page 7-6

Time and date values, page 7-10

Buffer status, page 7-10

Dynamically allocated buffers, page 7-11

Buffer programming examples, page 7-12

# Overview

The documentation in this section provides detailed information on using the buffer to store data and includes the following:

- "Data store overview", page 7-2
- "Front panel data store", page 7-2
- "Remote data store", page 7-5

## Data store overview

The SourceMeter has two buffers per channel that can store from 1 to more than 100,000 readings. The instrument can store the readings that are displayed during the storage process. Each buffer reading is numbered and can also include the source value and a timestamp.

## Front panel data store

### Buffer configuration

The buffer can be configured through the buffer configuration menu, which is accessed as follows:

1. Press the CONFIG key followed by the STORE key.
2. Using the following menu, configure the buffer as required.

**NOTE** You must clear the buffer before enabling or disabling data element storage (source value or timestamp).

## Buffer configuration menu

The various buffer configuration menu items include:

- **COUNT:** Sets number of readings to store (1 to 110,000).
- **CHANA\_BUFF:** Configures Channel A buffer.
  - **DEST:** Sets data storage destination (Buffer 1, Buffer 2, or NONE).
  - **BUFFER1:** Configure buffer 1.
    - **CLEAR:** Clear buffer (YES or NO).
    - **ELEMENTS:** Enable (ON) or disable (OFF) data storage elements; SRC-VAL (source value) or TSTAMP (time stamp).
  - **BUFFER2:** Configure buffer 2.
    - **CLEAR:** Clear buffer (YES or NO).
    - **ELEMENTS:** Enable (ON) or disable (OFF) data storage elements; SRC-VAL (source value) or TSTAMP (time stamp).
- **CHANB\_BUFF:** Configures Channel B buffer (Model 2602 only).
  - **DEST:** Sets data storage destination (Buffer 1, Buffer 2, or NONE).
  - **BUFFER1:** Configure buffer 1.
    - **CLEAR:** Clear buffer (YES or NO).
    - **ELEMENTS:** Enable (ON) or disable (OFF) data storage elements; SRC-VAL (source value) or TSTAMP (time stamp).
  - **BUFFER2:** Configure buffer 2.
    - **CLEAR:** Clear buffer (YES or NO).
    - **ELEMENTS:** Enable (ON) or disable (OFF) data storage elements; SRC-VAL (source value) or TSTAMP (time stamp).

**NOTE** Model 2601 buffer configuration menu items are the same as covered above except for channel selection.

## Storing readings

**NOTE** Pressing the Model 2602 STORE key will overwrite the destination buffer on both channels unless NONE is selected (see above).

Perform the following steps to store readings:

1. Set up the SourceMeter for the desired settings and buffer configuration using the configuration menu described above.
2. Turn on the output.
3. Press the STORE key. The asterisk (\*) annunciator turns on to indicate data storage operation is enabled. It will turn off when the storage is finished.
4. Press EXIT to stop data storage before it finishes.

## Recalling readings

Readings stored in the buffer are displayed by pressing the RECALL key. Repeatedly pressing RECALL will cycle through Buffer 1 then Buffer 2 for Channel A and then Channel B (Model 2602 only). A message will be displayed if a buffer is empty.

The reading display is on the top left, while the buffer location number is on the right. The source values are positioned at the lower left side of the display (if enabled), while the timestamp (if used) is positioned at the lower right side. When toggling between buffers with RECALL, the source display field will identify the buffer: Src1A (Buffer 1, Channel A), then Src2A (Buffer 2, Channel A); followed by (Model 2602 only) Src1B (Buffer 1, Channel B) then Src2B (Buffer 2, Channel B).

### Buffer location number

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

### Timestamp

If the timestamp is enabled, the first source-measure reading stored in the buffer (#0000001) is timestamped at 0000000.001 seconds. Subsequent readings are timestamped relative to the time storage was started, and the interval between readings will depend on the reading rate.

### Displaying other buffer readings

To display the other source-measure readings stored in the buffer, display the desired memory location number. Use the Rotary Knob to increment and decrement the selected digit of the location number. Cursor position is controlled by the Rotary Knob or CURSOR keys.

To exit from the data store recall mode, press EXIT.

# Remote data store

## Data store commands

Table 7-1 summarizes commands associated with data store operation. See Section 12 for more detailed information on data store commands.

Table 7-1

**Data store commands**

Command <sup>1</sup>	Description
smuX.nvbuffer1.clear() smuX.nvbuffer2.clear() mybuffer = smuX.makebuffer(n) mybuffer = nil  <b>Commands to store readings:<sup>2</sup></b>  smuX.measure.count = count smuX.measure.overlappedi(rbuffer) smuX.measure.overlappediv(ibuffer, vbuffer)  smuX.measure.overlappedp(rbuffer) smuX.measure.overlappedr(rbuffer) smuX.measure.overlappedv(rbuffer)  <b>Commands to access readings:<sup>2</sup></b>  print(smuX.measure.i(rbuffer)) print(smuX.measure.iv(ibuffer, vbuffer))  print(smuX.measure.p(rbuffer)) print(smuX.measure.r(rbuffer)) print(smuX.measure.v(rbuffer)) printbuffer(start_index, end_index, st_1 [, st_n])  printnumber(v1[, vn])	Clear Buffer 1. Clear Buffer 2. Create dynamically allocated buffer, n readings. Delete dynamically allocated buffer.  Store count number of buffer readings. Store current readings in buffer. Store current and voltage readings in respective buffers (current and then voltage are stored in separate buffers). Store power readings in buffer. Store resistance readings in buffer. Store voltage readings in buffer.  Return first buffer current reading. Return first current and then voltage reading from two separate buffers. Return first buffer power reading. Return first buffer resistance reading. Return first buffer voltage reading. Print data from buffer subtables: start_index (Starting index of values to print). end_index (Ending index of values to print). st_1 ... st_n (Sub-tables from which to print). <sup>3</sup> Print numbers with selected buffer format: v1 ... vn (Numbers to print).

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

2. rbuffer, ibuffer, and vbuffer = smuX.nvbuffer1 or smuX.nvbuffer2.

3. See "Reading buffers" on page 7-6 for more information.



## Reading buffers

Readings can be obtained in multiple ways including synchronous or overlapped. Furthermore, the routines that make single point measurements can be configured to make multiple measurements where one would ordinarily be made. The measured value is not the only component of a reading. The measurement status (e.g. “In Compliance” or “Overranged”) is also an element of data associated with a particular reading.

All routines that return measurements can return them as reading buffers. Overlapped measurements are always returned in a reading buffer. Synchronous measurements return either a single-point measurement or a buffer reading. The more advanced user can access additional information stored in the reading buffer.

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

### Reading buffer designations

There are two non-volatile reading buffers designated as `smuX.nvbuffer1` (Buffer 1) and `smuX.nvbuffer2` (Buffer 2). To access the buffer, simply include the buffer attribute in the respective command. For example, the following command would store current readings from Channel A into Buffer 1:

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

### Buffer storage control attributes

Buffer storage attributes are summarized in [Table 7-2](#). Read-only attributes used to access buffer parameters are listed in [Table 7-3](#). To control which elements are stored in the buffer, simply assign the desired attribute to the specific buffer. Control examples for Channel A, Buffer 1 are shown in [Table 7-4 on page 7-8](#), while read-only attribute programming examples are listed in [Table 7-5 on page 7-8](#).

**NOTE** You must clear the buffer using the `smuX.nvbufferX.clear()` command before changing buffer control attributes.

Table 7-2  
**Buffer storage control attributes**

Storage attribute	Description
appendmode	The append modes are either off (default) or on. When the append mode is off, a new measurement to this buffer will overwrite the previous contents. When the append mode is on, the first new measurement will be stored at what was formerly rb[n+1].
collectsourcevalues	When this attribute is on, source values will be stored with readings in the buffer. This requires 4 extra bytes of storage per reading. This value, off (default) or on, can only be changed when the buffer is empty.
collecttimestamps	When this attribute is on, timestamps will be stored with readings in the buffer. This requires 4 extra bytes of storage per reading. This value, off (default) or on, can only be changed when the buffer is empty.
timestampresolution	The timestamp resolution, in seconds. The default resolution is 0.000001 seconds. At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. This value can be increased for very long tests. Note: The minimum resolution setting is 1µs (0.000001 seconds).

Table 7-3  
**Buffer read-only attributes**

Storage attribute	Description
basetimestamp	The timestamp of when the reading at rb[1] was stored, in seconds from powerup.
capacity	The total number of readings that can be stored in the reading buffer.
n	The number of readings in the reading buffer.

Table 7-4  
**Buffer control programming examples**

Command	Description
<code>smua.nvbuffer1.collectsourcevalues = 1</code>	Enable source value storage.
<code>smua.nvbuffer1.appendmode = 1</code>	Enable buffer append mode.
<code>smua.nvbuffer1.collecttimestamps = 0</code>	Disable timestamp storage.
<code>smua.nvbuffer1.timestampresolution = 0.001</code>	Set timestamp resolution to 0.001s.

Table 7-5  
**Buffer read-only attribute programming examples**

Command	Description
<code>number = smua.nvbuffer1.n</code>	Request number of readings in buffer.
<code>buffer_size = smua.nvbuffer1.capacity</code>	Request buffer size.

### Buffer reading attributes

Attributes that control which elements are recalled from the buffer are listed in [Table 7-6](#). To access specific elements, simply append the desired attribute to the buffer designation.

For example, the following would return 100 Channel A readings from buffer 1:

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

Similarly, the following would return 100 Channel A source values from buffer 1:

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

Note that `readings` is the default reading attribute and can be omitted. Thus, the following would also return 100 Channel A readings from buffer 1:

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

Table 7-6  
**Recall attributes**

Recall attribute <sup>1</sup>	Description
<code>measurefunctions</code>	An array (a LUA table) of strings indicating the function measured for the reading (Current, Voltage, Ohms or Watts).
<code>measureranges</code>	An array (a LUA table) of full scale range values for the measure range used when the measurement was made.
<code>readings</code>	An array (a LUA table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly, i.e., <code>rb[2]</code> and <code>rb.readings[2]</code> are the same value.
<code>sourcefunctions</code>	An array (a LUA table) of strings indicating the source function at the time of the measurement (Current or Voltage).
<code>sourceoutputstates</code>	An array (a LUA table) of strings indicating the state of the source (Off or On).
<code>sourceranges</code>	An array (a LUA table) of full scale range values for the source range used when the measurement was made.
<code>sourcevalues</code>	If enabled, an array (a LUA table) of the sourced values in effect at the time of the reading.
<code>statuses</code>	An array (a LUA table) of status values for all of the readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value (see <a href="#">Table 7-7</a> ).
<code>timestamps</code>	An array (a LUA table) of timestamps, in seconds, of when each reading occurred. These are relative to the <code>basetimestamp</code> for the buffer ( <a href="#">Table 7-3</a> ).

1. The default attribute is `readings` and can be omitted. For example, `smua.nvbuffer1` and `smua.nvbuffer1.readings` will both return readings from Channel A, buffer 1.

## Time and date values

All time and date values are represented as the number of seconds since the unit was powered on. The `os.clock()` function returns values in this format.

Representing time as the number of seconds will be referred to as “standard time format.” Note that because TSL numbers are floating point numbers, the precision of time and date values relative to power-on will decrease the longer the unit is powered on. This decrease in precision is approximately 0.06ppm of the total elapsed time. This precision is generally much better than the time base of the instrument and should not present any problems in practice. It is worth noting because the user can directly see the affects as compared to the less obvious time-base drift.

## Buffer status

The buffer reading status attribute can include the status information as a numeric value shown in [Table 7-7](#). To access status information, send the following command:

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

Table 7-7

### Buffer status bits

Bit	Name	Hex value	Description
B0	TBD	0x01	Reserved for future use.
B1	Overtemp	0x02	Over temperature condition.
B2	AutoRangeMeas	0x04	Measure range was auto ranged.
B3	AutoRangeSrc	0x08	Source range was auto ranged.
B4	4Wire	0x10	4W (remote) sense mode enabled.
B5	Rel	0x20	Rel applied to reading.
B6	Compliance1	0x40	Source function in compliance.
B7	Filtered	0x80	Reading was filtered.

## Dynamically allocated buffers

RAM reading buffers can also be allocated dynamically. The buffers are created and allocated with the `smuX.makebuffer(n)` command, where `n` is the number of readings the buffer can store. For example, the following command allocates a Channel A buffer named `mybuffer` that can store 100 readings:

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

Allocated buffers can be deleted as follows:

```
mybuffer = nil
```

Dynamically allocated reading buffers can be used interchangeably with the `smuX.nvbufferY` buffers that are described in [“Reading buffer designations” on page 7-6](#).

## Buffer programming examples

### Defined buffer example

The listing below shows a programming example for storing data using the pre-defined Buffer 1 for Channel A. The SourceMeter loops for voltages from 0.01V to 1V with 0.01V steps (essentially performing a staircase sweep), stores 100 current readings and source values in Buffer 1, and then recalls all 100 readings and source values.

```

smua.reset()                                --Restore Model 260x defaults.
display.screen = 0                           --Select Channel A display.
display.smua.measure.func =                 --Display current.
display.MEASURE_DCAMPS
smua.measure.autorangei = smua.AUTORANGE_ON --Select measure I auto range.
format.data = format.ASCII                  --Select ASCII data format.
smua.nvbuffer1.clear()                      --Clear Buffer 1.
smua.nvbuffer1.appendmode = 1               --Enable append buffer mode.
smua.nvbuffer1.collectsourcevalues = 1      --Enable source value storage.
smua.measure.count = 1                      --Set count to 1.
smua.source.func = smua.OUTPUT_DCVOLTS     --Select source voltage function.
smua.source.levelv = 0.0                   -- Set bias voltage to 0V.
smua.source.output =smua.OUTPUT_ON         --Turn on output.
for v = 0.01, 1.0, 0.01 do                  --Loop for voltages from 0.01 to 1V.
    smua.source.levelv = v                  --Set source voltage.
    smua.measure.i(smua.nvbuffer1)         --Measure current, store in buffer.
    waitcomplete()                          --Wait for reading to complete.
end                                           --End loop.
smua.source.output =smua.OUTPUT_OFF        --Turn off output.
printbuffer(1, 100, smua.nvbuffer1.readings) --Return readings 1-100.
printbuffer(1, 100,                          --Return source values 1-100.
smua.nvbuffer1.sourcevalues)

```

## Dual buffer example

The listing below shows a programming example for storing both current and voltage readings using buffer 1 for current and buffer 2 to store voltage readings. The SourceMeter stores 100 current and voltage readings and then recalls all 100 sets of readings.

```
smua.reset()
smua.measure.autorangei = smua.AUTORANGE_ON
smua.measure.autorangev = smua.AUTORANGE_ON
format.data = format.ASCII
smua.nvbuffer1.clear()
smua.nvbuffer2.clear()
smua.measure.count = 100
smua.measure.interval = 0.1
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.levelv = 1
smua.source.output = smua.OUTPUT_ON
smua.measure.overlappediv
(smua.nvbuffer1, smua.nvbuffer2)
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
printbuffer(1, 100, smua.nvbuffer1)
printbuffer(1, 100, smua.nvbuffer2)
```

--Restore Model 260x defaults.  
--Select measure I auto range.  
--Select measure V auto range.  
--Select ASCII data format.  
--Clear buffer 1.  
--Clear buffer 2.  
--Set buffer count to 100.  
--Set measure interval to 0.1s.  
--Select source voltage function.  
--Output 1V.  
--Turn on output.  
--Store current readings in buffer 1,  
voltage readings in buffer 2.  
--Wait for buffer to fill.  
--Turn off output.  
--Return buffer 1 readings 1-100.  
--Return buffer 2 readings 1-100.



## Dynamically allocated buffer example

The listing below shows a programming example for storing data using an allocated buffer called `mybuffer` for Channel A. The SourceMeter stores 100 current readings in `mybuffer` and then recalls all the readings.

```
smua.reset()
smua.measure.autorangei = smua.AUTORANGE_ON
format.data = format.ASCII
mybuffer = smua.makebuffer(100)
smua.measure.count = 100
smua.measure.interval = 0.1
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.levelv = 1
smua.source.output = smua.OUTPUT_ON
smua.measure.overlappedi(mybuffer)
waitcomplete()
smua.source.output = smua.OUTPUT_OFF
printbuffer(1, 100, mybuffer)
mybuffer = nil
```

- Restore Model 260x defaults.
- Select measure auto range.
- Select ASCII data format.
- Allocate mybuffer, 100 readings.
- Set buffer count to 100.
- Set measure interval to 0.1s.
- Select source voltage function.
- Output 1V.
- Turn on output.
- Store current readings in mybuffer.
- Wait for buffer to fill.
- Turn off output.
- Return readings 1-100 from mybuffer.
- Delete mybuffer.

# Source-Measure Concepts

---

## Section 8 topics

**Overview**, page 8-2

**Compliance limit**, page 8-2

Maximum compliance, page 8-2

Compliance principles, page 8-3

**Sweep waveforms**, page 8-4

Staircase sweeps, page 8-4

Pulse sweeps, page 8-5

**Overheating protection**, page 8-6

Power equations to avoid overheating, page 8-6

Power calculations, page 8-6

**Operating boundaries**, page 8-7

Source or sink, page 8-7

I-Source operating boundaries, page 8-8

V-Source operating boundaries, page 8-12

Source I measure I, source V measure V,  
page 8-16

**Basic circuit configurations**, page 8-16

Source I, page 8-16

Source V, page 8-17

Measure only (V or I), page 8-18

**Guard**, page 8-20

Guard overview, page 8-20

Guard connections, page 8-20

**Pulse concepts**, page 8-23

Pulse period, page 8-23

Pulse rise and fall times, page 8-23

Pulse duty cycle, page 8-24

## Overview

The documentation in this section provides detailed information on source-measure concepts and includes the following information:

- “Compliance limit”, page 8-2
- “Sweep waveforms”, page 8-4
- “Overheating protection”, page 8-6
- “Operating boundaries”, page 8-7
- “Basic circuit configurations”, page 8-16
- “Guard”, page 8-20
- “Pulse concepts”, page 8-23

## Compliance limit

When sourcing voltage, the SourceMeter can be set to limit current (from 100pA to 3A). Conversely, when sourcing current, the SourceMeter can be set to limit voltage (from 100nV to 40V). The SourceMeter output will not exceed the compliance limit, except for the condition described in [“Compliance limit” on page 4-3](#).

### Maximum compliance

The maximum compliance values for the source ranges are summarized in [Table 8-1](#).

Table 8-1  
**Maximum compliance values**

Source range	Maximum compliance value
100mV	3A
1V	3A
6V	3A
40V	1A
100nA	40V
1μA	40V
10μA	40V
100μA	40V
1mA	40V
10mA	40V
100mA	40V
1A	40V
3A	6V

## Compliance principles

Compliance acts as a clamp. If the output reaches the compliance value, the SourceMeter will attempt to prevent the output from exceeding that value. This action implies that the source will switch from a V-source to an I-source (or from an I-source to a V-source) when in compliance.

As an example, assume the following:

SourceMeter:  $V_{SRC} = 10V$ ;  $I_{CMPL} = 10mA$

DUT resistance:  $10\Omega$

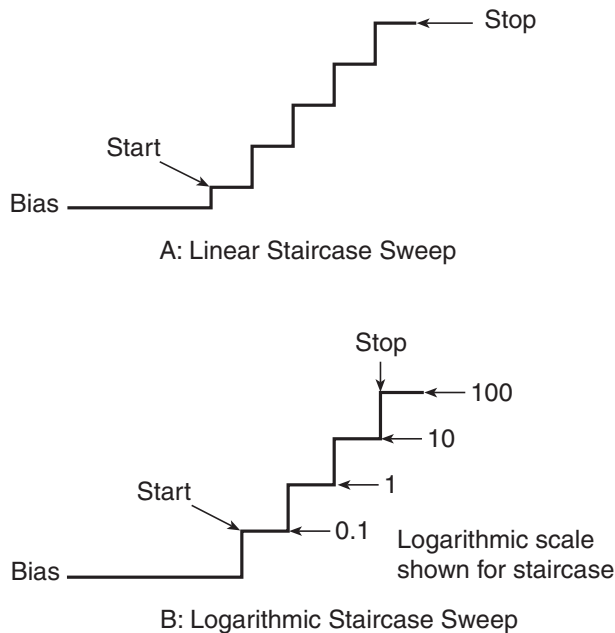
With a source voltage of 10V and a DUT resistance of  $10\Omega$ , the current through the DUT should be:  $10V/10\Omega = 1A$ . However, because the compliance is set to 10mA, the current will not exceed that value, and the voltage across the resistance is limited to 100mV. In effect, the 10V voltage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

# Sweep waveforms

## Staircase sweeps

There are two basic staircase sweeps: linear staircase and logarithmic staircase as shown in [Figure 8-1](#). The linear staircase sweep goes from the start level to the stop level in equal linear steps. The logarithmic staircase sweep is similar except it functions on a log scale with a specified number of steps per decade. See [Section 5](#) for more details on sweep operation.

Figure 8-1  
**Two basic sweep waveforms**



Typical applications for staircase sweeps include: I-V curves for two- and three-terminal semiconductor devices, characterization of leakage versus voltage, and semiconductor breakdown.

## Pulse sweeps

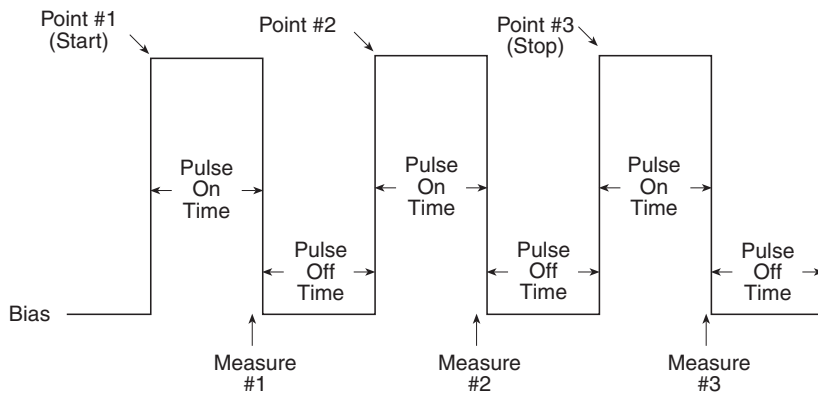
The Model 260x can also perform the following pulse sweeps:

- Fixed voltage pulse
- Fixed current pulse

These sweeps are similar to those discussed above, except, the source level at each sweep step is a pulse instead of a constant level. The pulse width (on time) and pulse delay (off time) can be programmed for each type of pulse mode sweep. [Figure 8-2](#) shows an example of a three-pulse sweep. See [“Pulse sweeps” on page 5-6](#) and [“Pulse concepts” on page 8-23](#) for more information.

Pulse sweeps are used in applications where thermal response is measured or where sustained power levels can damage the external Device Under Test (DUT).

Figure 8-2  
**Pulse sweep example**



# Overheating protection

Proper ventilation is required to keep the SourceMeter from overheating. The SourceMeter has an over-temperature protection circuit that will turn the output off in the event that the instrument overheats. If the output trips due to overheating, a message indicating this condition will be displayed. You will not be able to turn the output back on until the instrument cools down.

## Power equations to avoid overheating

Power equations can be used to determine if the SourceMeter will overheat<sup>1</sup>. For the following power equations:

$I$  = Output current

$V_{EXT}$  = External power source voltage (quadrant 2 and 4 operation, [Figure 8-3](#))

$D_C^2$  = Duty cycle (0 to 1; for example, 50% = 0.5)

$T_{amb}$  = Ambient temperature (0° to 50°C)

$T_{der}$  = Number of degrees C above 30°C (The SourceMeter output is derated 1W/°C above 30°C to 50°C.)

## Power calculations

When using the 40V and 6V ranges, use the appropriate equation below to determine if the SourceMeter will overheat.

**For the 6V range:**

$$\left| (18 + V_{EXT}) I \sqrt{D_C} \right| \leq 55 - T_{der}$$

**For the 40V range:**

$$\left| (5.7 + V_{EXT}) I \sqrt{D_C} \right| \leq 55 - T_{der}$$

- 
- Equations apply to both channels, sinking or sourcing power simultaneously.
  - If a duty cycle less than 1 is required for the equation, the maximum on time must be less than 10 seconds.

Return to [Section 8 topics](#)

# Operating boundaries

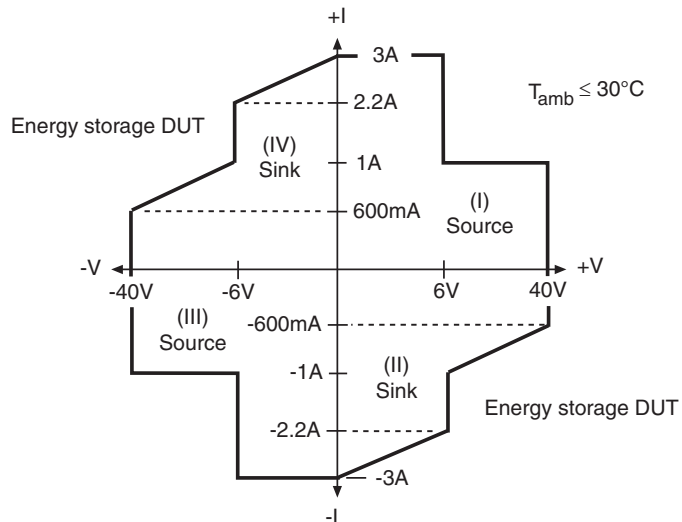
## Source or sink

Depending on how it is programmed and what is connected to the output (load or source), the SourceMeter can operate in any of the four quadrants. The four quadrants of operation are shown in Figure 8-3. When operating in the first (I) or third (III) quadrant, the SourceMeter is operating as a source (V and I have the same polarity). As a source, the SourceMeter is delivering power to a load.

When operating in the second (II) or fourth (IV) quadrant, the SourceMeter is operating as a sink (V and I have opposite polarity). As a sink, it is dissipating power rather than sourcing it. An external source or an energy storage device, such as a capacitor or battery, can force operation in the sink region.

The general operating boundaries for continuous power output are shown in Figure 8-3. (See “Power calculations” on page 8-6 for derating factors.) In this drawing, the 3A, 6V and 1A, 40V magnitudes are nominal values. Also note that the boundaries are not drawn to scale.

Figure 8-3  
Continuous power operating boundaries





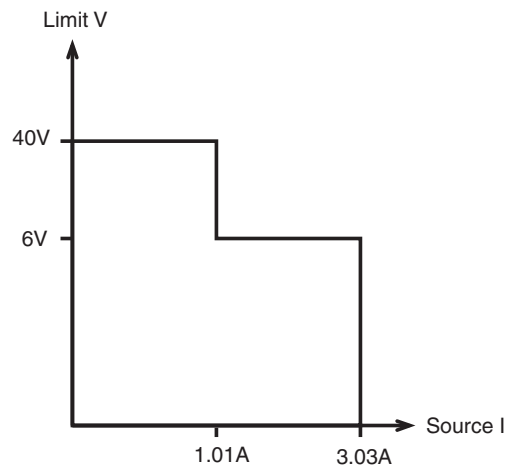
## I-Source operating boundaries

Figure 8-4 shows the operating boundaries for the I-Source. Only the first quadrant of operation is covered. Operation in the other three quadrants is similar.

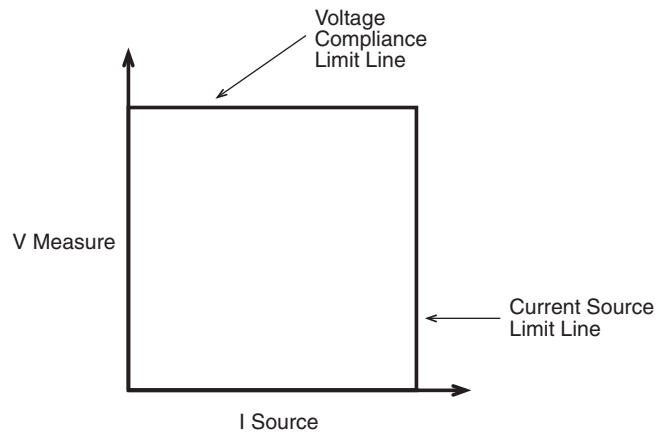
Figure 8-4A shows the output characteristics for the I-Source. As shown, the SourceMeter can output up to 1.01A at 40V, or 3.03A at 6V. Note that when sourcing more than 1.01A, voltage is limited to 6V.

Figure 8-4B shows the limit lines for the I-Source. The current source limit line represents the maximum source value possible for the presently selected current source range. The voltage compliance limit line represents the actual compliance that is in effect. Remember that compliance can be real or range. See “[Compliance limit](#)” on page 8-2. These limit lines are boundaries that represent the operating limits of the SourceMeter for this quadrant of operation. The operating point can be anywhere inside (or on) these limit lines. The limit line boundaries for the other quadrants are similar.

Figure 8-4  
**I-Source boundaries**



A) Output Characteristics



B) Limit Lines

The boundaries the SourceMeter operates in depends on the load (DUT) that is connected to its output. [Figure 8-5](#) shows operation examples for resistive loads that are  $50\Omega$  and  $200\Omega$ , respectively. For these examples, the SourceMeter is programmed to source  $100\text{mA}$  and limit  $10\text{V}$ .

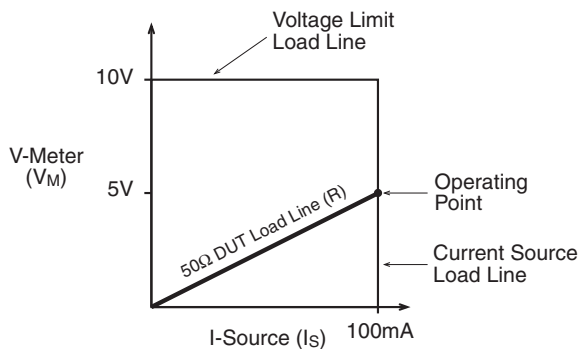
In [Figure 8-5A](#), the SourceMeter is sourcing  $100\text{mA}$  to the  $50\Omega$  load and subsequently measures  $5\text{V}$ . As shown, the load line for  $50\Omega$  intersects the  $100\text{mA}$  current source line at  $5\text{V}$ .

[Figure 8-5B](#) shows what happens if the resistance of the load is increased to  $200\Omega$ . The DUT load line for  $200\Omega$  intersects the voltage compliance limit line placing the SourceMeter in compliance. In compliance, the SourceMeter will not be able to source its programmed current ( $100\text{mA}$ ). For the  $200\Omega$  DUT, the SourceMeter will only output  $50\text{mA}$  (at the  $10\text{V}$  limit).

Notice that as resistance increases, the slope of the DUT load line increases. As resistance approaches infinity (open output), the SourceMeter will source virtually  $0\text{mA}$  at  $10\text{V}$ . Conversely, as resistance decreases, the slope of the DUT load line decreases. At zero resistance (shorted output), the SourceMeter will source  $100\text{mA}$  at virtually  $0\text{V}$ .

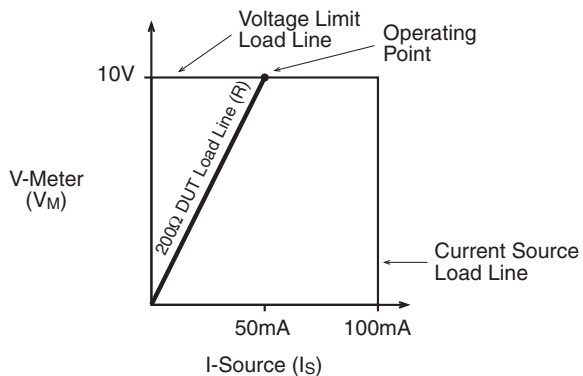
Regardless of the load, voltage will never exceed the programmed compliance of  $10\text{V}$ .

Figure 8-5  
**I-Source operating boundaries**



$$\begin{aligned} V\text{-Meter} &= I_S \cdot R \\ &= (100\text{mA}) (50\Omega) \\ &= 5\text{V} \end{aligned}$$

A) Normal I-source operation



$$\begin{aligned} I_S &= V_M / R \\ &= 10\text{V} / 200\Omega \\ &= 50\text{mA} \end{aligned}$$

B) I-source in compliance

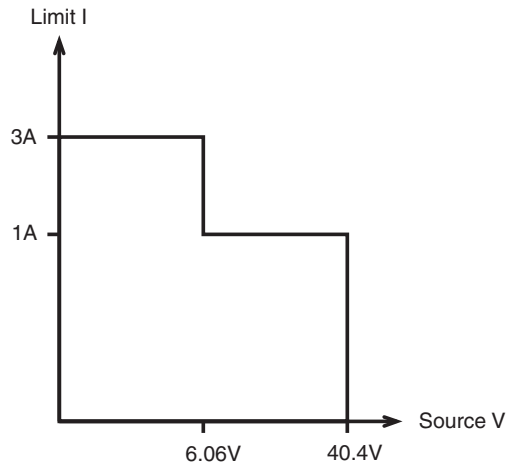
## V-Source operating boundaries

Figure 8-6 shows the operating boundaries for the V-Source. Only the first quadrant of operation is covered. Operation in the other three quadrants is similar.

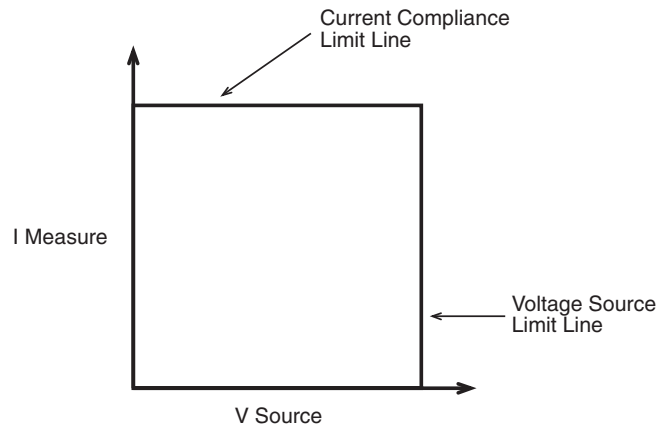
Figure 8-6A shows the output characteristics for the V-Source. As shown, the SourceMeter can output up to 6.06V at 3A, or 40.4V at 1A. Note that when sourcing more than 6.06V, current is limited to 1A.

Figure 8-6B shows the limit lines for the V-Source. The voltage source limit line represents the maximum source value possible for the presently selected voltage source range. For example, if on the 6V source range, the voltage source limit line is at 6.3V. The current compliance limit line represents the actual compliance in effect. Remember that compliance can be real or range. See [“Compliance limit” on page 8-2](#). These limit lines are boundaries that represent the operating limits of the SourceMeter for this quadrant of operation. The operating point can be anywhere inside (or on) these limit lines. The limit line boundaries for the other quadrants are similar.

Figure 8-6  
**V-Source boundaries**



A) Output characteristics



B) Limit lines

The boundaries the SourceMeter operates in depends on the load (DUT) that is connected to the output. [Figure 8-7](#) shows operation examples for resistive loads that are  $2\text{k}\Omega$  and  $800\Omega$ , respectively. For these examples, the SourceMeter is programmed to source  $10\text{V}$  and limit  $10\text{mA}$ .

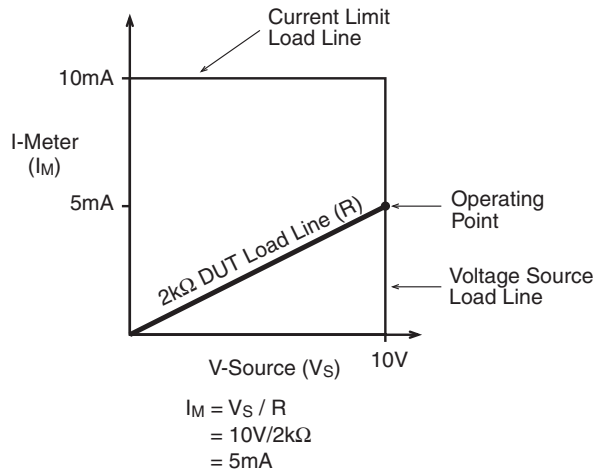
In [Figure 8-7A](#), the SourceMeter is sourcing  $10\text{V}$  to the  $2\text{k}\Omega$  load and subsequently measures  $5\text{mA}$ . As shown, the load line for  $2\text{k}\Omega$  intersects the  $10\text{V}$  voltage source line at  $5\text{mA}$ .

[Figure 8-7B](#) shows what happens if the resistance of the load is decreased to  $800\Omega$ . The DUT load line for  $800\Omega$  intersects the current compliance limit line placing the SourceMeter in compliance. In compliance, the SourceMeter will not be able to source its programmed voltage ( $10\text{V}$ ). For the  $800\Omega$  DUT, the SourceMeter will only output  $8\text{V}$  (at the  $10\text{mA}$  limit).

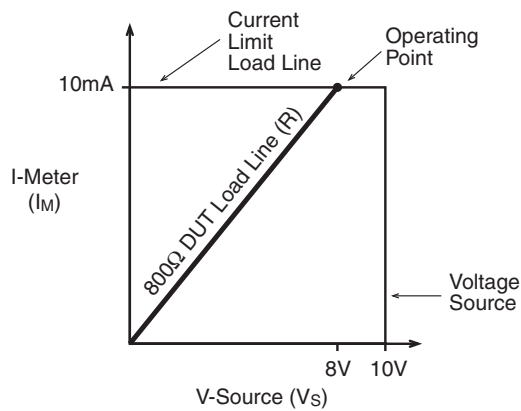
Notice that as resistance decreases, the slope of the DUT load line increases. As resistance approaches infinity (open output), the SourceMeter will source virtually  $10\text{V}$  at  $0\text{mA}$ . Conversely, as resistance increases, the slope of the DUT load line decreases. At zero resistance (shorted output), the SourceMeter will source virtually  $0\text{V}$  at  $10\text{mA}$ .

Regardless of the load, current will never exceed the programmed compliance of  $10\text{mA}$ .

Figure 8-7  
**V-Source operating examples**



A) Normal V-source operation



B) V-Source in compliance



## Source I measure I, source V measure V

The SourceMeter can measure the function it is sourcing. When sourcing a voltage, you can measure voltage. Conversely, if you are sourcing current, you can measure the output current. For these measure source operations, the measure range is the same as the source range.

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

## Basic circuit configurations

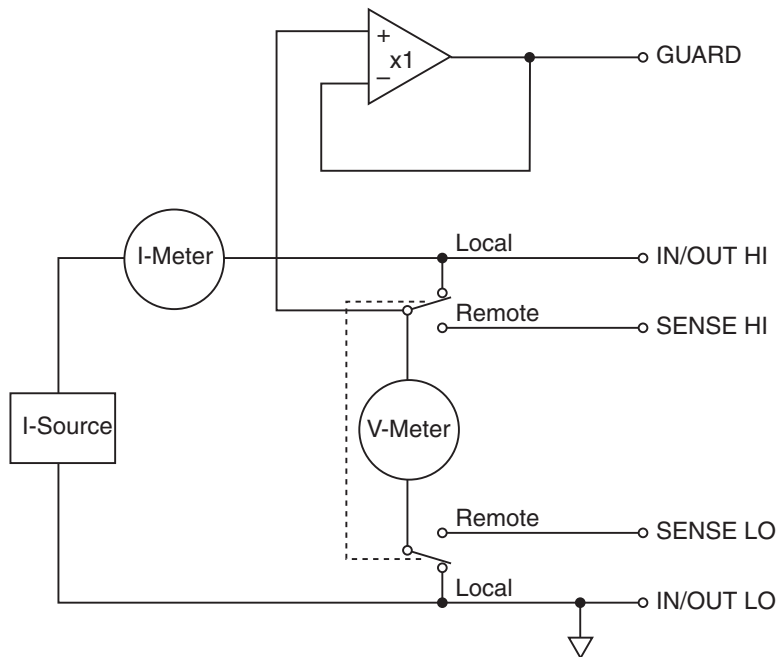
### Source I

When configured to source current (I-Source) as shown in [Figure 8-8](#), the SourceMeter functions as a high-impedance current source with voltage limit capability and can measure current (I-Meter) or voltage (V-Meter).

For 2-wire local sensing, voltage is measured at the Input/Output terminals of the SourceMeter. For 4-wire remote sensing, voltage is measured directly at the DUT using the sense terminals. This eliminates any voltage drops that may be in the test leads or connections between the SourceMeter and the DUT.

The current source does not require or use the sense leads to enhance current source accuracy. With 4-wire remote sensing selected, the sense leads must be connected or incorrect operation will result.

Figure 8-8  
**Source I configuration**



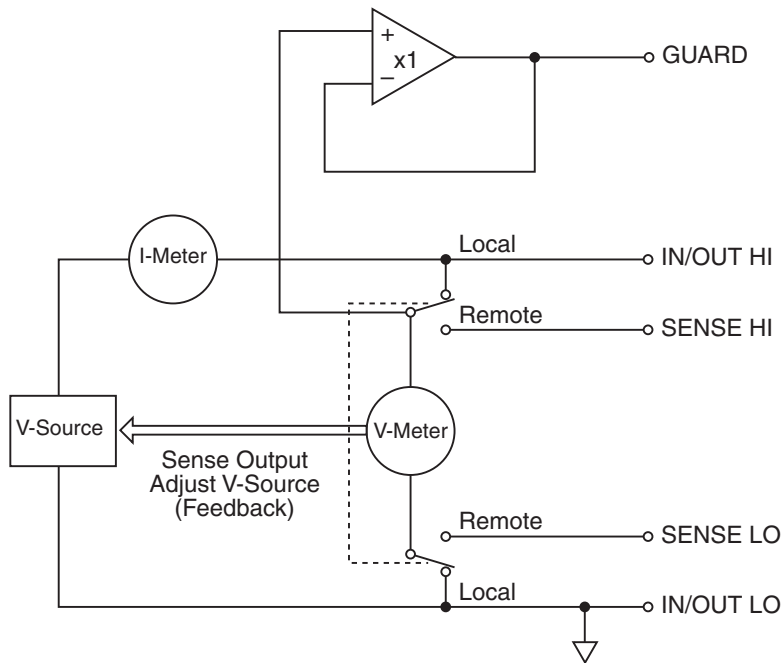
## Source V

When configured to source voltage (V-Source) as shown in [Figure 8-9](#), the SourceMeter functions as a low-impedance voltage source with current limit capability and can measure current (I-Meter) or voltage (V-Meter).

Sense circuitry is used to continuously monitor the output voltage and make adjustments to the V-Source as needed. The V-Meter senses the voltage at the input/output terminals (2-wire local sense) or at the DUT (4-wire remote sense using the sense terminals) and compares it to the programmed voltage level. If the sensed level and the programmed value are not the same, the V-Source is adjusted accordingly. Remote sense eliminates the effect of voltage drops in the test leads ensuring that the exact programmed voltage appears at the DUT.

The voltage error feedback to the V-Source is an analog function. The source error amplifier is used to compensate for IR drop in the test leads.

Figure 8-9  
**Source V configuration**



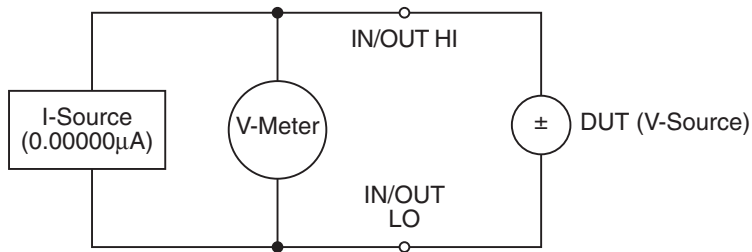
## Measure only (V or I)

Figure 8-10 shows the configurations for using the SourceMeter exclusively as a voltmeter or ammeter. As shown in Figure 8-10A, the SourceMeter is configured to measure voltage-only by setting it to source 0A and measure voltage.

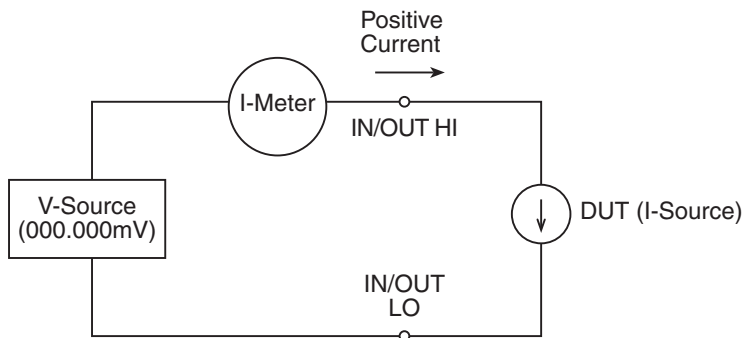
**CAUTION** V-Compliance must be set to a level that is higher than the measured voltage. Otherwise, excessive current will flow into the SourceMeter. This current could damage the SourceMeter. Also, when connecting an external voltage to the I-Source, set the output off state to the high-impedance mode. See “Compliance limit” on page 8-2 for details.

In Figure 8-10B, the SourceMeter is configured to measure current-only by setting it to source 0V and measure current. Note that in order to obtain positive (+) readings, conventional current must flow from IN/OUT HI to LO.

Figure 8-10  
**Measure only configurations**



**A. Measure Voltage Only**



**Note:** Positive current flowing out of IN/OUT HI results in positive (+) measurements.

**B. Measure Current Only**

Note: Use 2-wire local sensing.

# Guard

**WARNING** **GUARD is at the same potential as output HI. Thus, if hazardous voltages are present at output HI, they are also present at the GUARD terminal.**

## Guard overview

The driven guard (available at the rear panel GUARD terminals) is always enabled and provides a buffered voltage that is at the same level as the Input/Output HI (or Sense HI for remote sense) voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between input/output high and low. In the absence of a driven guard, leakage in the external test circuit could be high enough to adversely affect the performance of the SourceMeter.

Leakage current can occur through parasitic or non-parasitic leakage paths. An example of parasitic resistance is the leakage path across the insulator in a coax or triax cable. An example of non-parasitic resistance is the leakage path through a resistor that is connected in parallel to the DUT.

## Guard connections

Guard is typically used to drive the guard shields of cables and test fixtures. Guard is extended to a test fixture from the cable guard shield. Inside the test fixture, the guard can be connected to a guard plate or shield that surrounds the DUT.

**WARNING** **To prevent injury or death, a safety shield must be used to prevent physical contact with a guard plate or guard shield that is at a hazardous potential (>30Vrms or 42.4V peak). This safety shield must completely enclose the guard plate or shield and must be connected to safety earth ground. [Figure 8-11B](#) shows the metal case of a test fixture being used as a safety shield.**

**NOTE** See [Section 3](#) for details on guarded test connections.

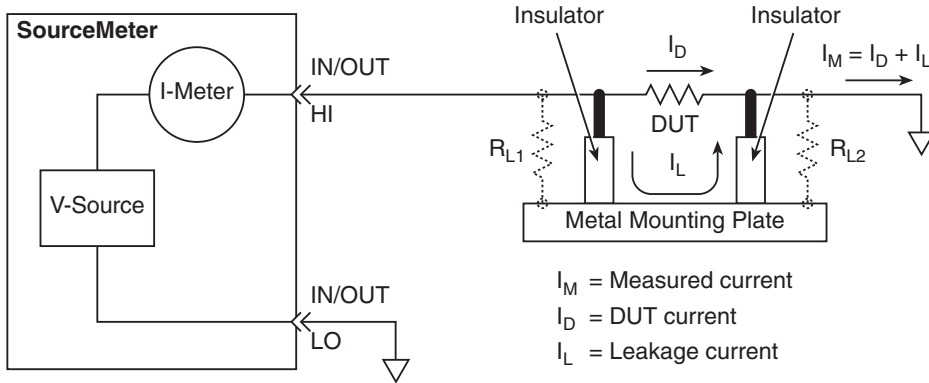
Inside the test fixture, a triaxial cable can be used to extend guard to the DUT. The center conductor of the cable is used for In/Out HI, the inner shield is used for guard, and the outer shield is used for In/Out LO and is connected to the safety shield (which is connected to safety earth ground).

A coaxial cable can be used if the guard potential does not exceed 30Vrms (42.4V peak). The center conductor is used for In/Out HI, and the outer shield is used for guard. For higher guard potentials, use a triaxial cable as previously explained.

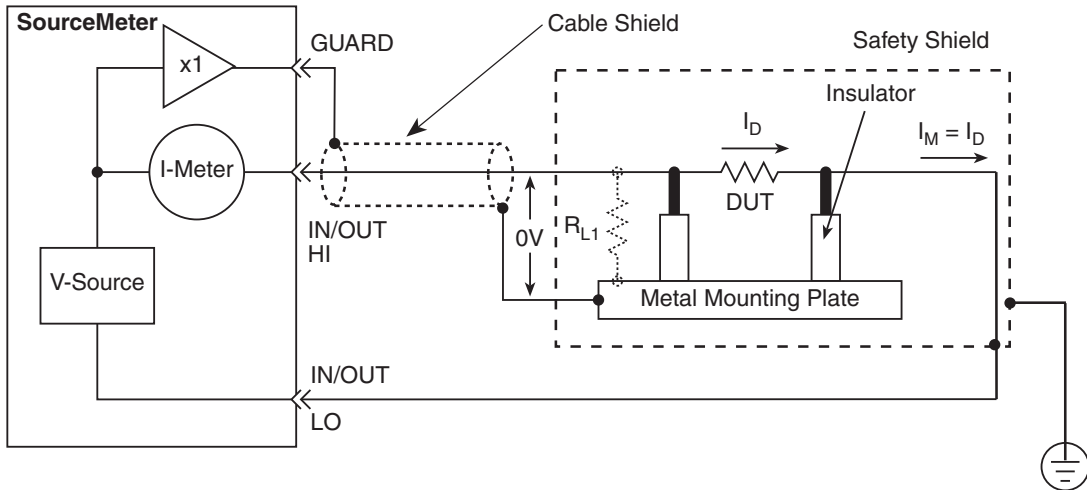
[Figure 8-11](#) shows how cable guard can eliminate leakage current through the insulators in a test fixture. In [Figure 8-11A](#), leakage current ( $I_L$ ) flows through the insulators ( $R_{L1}$  and  $R_{L2}$ ) to In/Out LO, adversely affecting the low-current (or high-resistance) measurement of the DUT.

In [Figure 8-11B](#), the driven guard is connected to the cable shield and extended to the metal guard plate for the insulators. Since the voltage on either end of  $R_{L1}$  is the same (0V drop), no current can flow through the leakage resistance path. Thus, the SourceMeter only measures the current through the DUT.

Figure 8-11  
**Comparison of unguarded and guarded measurements**



**A. Unguarded**



**B. Guarded**

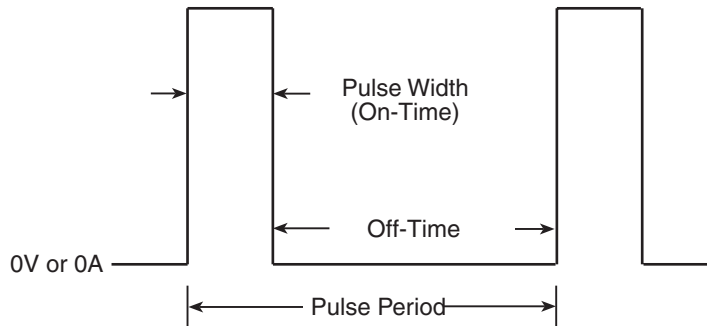
## Pulse concepts

Using factory scripts, the Model 260x can perform fixed, linear staircase, and logarithmic staircase pulse sweeps (see [“Pulse sweeps”](#) on page 5-6 for more information). The following paragraphs discuss pulse period, rise and fall times, and duty cycle. See the specifications in Appendix A for details on source transient response and settling times.

### Pulse period

As shown in [Figure 8-12](#), the pulse period is the sum of the pulse on time (pulse width) and the pulse off time. When the pulse is off, the output assumes a 0V or 0A level, depending on the function used. When the pulse is on, the output assumes the programmed current or voltage source value. For the fixed pulse sweep, the amplitude of each pulse is the same. For the staircase sweeps, each pulse assumes the programmed sweep step value. Pulse on and off times can be separately programmed for each type of pulse sweep.

Figure 8-12  
**Pulse period**

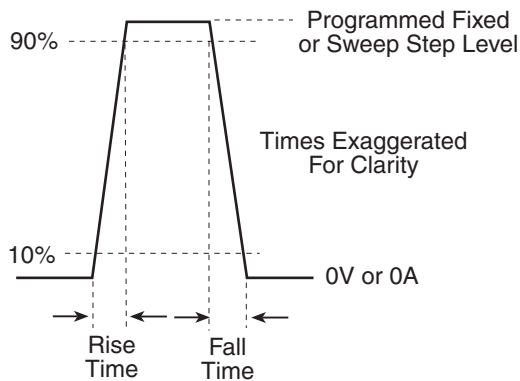


### Pulse rise and fall times

As shown in [Figure 8-13](#), the pulse rise time is the interval it takes the pulse to go from 10% of maximum value to 90% of maximum value. For the Model 260x, pulse rise and fall times depend on the transient response and source output settling times, which are in turn affected by the selected source range. See the specifications in [Appendix A](#) for details on transient response and source settling times.



Figure 8-13  
**Pulse rise and fall times**



## Pulse duty cycle

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

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

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

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

# 9

# System Expansion

## (TSP-Link)

---

### **Section 9 topics**

#### **Overview**, page 9-2

Master and Slaves, page 9-2

System configurations, page 9-2

#### **Connections**, page 9-3

#### **Initialization**, page 9-3

Assigning node numbers, page 9-3

Resetting the TSP-Link, page 9-4

#### **Using the expanded system**, page 9-6

Accessing nodes, page 9-6

System behavior, page 9-7

Abort, page 9-7

# Overview

The TSP-Link is an expansion interface that allows the instruments to communicate with each other. The test system can be expanded to include up to 64 TSP-Link-enabled instruments.

## Master and Slaves

In a TSP-Link system, one of the nodes (instruments) is the Master and the other nodes are the Slaves.

The Master can control the other nodes (Slaves) in the system. When any node transitions from local operation to remote, it becomes the Master of the system; all other nodes also transition to remote operation, and become its Slaves. When any node transitions from remote operation to local, all other nodes also transition to local operation, and the Master/Slave relationship between nodes is dissolved. For more information about remote and local operations, see “[Differences: remote vs. local state](#)” in [Section 2](#).

A Slave is a node that is controlled by the Master. The GPIB and RS-232 command interfaces of the Slaves are disabled.

## System configurations

A TSP-Link system can be used without a PC (stand-alone system) or as a PC-based system.

**Stand-alone system** – In a stand-alone system, scripts that control the system are executed from the front panel of one of the instruments. No PC connection is required. In [Figure 9-1](#), a script can be run from the front panel of any one of the instruments.

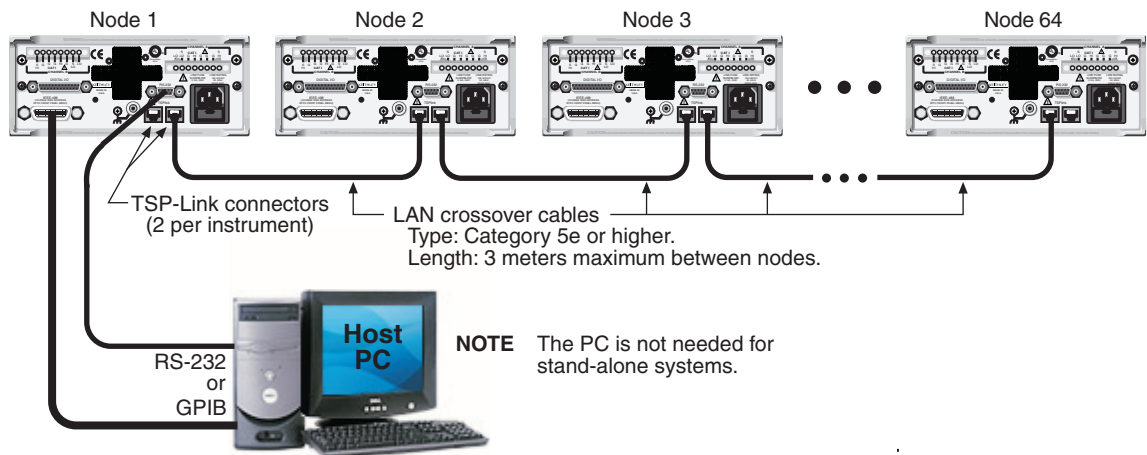
**PC-based system** – In a PC-based system, the GPIB or RS-232 interface to any single node becomes the interface to the entire system. In [Figure 9-1](#), the system can be controlled via the GPIB or RS-232 interface of Node 1.

# Connections

Connections for an expanded system are shown in [Figure 9-1](#). As shown, one unit is optionally connected to the PC using the GPIB or RS-232 interface. Details on these PC communication connections are covered in [Section 2](#).

As shown in [Figure 9-1](#), all the units in the system are daisy-chained together using LAN crossover cables.

Figure 9-1  
TSP-Link connections



## Initialization

Before a TSP-Link system can be used, it must be initialized. For initialization to succeed, each instrument in a TSP-Link system must be assigned a different node number.

### Assigning node numbers

At the factory, each Model 260x instrument is assigned as Node 1. The node number for each unit is stored in its non-volatile memory and will not be lost when the instrument is turned off.

## Front panel operation

The TSP-Link node number can be set from the MAIN MENU of each instrument; this process is summarized in [Table 9-1](#).

Table 9-1

### Assigning a node number to an instrument from the front panel

Model 260x MAIN MENU
<ol style="list-style-type: none"> <li>1) Press the <b>MENU</b> key to access <b>MAIN MENU</b>.</li> <li>2) Select the <b>COMMUNICATION</b> menu.</li> <li>3) Select the <b>TSPLINK_CFG</b> menu.</li> <li>4) Select the <b>NODE</b> menu.</li> <li>5) Set the <b>NODE</b> number (<b>01</b> to <b>64</b>) for the instrument.</li> </ol>

## Remote programming

The `tsplink.node` attribute is used to set the node number for an instrument:

```
tsplink.node = N
  where: N = 1 to 64
```

The node number of an instrument can be determined by reading the `tsplink.node` attribute as follows:

```
print (tsplink.node)
```

The above `print` command will output the node number. For example, if the node number is 1, the value `1.000000e00` will be output.

## Resetting the TSP-Link

After all the node numbers are set, you must initialize the system by performing a TSP-Link reset. For initialization to succeed, all units must be powered on when the TSP-Link reset is performed.

<b>NOTE</b>	<p><b>If you change the system topology after initialization, you must re-initialize the system by performing a TSP-Link reset. Changes that affect the system topology include:</b></p> <ul style="list-style-type: none"> <li>• <b>Powering down or rebooting any unit in the system.</b></li> <li>• <b>Rearranging or disconnecting the LAN cable connections between units.</b></li> </ul>
-------------	--

## Front panel operation

The TSP-Link reset can be performed from the MAIN MENU of any instrument; this process is summarized in [Table 9-2](#).

Table 9-2

### Resetting the TSP-Link from the front panel

Model 260x MAIN MENU
1) Press the <b>MENU</b> key to access <b>MAIN MENU</b> .
2) Select the <b>COMMUNICATION</b> menu.
3) Select the <b>TSPLINK-CFG</b> menu.
4) Select the <b>RESET</b> to initialize the TSP-Link.

## Remote programming

The commands associated with TSP-Link reset are listed in [Table 9-3](#).

Table 9-3

### TSP-Link reset commands

Command	Description
<code>tsplink.reset()</code>	Initializes the TSP-Link system.
<code>tsplink.state</code>	Returns <code>online</code> if the most recent TSP-Link reset was successful. Returns <code>offline</code> if the reset failed.

An attempted TSP-Link reset will fail if any of the following conditions are true:

- Two or more instruments in the system have the same node number.
- There are no other instruments connected to the unit performing the reset.
- One or more of the units in the system is not powered on.

**Programming example** – The following code will reset the TSP-Link and output its state:

```
tsplink.reset()
print(tsplink.state)
```

If the reset is successful, `online` will be returned to indicate that communications with all nodes have been established.

# Using the expanded system

## Accessing nodes

A TSP-Link reset creates the `node` table. Each unit in the system corresponds to an entry in this table, indexed by the unit's node number. The `node [N]` variable (where `N` is the node number) is used to access any node in the system. For example, Node 1 is represented in the `node` table as entry `node [1]`.

Each of these entries is, in turn, a table, holding all of the logical instruments (and associated ICL commands) shared by the corresponding unit (see “[Logical instruments](#)” on [page 12-5](#) for more details). SMU A on Node 1, therefore, could be accessed as `node [1] . smua`.

The `localnode` variable is an alias for `node [N]`, where `N` is the node number of the Master. For example, if Node 1 is the Master, `localnode` can be used instead of `node [1]`.

<b>NOTE</b> For remote programming, scripts that reside on Slave nodes are not accessible.
--

**Programming examples** – The following examples show how to access instruments in the TSP-Link system shown in [Figure 9-1](#):

- Any of the following three commands can be used to reset SMU A of Node 1 (which, in this example, is the Master). The other nodes in the system are not affected.

```
smua.reset ()
localnode.smua.reset ()
node [1] . smua.reset ()
```

- The following command will reset SMU A of Node 4, which is a Slave. The other nodes are not affected.

```
node [4] . smua.reset ()
```

## System behavior

### Using the `reset ()` command

While most TSP-Link operations target a single node in the system, the `reset ()` command affects the system as a whole. The `reset ()` command, by definition, resets all nodes to their default settings:

```
reset ()                -- Resets all nodes in a TSP-Link system.
```

`node [N]` and `localnode` can be used with `reset` to reset only one of the nodes. The other nodes are not affected. Examples:

```
node [1] .reset ()     -- Resets Node 1 only.  
localnode.reset ()    -- Resets Node 1 only.  
node [4] .reset ()    -- Resets Node 4 only.
```

### Abort

An `abort` will terminate an executing script and return all nodes to local operation (REM annunciators turn off), dissolving the Master/Slave relationships between nodes. An abort is invoked by either issuing an `abort` command to the Master or pressing the EXIT key on any node in the system.

An abort can also be performed by pressing the OUTPUT ON/OFF key on any node. The results are the same as above, with the addition that all SMU outputs in the system are turned off.



# 10

# Digital I/O and Output Enable

---

## **Section 10 topics**

**Overview**, page 10-2

**Digital I/O port**, page 10-2

Port configuration, page 10-2

Digital I/O configuration, page 10-4

Controlling digital I/O lines, page 10-4

**Output enable**, page 10-8

Overview, page 10-8

Operation, page 10-8

Control, page 10-9

## Overview

The documentation in this section provides detailed information on using the Digital I/O port and includes the following:

- “Digital I/O port”, page 10-2
- “Output enable”, page 10-8

## Digital I/O port

The SourceMeter has a digital input/output port that can be used to control external digital circuitry. For example, a handler that is used to perform binning operations can be used with a Digital I/O port.

### Port configuration

The Digital I/O Port is located on the rear panel and is shown in [Figure 10-1](#). Note that a standard female DB-25 connector is used with the Digital I/O port.

### Connecting cables

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

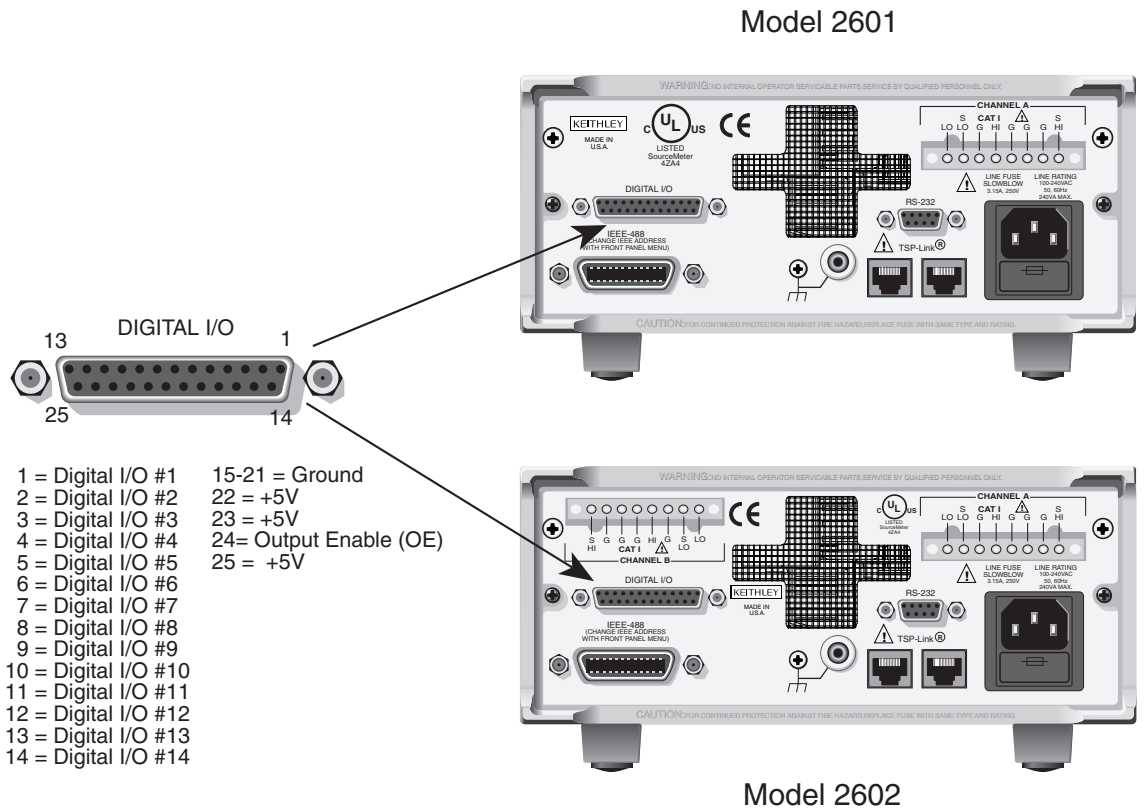
### Digital I/O lines

The port provides 14 digital I/O lines. Each output is set high (+5V) or low (0V) and can read high or low logic levels.

### +5V output

The Digital I/O Port provides a +5V output that is used to drive external logic circuitry. Maximum current output for this line is 600mA. This line is protected by a self-resetting fuse (one hour recovery time).

Figure 10-1  
Digital I/O port



### Output enable line

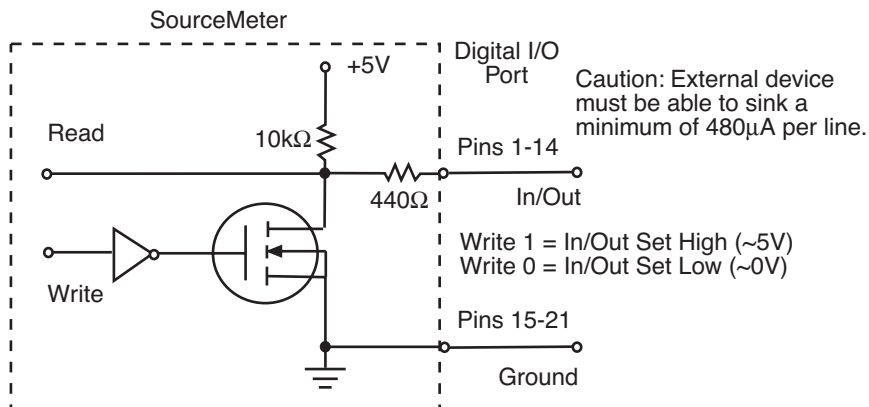
The OE (output enable) line of the Digital I/O can be used with a switch in the test fixture or component handler. With proper use, power is removed from the DUT when the lid of the fixture is opened. See [“Output enable” on page 10-8](#) for operation details.

**WARNING** When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages. The Digital I/O port of the 2600 Series SourceMeter is not suitable for control of safety circuits and should not be used to control a safety interlock.

## Digital I/O configuration

Figure 10-2 shows the basic configuration of the Digital I/O port. Writing a 1 to a line sets that line high ( $\sim +5V$ ). Writing a 0 to a line sets that line low ( $\sim 0V$ ). Note that an external device pulls an I/O line low by shorting it to ground, so that a device must be able to sink at least  $480\mu A$  per I/O line.

Figure 10-2  
Digital I/O port configuration



## Controlling digital I/O lines

Although the digital I/O lines are primarily intended for use with a device handler for limit testing, they can also be used for other purposes such as controlling external logic circuits. You can control lines either from the front panel or via remote as follows.

## Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in [Table 10-1](#).

Table 10-1

### Digital I/O bit weighting

Line #	Bit	Decimal weighting	Hexadecimal weighting
1	B1	1	0x0001
2	B2	2	0x0002
3	B3	4	0x0004
4	B4	8	0x0008
5	B5	16	0x0010
6	B6	32	0x0020
7	B7	64	0x0040
8	B8	128	0x0080
9	B9	256	0x0100
10	B10	512	0x0200
11	B11	1024	0x0400
12	B12	2048	0x0800
13	B13	4096	0x1000
14	B14	8192	0x2000

## Front panel digital I/O control

To enable the digital I/O lines:

1. Press the CONFIG key followed by the OUTPUT key.
2. Choose DIO-CONTROL, then press ENTER or the Rotary Knob.
3. Select ON or OFF as desired, then press ENTER or the Rotary Knob.
4. Press EXIT as needed to return to the normal display.

## Setting digital I/O values

To set digital I/O values:

1. Press the MENU key.
2. Select GENERAL, and then press ENTER or the Rotary Knob.
3. Choose DIGOUT, and then press ENTER or the Rotary Knob.
4. Select DIG\_IO\_OUTPUT, and then press ENTER or the Rotary Knob.
5. Set the decimal value as required to set digital I/O line(s) within the range of 0 to 16,383 (see [Table 10-1](#)), then press ENTER or the Rotary Knob.
6. Press EXIT as needed to return to the normal display.

## Write protecting digital I/O lines

You can also write protect specific digital I/O lines to prevent their values from being changed as follows:

1. Press the MENU key.
2. Select GENERAL, and then press ENTER or the Rotary Knob.
3. Choose DIGOUT, and then press ENTER or the Rotary Knob.
4. Select WRITE\_PROTECT, then press ENTER or the Rotary Knob.
5. Set the decimal value as required to write protect digital I/O line(s) within the range of 0 to 16,383 (see [Table 10-1](#)), then press ENTER or the Rotary Knob.
6. Press EXIT as needed to return to the normal display.
7. To remove write protection, simply repeat the above procedure, entering the same value.

## Remote digital I/O commands

Commands that control and access the digital I/O port are summarized in [Table 10-2](#). See [Section 12](#) for complete details on these commands. See [Table 10-1](#) for decimal and hexadecimal values used to control and access the digital I/O port and individual lines.

## Basic digital I/O commands

Use these commands to perform basic steady-state digital I/O operations such as reading and writing to individual I/O lines or reading and writing to the entire port.

### Digital I/O trigger commands

Use these commands to trigger the Model 260x using external trigger pulses applied to the digital I/O port, or to provide trigger pulses to external devices.

**NOTE** The digital I/O lines can be used for both input and output. You must write a 1 to all digital I/O lines that are to be used as inputs.

Table 10-2  
Digital I/O commands

Command	Description
<b>Commands for basic I/O:</b>	
<code>digio.readbit(bit)</code>	Read one digital I/O input line.
<code>digio.readport()</code>	Read digital I/O port.
<code>digio.writebit(bit, data)</code>	Write data to one digital I/O output line.
<code>digio.writeport(data)</code>	Write data to digital I/O port.
<code>digio.writeprotect = mask</code>	Write protect mask to digital I/O port.
<b>Commands for digital I/O triggering:</b>	
<code>digio.trigger[line].assert()</code>	Generate a trigger on digital I/O line.
<code>digio.trigger[line].clear()</code>	Clear the event detector for a trigger.
<code>digio.trigger[line].mode = mode</code>	Control I/O trigger event detector mode: <code>digio.TRIG_FALLING</code> (falling edge triggers). <code>digio.TRIG_RISING</code> (rising edge triggers). <code>digio.TRIG_EITHER</code> (falling or rising triggers). <code>digio.TRIG_SYNCHRONOUS</code> (detect and latch falling edge triggers).
<code>digio.trigger[line].pulsewidth = width</code>	Set trigger pulse width.
<code>digio.trigger[line].release()</code>	Release latched trigger.
<code>digio.trigger[line].wait(timeout)</code>	Wait for a trigger.

## Digital I/O programming examples

### Basic digital I/O programming example

The commands below set bit 1 of the digital I/O port high, and then read the entire port value.

```
digio.writebit(1,1)           --Set bit 1 high.
data = digio.readport()      --Read digital I/O port.
```

### Digital I/O trigger example

The commands below set the line 2 pulse width to 10 $\mu$ s, trigger mode to falling edge, and then assert a trigger pulse on that digital I/O line:

```
digio.trigger[2].pulsewidth = 1e-5--Set line 2 pulse width to 10 $\mu$ s.
digio.trigger[2].mode =         --Set line 2 mode to falling edge.
digio.TRIG_FALLING
digio.trigger[2].assert()      --Assert trigger on line 2.
```

# Output enable

## Overview

The Digital I/O Port provides an output enable line for use with a test fixture switch. When properly used, the output of the SourceMeter will turn OFF when the lid of the test fixture is opened. See [Section 3](#) for important safety information when using a test fixture.

**WARNING** When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages. The Digital I/O port of the 2600 Series SourceMeter is not suitable for control of safety circuits and should not be used to control a safety interlock.

## Operation

When enabled, the output of the SourceMeter can only be turned on when the output enable line is pulled high through a switch to +5V as shown in [Figure 10-3A](#). If the lid of the test fixture opens ([Figure 10-3B](#)), the switch opens, and the output enable line goes low, turning the output of the SourceMeter off. The output will not be automatically turned on when output enable is set high. The output cannot be turned back on until the output is set high.



## Control

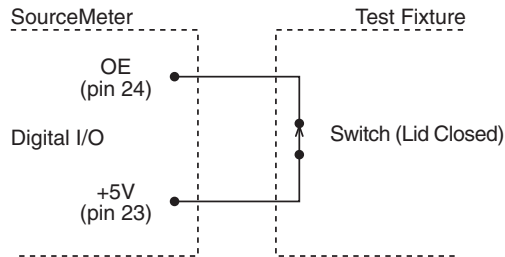
Use one of these commands to control output enable action:

```
smuX.source.outputenableaction = smuX.OE_NONE
smuX.source.outputenableaction = smuX.OE_OUTPUT_OFF
```

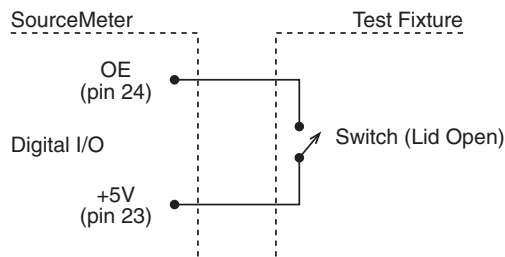
When set to `smuX.OE_NONE`, the SourceMeter will take no action when the output enable line goes low. When set to `smuX.OE_OUTPUT_OFF`, the SourceMeter will turn its output off as if the `smuX.source.output = smuX.OUTPUT_OFF` command had been received. The SourceMeter will not automatically turn its output on when the output enable line returns to the high state. For example, the following command enables output enable off action for Channel A:

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

Figure 10-3  
Using output enable



A. OUTPUT can be turned on.



B. OUTPUT cannot be turned on.

# 11

# Communications Interfaces

---

## Section 11 topics

**Overview**, page 11-2

**Selecting an interface**, page 11-2

**GPIB operation**, page 11-3

GPIB standards, page 11-3

GPIB connections, page 11-3

Primary address, page 11-5

Terminator, page 11-6

**General bus commands**, page 11-7

REN (remote enable), page 11-7

IFC (interface clear), page 11-7

LLO (local lockout), page 11-8

GTL (go to local), page 11-8

DCL (device clear), page 11-8

SDC (selective device clear), page 11-8

GET (group execute trigger), page 11-8

SPE, SPD (serial polling), page 11-8

**Front panel GPIB operation**, page 11-9

Error and status messages, page 11-9

GPIB status indicators, page 11-9

LOCAL key, page 11-10

**RS-232 interface operation**, page 11-10

Setting RS-232 interface parameters, page 11-10

Sending and receiving data, page 11-11

Terminator, page 11-11

Baud rate, page 11-12

Data bits and parity, page 11-12

Flow control (signal handshaking), page 11-12

RS-232 connections, page 11-13

Error messages, page 11-14

## Overview

The documentation in this section provides detailed information on using communications interfaces:

- “Selecting an interface”, page 11-2
- “GPIB operation”, page 11-3
- “General bus commands”, page 11-7
- “Front panel GPIB operation”, page 11-9
- “RS-232 interface operation”, page 11-10

**NOTE** See [Section 2](#) for more information on the GPIB and RS-232 communications interfaces. The TSP-Link is also a communications interface. See [Section 9](#) for details.

## Selecting an interface

The Model 260x SourceMeter supports two built-in remote interfaces:

- GPIB (General Purpose Interface Bus)
- RS-232 interface

You can manually select the GPIB or RS-232 interface, or have the unit automatically select the interface (the default) by using the COMMUNICATIONS menu accessed with the MENU key. The unit can only be remote to one interface at a time. In auto-select, the unit will remote to the first interface on which it receives a message. It will ignore the other interface until the unit is taken back to local operation.

When the unit powers up, it will display the communication parameters of the selected interface. If auto-selection is enabled, the unit will display the communication parameters of both interfaces.

# GPIB operation

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

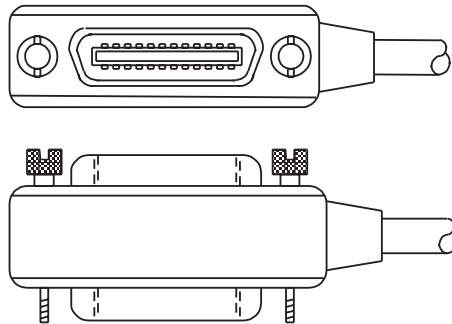
## GPIB standards

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

## GPIB connections

To connect the SourceMeter to the GPIB bus, use a cable equipped with standard IEEE-488 connectors as shown in [Figure 11-1](#).

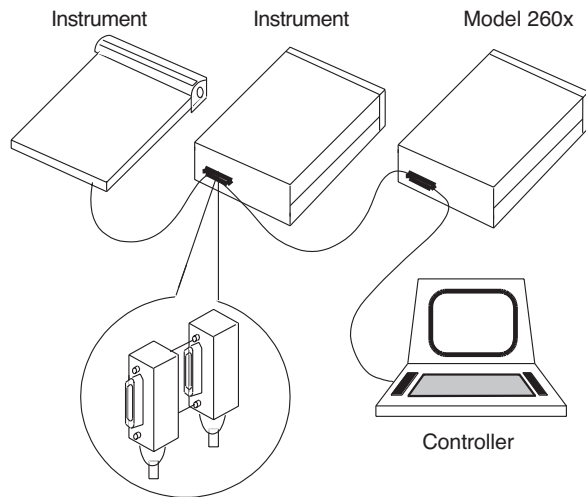
Figure 11-1  
**IEEE-488 connector**



To allow many parallel connections to one instrument, stack the connectors. Two screws are located on each connector to ensure that connections remain secure. [Figure 11-2](#) shows a typical connecting scheme for a multi-unit test system.

To avoid possible mechanical damage, stack no more than three connectors on any one unit. To minimize interference caused by electromagnetic radiation, use only shielded IEEE-488 cables. Available shielded cables from Keithley are listed in [“Options and accessories”](#) on page 1-5.

Figure 11-2  
**IEEE-488 connections**

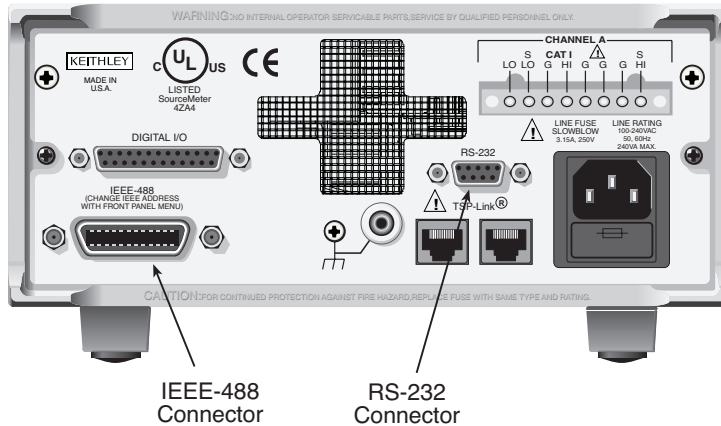


To connect the SourceMeter to the IEEE-488 bus, line up the cable connector with the connector located on the rear panel. Install and tighten the screws securely, making sure not to overtighten them. (Figure 11-3 shows the location of the connections.)

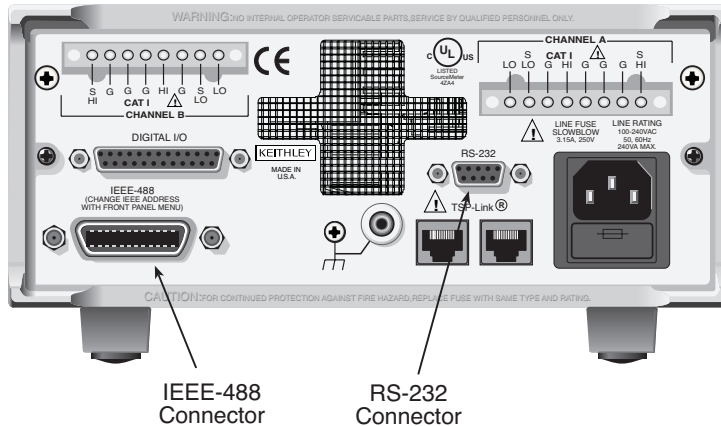
Connect any additional connectors from other instruments as required for your application. Make sure the other end of the cable is properly connected to the controller. You can only have 15 devices connected to an IEEE-488 bus, including the controller. The maximum cable length is either 20 meters or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

Figure 11-3  
IEEE-488 and RS-232 connector locations

Model 2601



Model 2602



## Primary address

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

## Front panel primary address

To set or check the primary address:

1. Press the MENU key.
2. Select COMMUNICATIONS, then press ENTER or the Rotary Knob.
3. Select INTERFACE\_CFG, then press ENTER or the Rotary Knob.
4. Select GPIB, then press ENTER or the Rotary Knob.
5. Set the primary address to the desired value, then press ENTER or the Rotary Knob.
6. Press EXIT to back out of the menu structure.

## Remote primary address

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

```
gpib.address = address
```

For example, the following command sets the address to 20:

```
gpib.address = 20
```

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

## Terminator

When receiving data over the GPIB, the Model 260x will terminate on any line feed character or any data byte with EOI asserted (line feed with EOI asserted is also valid). When sending data, it will append a line feed character to all outgoing messages. The EOI line will be asserted with the terminating line feed character.

## General bus commands

General commands are those commands, such as DCL, that have the same general meaning regardless of the instrument. [Table 11-1](#) lists the general bus commands.

Table 11-1  
**General bus commands**

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

### REN (remote enable)

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

### IFC (interface clear)

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

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



## LLO (local lockout)

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

## GTL (go to local)

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

## DCL (device clear)

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

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

## SDC (selective device clear)

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

## GET (group execute trigger)

GET is a GPIB trigger that is used to trigger the instrument to take readings by remote.

## SPE, SPD (serial polling)

Use the serial polling sequence to obtain the SourceMeter serial poll byte. The serial poll byte contains important information about internal functions. (See [Appendix D](#).) Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line. The serial polling sequence may be performed at any time to obtain the status byte from the SourceMeter.

# Front panel GPIB operation

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

## Error and status messages

See [Appendix B](#) for a list of status and error messages associated with IEEE-488 programming. The instrument can be programmed to generate an SRQ, and command queries can be performed to check for specific error conditions.

## GPIB status indicators

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

### REM

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

### TALK

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

### LSTN

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

### SRQ

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

## LOCAL key

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

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

# RS-232 interface operation

## Setting RS-232 interface parameters

### Front panel RS-232 parameters

To set interface parameters:

1. Press the MENU key.
2. Select COMMUNICATIONS, then press ENTER or the Rotary Knob.
3. Select INTERFACE\_CFG, then press ENTER or the Rotary Knob.
4. Select RS-232, then press ENTER or the Rotary Knob. Set these interface parameters as outlined in the following paragraphs:
  - Baud rate
  - Number of bits
  - Parity
  - Flow control
5. Press EXIT as needed to back out of the menu structure.

### Remote RS-232 parameters

Commands to set RS-232 parameters are listed in [Table 11-2](#). See [Section 12](#) for more information.

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

Table 11-2  
**RS-232 interface commands**

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

### RS-232 programming example

Send the following commands to set the baud rate to 9600 with no flow control:

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

## Sending and receiving data

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

## Terminator

When receiving data over the RS-232 interface, the Model 260x will terminate on any line feed character. When sending data, it will append a line feed character to all outgoing messages.

## Baud rate

The baud rate is the rate at which the SourceMeter and the programming terminal communicate. Choose one of the following available rates:

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

The factory selected baud rate is 9600.

When you choose a baud rate, make sure the programming computer that you are connecting to the SourceMeter can support the baud rate you selected. Both the SourceMeter and the other device must be configured for the same baud rate.

## Data bits and parity

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

## Flow control (signal handshaking)

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

The RS-232 interface provides two control lines (RTS and CTS) for this purpose (see [Figure 11-4](#) and [Table 11-3](#)). When the Model 260x is ready to send (RTS) data, it will transmit when it receives the clear to send (CTS) signal from the computer.

To enable or disable flow control, use the RS-232 configuration menu. Select **HARDWARE** to enable flow control, or **NONE** to disable it.

## RS-232 connections

The RS-232 serial port is connected to the serial port of a computer using a straight-through RS-232 cable terminated with DB-9 connectors. Do not use a null modem cable. The serial port uses the transmit (TXD), receive (RXD), CTS and RTS (if flow control is enabled), and signal ground (GND) lines of the RS-232 standard. [Figure 11-4](#) shows the rear panel connector for the RS-232 interface, and [Table 11-3](#) shows the pinout for the connector. The connector location is shown in [Figure 11-3 on page 11-5](#).

If your computer uses a DB-25 connector for the RS-232 interface, you will need a standard cable or adapter with a DB-25 connector on one end and a DB-9 connector on the other. An available RS-232 cable from Keithley is listed in [“Options and accessories” on page 1-5](#).

Figure 11-4  
**RS-232 interface connector**

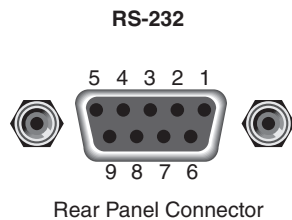


Table 11-3  
**RS-232 connector pinout**

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

[Table 11-4](#) provides pinout identification for the 9-pin (DB-9) or 25-pin (DB-25) serial port connector on the computer (PC).

Table 11-4  
**PC serial port pinout**

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

\* The Model 260x does not use all RS-232 signals. See [Table 11-3](#).

## Error messages

See [Appendix B](#) for RS-232 error messages.

# 12

## Instrument Control Library

---

### Section 12 topics

**Command programming notes**, page 12-2

Conventions, page 12-2

Functions and attributes, page 12-3

TSP-Link nodes, page 12-5

Logical instruments, page 12-5

Reading buffers, page 12-6

Time and date values, page 12-8

**ICL functions and attributes list**, page 12-9



# Command programming notes

## Conventions

For the following command reference, it is necessary to understand the following conventions:

### Wild characters

Many SMU commands are expressed in a generic form using wild characters. A wild character indicates an SMU channel, function or trigger line. Keep in mind that wild characters used in the generic form are NOT to be included in the command sent to the instrument.

#### **x** and **y**

The **x** character is used for functions and attributes to indicate the SMU channel (**a** or **b**) and **y** is used to indicate the SMU function (**v**, **i**, **r** or **p**). For example, the attribute for the source output setting is generically expressed as follows:

```
smuX.source.levelY
```

To program SMU channel A to 5 volts, the following command statement is to be sent to the instrument:

```
smua.source.levelv = 5.0
```

To program SMU channel B to 1 milliampere, the following command statement is to be sent to the instrument:

```
smub.source.leveli = 0.001
```

The wild characters **x** and/or **y** are NEVER sent to the instrument. They are used in this command reference for notational convenience only.

#### **[N]**

The **N** character, enclosed by brackets (**[ ]**), is used in functions and attributes for the Digital I/O line (**1** to **14**). For example, the function to assert an output trigger is generically expressed as follows:

```
digio.trigger[N].assert
```

To program the Model 260x to assert an output trigger on trigger line 5, the following command statement is sent to the instrument.

```
digio.trigger[5].assert()
```

The wild character **N** should NOT be sent to the instrument. However, the brackets (**[ ]**) must be included in the command. Also, note that the above command requires that a set of open and closed parenthesis (**()**) be appended to the function (see “[Functions](#)” on [page 12-3](#)).

## Functions and attributes

Commands can be function based or attribute based.

### Functions

Function based commands are used to control actions or activities. For example, performing a voltage measurement is a function (action) of an SMU. A function based command is not necessarily directly related to a Model 260x operation. For example, the `bit.bitand` function will logically AND two numbers.

Each function consists of a function name followed by a set of parenthesis (`()`). If the function does not have a parameter, the parenthesis set is left empty. Examples:

<code>digio.writeport(15)</code>	Sets digital I/O lines 1, 2, 3 and 4 high.
<code>digio.writebit(3, 0)</code>	Sets line 3 low (0).
<code>smua.reset()</code>	Returns SMU A to its default settings.
<code>digio.readport()</code>	Reads the digital I/O port.

The results of a function call are used by assigning the return values to variables and accessing those variables. The following code will measure SMU A voltage and return the reading:

```
reading = smua.measure.v()
print(reading)
```

Output: 2.360000e+00

The above output indicates that the voltage reading is 2.36V.

For a function that returns one value, the function call can be used in an expression. For example:

```
if smua.measure.v() > 5 then
    ...
end
```

### Attributes

An attribute is a characteristic of an instrument feature or operation. For example, some characteristics of an SMU source include the source function, range and output level.

### Assigning a value to an attribute

An attribute-based command can be used to assign a new value to an attribute. For many attributes, the value can be in the form of a discrete number or a pre-defined identifier. For example, filter type is an attribute. The moving average filter is selected by assigning the attribute to either of the following values:

0 OR `smuX.FILTER_MOVING_AVG`.

Either of the following command messages will configure SMU A for the moving average filter:

```
smua.measure.filter.type = 0
smua.measure.filter.type = smua.FILTER_MOVING_AVG
```

Some attributes can take any numeric value that is within a valid range. For example, the voltage source can be set from -40.4V to +40.4V. The following command message sets the SMU A source level to 1.53V:

```
smua.source.levelv = 1.53
```

### Reading an attribute

Reading an attribute is accomplished by passing it to a function call as a parameter or by assigning it to another variable.

**Parameter passing example** – The following command reads the filter type for SMU A by passing the attribute to the `print` function, which outputs a value:

```
print(smua.measure.filter.type)
```

Output: 0.000000e+00

The above output indicates that the moving average filter is selected.

**Variable assignment example** – The following command reads the filter type by assigning the attribute to a variable named `filtertype`:

```
filtertype = smua.measure.filter.type
```

### Syntax rules

- Commands for functions and attributes are case sensitive. As a general rule, all function and attribute names must be in lower case, while parameters use a combination of lower and upper case characters. Upper case characters are required for attribute constants. Example:

```
smua.source.func = smua.OUTPUT_DCVOLTS
```

In the above command to select the volts source function, `OUTPUT_DCVOLTS` is the attribute constant.

- Whitespace in a function is not required. The function to set digital I/O line 3 low can be sent with or without whitespaces as follows:

```
digio.writebit(3,0)           -- Whitespaces NOT used in string.
digio.writebit (3, 0)        -- Whitespaces used in string.
```

- Some commands require multiple parameters. Multiple parameters must be separated by commas (`,`), as shown above for the `digio.writebit` function.

## TSP-Link nodes

Each instrument or enclosure attached to the TSP-Link bus must be uniquely identified. This identification is called a TSP-Link node number and the enclosures are called nodes. Each node must be assigned a unique node number.

From a TSP point of view, nodes look like tables. There is one global table named `node` that contains all the actual nodes that are themselves tables. An individual node is accessed as `node[N]` where `N` is the node number assigned to the node. Each node has certain attributes that can be accessed as elements of its associated table. These are listed as follows:

<code>id</code>	The node number assigned to the node.
<code>model</code>	The product model number string of the node.
<code>revision</code>	The product revision string of the node.
<code>serialno</code>	The product serial number string of the node.

There is also an entry for each logical instrument on the node (see [“Logical instruments”](#)).

It is not necessary to know the node number of the node running a script. The variable `localnode` is an alias for the node entry the script is running on. For example, if a script is running on node 5, the global variable `localnode` will be an alias for `node[5]`.

## Logical instruments

You would normally refer to all instrumentation within one enclosure or node as a single instrument. From an TSP/ICL point of view, it is useful to think of individual SMUs as instruments. To avoid confusion, SMUs and other subdivisions of the instrumentation within an enclosure will be referred to as “logical instruments.”

Each logical instrument is given a unique identifier in a system. These identifiers are used as part of all ICL function calls that control a given logical instrument. A Model 260x SMU has the following logical instruments per enclosure:

<code>beeper</code>	<code>errorqueue</code>	<code>smua</code>	<code>status</code>
<code>digio</code>	<code>gpib</code>	<code>smub</code>	<code>tsplink</code>
<code>display</code>	<code>serial</code>	<code>timer</code>	

Logical instruments also look like TSP tables. In addition to the logical-instrument-specific attributes and the commands to which they respond, there are a few attributes that provide information about the logical instrument. These attributes are listed below:

<code>name</code>	A string that represents the logical instrument’s name. For example, <code>smua</code> .
-------------------	--

`node` A reference to the TSP-Link node of which the logical instrument is a part.

Each logical instrument has an element for each command that it supports. These commands are documented in this section. Note that `smua` and `smub` support the same command set and are documented jointly as `smuX`.

On any given node, the logical instrument identifiers from that node are also global variables. They can be accessed as elements of the node they belong or directly if running on that node. For example, to execute the `measure.v` command on `smua` on node `[5]`, one could use `node[5].smua.measure.v()`. If the command is being issued (executed) on `node[5]`, then `smua.measure.v()` is sufficient. Only be concerned with node numbers when controlling multiple units via the TSP-Link.

## Reading buffers

Readings can be obtained in multiple ways. Reading acquisition can be synchronous or overlapped. Furthermore, the routines that make single point measurements can be configured to make multiple measurements where only one would ordinarily be made. Also, consider that the measured value is not the only component of a reading. The measurement status (e.g. “In Compliance” or “Over ranged”) is also data associated with a particular reading.

All routines that return measurements can return them as reading buffers. Overlapped measurements are always returned in a reading buffer. Synchronous measurements return a single value or both a single value and a reading buffer. The more advanced user can use the reading buffer to access the additional information stored in the reading buffer.

A reading buffer is based on a TSL table. The measurements themselves are accessed by ordinary array access. If `rb` is a reading buffer, the first measurement is accessed as `rb[1]` and the 9<sup>th</sup> measurement as `rb[9]`, etc. The additional information in the table is accessed as additional members of the table. The following values are all available per reading buffer, i.e., `rb.appendmode`:

<code>appendmode</code>	Off (default) or on. If off, a new measurement to this buffer will overwrite the previous contents. If on, the first new measurement will be stored at what was formerly <code>rb[n+1]</code> .
<code>basetimestamp</code>	The timestamp of when the reading at <code>rb[1]</code> was stored, in seconds, from time of power-up.
<code>capacity</code>	The total number of readings that can be stored in the reading buffer.
<code>collectsourcevalues</code>	When on, source values will be stored with readings in the buffer. This requires four extra bytes of storage per reading. This value, off (default) or on, can only be changed when the buffer is empty.

<code>collecttimestamps</code>	When on, timestamps will be stored with readings in the buffer. This requires four extra bytes of storage per reading. This value, off (default) or on, can only be changed when the buffer is empty.
<code>n</code>	The number of readings in the reading buffer.
<code>timestampresolution</code>	The timestamp resolution, in seconds. The default resolution is 0.000001 seconds. At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. This value can be increased for very long tests.

The following values are available per reading, i.e., `rb.measurefunctions[3]`, as enabled. Each is actually a nested table. Related entries are stored at the same index as the relevant measurement.

<code>measurefunctions</code>	An array (TSL table) of strings indicating the function measured for the reading (Current, Voltage, Ohms or Watts).
<code>measureranges</code>	An array (TSL table) of full scale range values for the measure range used when the measurement was made.
<code>readings</code>	An array (TSL table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly, i.e., <code>rb[2]</code> and <code>rb.readings[2]</code> are the same value.
<code>sourcefunctions</code>	An array (TSL table) of strings indicating the source function at the time of the measurement (Current or Voltage).
<code>sourceoutputstates</code>	An array (TSL table) of strings indicating the state of the source (Off or On).
<code>sourceranges</code>	An array (TSL table) of full scale range values for the source range used when the measurement was made.
<code>sourcevalues</code>	If enabled (see <code>collectsourcevalues</code> above), an array (TSL table) of the sourced value in effect at the time of the reading.
<code>statuses</code>	An array (TSL table) of status values for all of the readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value.
<code>timestamps</code>	An array (TSL table) of timestamps, in seconds, of when each reading occurred. These can be compared to the base-timestamp for the buffer for time lapsed.

For example, the number of readings the reading buffer can store is accessed as `rb.capacity`.

## Time and date values

Date/time values are represented as a number of seconds since some base. There are three time bases:

1. UTC 12:00 am Jan 1 1970.
2. When the Model 260x is powered on.
3. Time referenced to an event, such as the first reading stored in a reading buffer.

Time can be represented as the number of seconds since UTC 12:00 am on January 1, 1970. The commands that utilize this format are commands that require an absolute time reference such as setting the calibration date.

Time can also be represented as the number of seconds since the unit was powered on. The `os.clock()` function returns values in this format.

Representing time as a number of seconds will be referred to as “standard time format.” Note that because numbers are floating point numbers, the precision of date/time values will decrease the farther the elapsed time gets from its reference base. This decrease in precision is approximately 0.06ppm of the total elapsed time. This precision is generally more accurate than the time base of the instrument and should not present any problems. It is worth noting, however, because you can directly see the affects as compared to the less obvious time-base drift.

# ICL functions and attributes list

beeper function and attribute, page 12-11

beeper.beep                      beeper.enable

bit functions, page 12-12

bit.bitand	bit.clear	bit.set	bit.toggle
bit.bitor	bit.get	bit.setfield	
bit.bitxor	bit.getfield	bit.test	

delay function, page 12-18

delay

digio functions and attributes, page 12-19

digio.readbit	digio.trigger[N].clear	digio.trigger[N].release	digio.writeport
digio.readport	digio.trigger[N].mode	digio.trigger[N].wait	digio.writeprotect
digio.trigger[N].assert	digio.trigger[N].pulsewidth	digio.writebit	

display functions and attributes, page 12-24

display.clear	display.inputvalue	display.prompt	display.smuX.digits
display.getannunciators	display.loadmenu.add	display.screen	display.smuX.measure.func
display.getcursor	display.loadmenu.delete	display.sendkey	display.trigger.clear
display.getlastkey	display.locallockout	display.locallockout	display.trigger.wait
display.gettext	display.menu	display.settext	display.waitkey

errorqueue functions and attribute, page 12-40

errorqueue.clear                      errorqueue.count                      errorqueue.next

exit function, page 12-42

exit

format attributes, page 12-42

format.asciiprecision                      format.byteorder                      format.data

gpib attribute, pg. 12-45

gpib.address

localnode attributes, page 12-46

localnode.linefreq                      localnode.prompts                      localnode.showerrors

makegetter functions, page 12-48

makegetter                      makesetter

opc function, page 12-50

opc



printbuffer and printnumber functions, page 12-51

printbuffer                      printnumber

reset function, page 12-53

reset

serial functions and attributes, page 12-53

serial.baud	serial.flowcontrol	serial.read
serial.databits	serial.parity	serial.write

setup functions and attribute, page 12-56

setup.poweron	setup.recall	setup.save
---------------	--------------	------------

smuX functions and attributes, page 12-57

smuX.cal.date	smuX.measure.filter.type	smuX.nvbufferY.collecttimestamps
smuX.cal.due	smuX.measure.interval	smuX.nvbufferY.n
smuX.cal.lock	smuX.measure.lowrangeY	smuX.nvbufferY.timestampresolution
smuX.cal.password	smuX.measure.nplc	smuX.reset
smuX.cal.polarity	smuX.measure.overlappedY	smuX.sense
smuX.cal.restore	smuX.measure.rangeY	smuX.source.autorangeY
smuX.cal.save	smuX.measure.rel.enableY	smuX.source.calibrateY
smuX.cal.state	smuX.measure.rel.levelY	smuX.source.compliance
smuX.makebuffer	smuX.measure.Y	smuX.source.func
smuX.makebuffer	smuX.measureYandstep	smuX.source.levelY
smuX.measure.autorangeY	smuX.nvbufferY	smuX.source.limitY
smuX.measure.autozero	smuX.nvbufferY.appendmode	smuX.source.lowrangeY
smuX.measure.calibrateY	smuX.nvbufferY.basetimestamp	smuX.source.offmode
smuX.measure.count	smuX.nvbufferY.capacity	smuX.source.output
smuX.measure.filter.count	smuX.nvbufferY.clear	smuX.source.outputenableaction
smuX.measure.filter.enable	smuX.nvbufferY.collectsourcevalues	smuX.source.rangeY

**status** functions and attributes are documented in [Appendix D](#).

timer functions, page 12-86

timer.measure.t	timer.reset
-----------------	-------------

trigger functions, page 12-87

trigger.clear	trigger.wait
---------------	--------------

tsplink function and attributes, page 12-88

tsplink.node	tsplink.reset	tsplink.state
--------------	---------------	---------------

userstring functions, page 12-89

userstring.add	userstring.catalog	userstring.delete	userstring.get
----------------	--------------------	-------------------	----------------

waitcomplete function, page 12-91

waitcomplete

## beeper function and attribute

The beeper generates a beep tone. It is typically used to announce the start and/or completion of a test or operation.

<b>beeper.beep</b>	
<b>Function</b>	Generates a beep tone.
<b>Usage</b>	<pre>beeper.beep(duration, frequency)</pre> <p>duration      Set from 0.1 to 100 (seconds). frequency     Set to 453, 621, 987 or 2400 (Hz).</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• There are four beeper frequencies: 453Hz, 621Hz, 987Hz and 2400Hz. If you set <code>frequency</code> to a different value, the closest supported frequency will be selected.</li> <li>• The beeper will not sound if it is disabled (see <a href="#">beeper.enable</a> attribute).</li> <li>• This function is an overlapped command. Script execution will continue and not wait for the beep to finish. If another beep command is issued before the previous beep finishes, the first beep will be terminated. The <a href="#">waitcomplete</a> function can be used to hold up script execution until the beep command finishes.</li> </ul>
<b>Also see</b>	<a href="#">beeper.enable</a>
<b>Example</b>	Enables the beeper and generates a two-second, 2400Hz beep: <pre>beeper.enable = 1 beeper.beep(2, 2400)</pre>

<b>beeper.enable</b>	
<b>Attribute</b>	Beeper control (on/off).
<b>Usage</b>	<pre>beeperstate = beeper.enable      -- Reads beeper state. beeper.enable = beeperstate      -- Writes beeper state.</pre> <p>Set <code>beeperstate</code> to one of the following values:</p> <pre>0      Beeper disabled 1      Beeper enabled</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute enables or disables the beeper. Disabling the beeper also disables front panel key clicks.</li> <li>• Cycling power enables the beeper. The <code>reset</code> function does not affect the beeper state.</li> </ul>
<b>Also see</b>	<a href="#">beeper.beep</a>
<b>Example</b>	Enables the beeper and generates a two second, 2400Hz beep: <pre>beeper.enable = 1 beeper.beep(2, 2400)</pre>

## bit functions

### Logic and bit operations

The bit functions are used to perform bitwise logic operations on two given numbers, and bit operations on one given number. Logic and bit operations truncate the fractional part of given numbers to make them integers.

**NOTE** The TSP stores all numbers internally as single precision IEEE-754 floating point values. The internal number representation only stores 24 bits of numeric data. The logic operations will work correctly for all integer values between 0 and 4294967295. However, only the 24 most significant bits will be stored for the return value.

**Logic operations** – The `bit.bitand`, `bit.bitor` and `bit.bitxor` functions in this group perform logic operations on two numbers. The TSP will perform the indicated logic operation on the binary equivalents of the two integers. Logic operations are performed bitwise. That is, bit 1 of the first number is AND’ed, OR’ed or XOR’ed with bit 1 of the second number. Bit 2 of the first number is AND’ed, OR’ed or XOR’ed with bit 2 of the second number. This bitwise logic operation is performed on all corresponding bits of the two numbers. The result of a logic operation will be returned as an integer.

**Bit operations** – The rest of the functions in this group are used for operations on the bits of a given number. These functions can be used to clear a bit, toggle a bit, test a bit, set a bit (or bit field) and retrieve the weighted value of a bit (or field value). All of these functions use an `index` parameter to “point” to the bit position of the given number. The least significant bit of a given number has an index of 1, and the most significant bit has an index of 32.

<b>bit.bitand</b>	
<b>Function</b>	Performs a bitwise logical AND operation on two numbers.
<b>Usage</b>	<pre>value = bit.bitand(value1, value2)</pre> <p> <code>value1</code>      First number for the AND operation.  <code>value2</code>      Second number for the AND operation.  <code>value</code>        Returned result of the AND operation. </p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function performs a logical AND operation on two numbers.</li> <li>• Any fractional parts of <code>value1</code> and <code>value2</code> are truncated to make them integers. The returned <code>value</code> is also an integer.</li> <li>• See “Logic and bit operations” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.bitor</a> , <a href="#">bit.bitxor</a>

<b>Example</b>	<p>AND'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a value of decimal 8 (binary 1000):</p> <pre>value = bit.bitand(10, 9) print(value)</pre> <p>Output: 8.000000e+00</p>
----------------	--

<b>bit.bitor</b>	
<b>Function</b>	Performs a bitwise logical OR operation on two numbers.
<b>Usage</b>	<pre>value = bit.bitor(value1, value2)</pre> <p>value1            First number for the OR operation.  value2            Second number for the OR operation.  value              Returned result of the OR operation.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function performs a logical OR operation on two numbers.</li> <li>• Any fractional parts of <code>value1</code> and <code>value2</code> are truncated to make them integers. The returned <code>value</code> is also an integer.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.bitand</a> , <a href="#">bit.bitxor</a>
<b>Example</b>	<p>OR'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a value of decimal 11 (binary 1011):</p> <pre>value = bit.bitor(10, 9) print(value)</pre> <p>Output: 1.100000e+01</p>

<b>bit.bitxor</b>	
<b>Function</b>	Performs a bitwise logical XOR (Exclusive OR) operation on two numbers.
<b>Usage</b>	<pre>value = bit.xor(value1, value2)</pre> <p>value1            First number for the XOR operation.  value2            Second number for the XOR operation.  value              Returned result of the XOR operation.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function performs a logical Exclusive OR operation on two numbers.</li> <li>• Any fractional parts of <code>value1</code> and <code>value2</code> are truncated to make them integers. The returned <code>value</code> is also an integer.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.bitand</a> , <a href="#">bit.bitor</a>

<b>Example</b>	<p>XOR'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a value of decimal 3 (binary 0011):</p> <pre>value = bit.bitxor(10, 9) print(value)</pre> <p>Output: 3.000000e+00</p>
----------------	--

<b>bit.clear</b>	
<b>Function</b>	Clears a bit at a given index position.
<b>Usage</b>	<pre>value = bit.clear(value1, index)</pre> <p>value1      Given number. index        Index position of the bit to be cleared (1 to 32). value        Returns the result of the manipulation.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function clears a bit at a given index position.</li> <li>• Any fractional part of <code>value1</code> is truncated to make it an integer. The returned <code>value</code> is also an integer.</li> <li>• The least significant bit of the given number is at index 1. The most significant bit is at index 32.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.get</a> , <a href="#">bit.getfield</a> , <a href="#">bit.set</a> , <a href="#">bit.setfield</a> , <a href="#">bit.test</a> , <a href="#">bit.toggle</a>
<b>Example</b>	<p>The binary equivalent of decimal 15 is 1111. If you clear the bit at index position 2, the returned decimal <code>value</code> would be 13 (binary 1101):</p> <pre>value = bit.clear(15, 2) print(value)</pre> <p>Output: 1.300000e+01</p>

<b>bit.get</b>	
<b>Function</b>	Retrieves the weighted value of a bit at a given index position.
<b>Usage</b>	<pre>value = bit.get(value1, index)</pre> <p>value1      Given number. index        Index position of the bit to be retrieved (1 to 32). value        Returned weighted value of the bit.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function returns the value of the bit in <code>value1</code> at the given index. This is the same as returning <code>value1</code> with all other non-indexed bits set to zero.</li> <li>• Prior to retrieving the indexed bit, any fractional part of the given number will be truncated to make it an integer. The least significant bit of the given number has an index of 1 and the most significant bit has an index of 32.</li> <li>• If the indexed bit for the number is set to 0, the result will be 0.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>

<b>Also see</b>	<a href="#">bit.clear</a> , <a href="#">bit.getfield</a> , <a href="#">bit.set</a> , <a href="#">bit.setfield</a> , <a href="#">bit.test</a> , <a href="#">bit.toggle</a>
<b>Example</b>	The binary equivalent of decimal 10 is 1010. Getting the bit at index position 4 will return decimal value 8: <pre>value = bit.get(10, 4) print(value)</pre> Output: 8.000000e+00

<b>bit.getfield</b>	
<b>Function</b>	Returns a field of bits starting at a given index position.
<b>Usage</b>	<pre>value = bit.getfield(value1, index, width)</pre> <p>value1            Given number.  index             Index position of the first bit; 1 to (33 – width).  width             Field width – number of bits to be included in the field; 1 to 24.  value             Returned value of the bit field.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A field of bits is a contiguous group of bits. This function retrieves a field of bits from value1 starting at the given index position. The index position is the least significant bit of the retrieved field. The number of bits to return is given by width.</li> <li>• Prior to retrieving the field of bits, any fractional part of the given number will be truncated to make it an integer.</li> <li>• The least significant bit of the given number has an index of 1 and the most significant bit has an index of 32.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.clear</a> , <a href="#">bit.get</a> , <a href="#">bit.set</a> , <a href="#">bit.setfield</a> , <a href="#">bit.test</a> , <a href="#">bit.toggle</a>
<b>Example</b>	The binary equivalent of decimal 13 is 1101. The field at index 2 and width 3 consists of the binary bits 110. The returned value will be decimal 6 (binary 110): <pre>value = bit.getfield(13, 2, 3) print(value)</pre> Output: 6.000000e+00

<b>bit.set</b>	
<b>Function</b>	Sets a bit at a given index position.
<b>Usage</b>	<pre>value = bit.set(value1, index)</pre> <p>value1            Given number.  index             Index position of the bit to be set (1 to 32).  value             Returned value of the new number.</p>

<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function returns <code>value</code>, which is <code>value1</code> with the indexed bit set. The <code>index</code> must be a value between 1 and 32. The least significant bit of the given number has an index of 1 and the most significant bit has an index of 32.</li> <li>• Any fractional part of <code>value1</code> will be truncated to make it an integer.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.clear</a> , <a href="#">bit.get</a> , <a href="#">bit.getfield</a> , <a href="#">bit.setfield</a> , <a href="#">bit.test</a> , <a href="#">bit.toggle</a>
<b>Example</b>	<p>The binary equivalent of decimal 8 is 1000. If the bit at <code>index 3</code> is set to 1, the returned <code>value</code> will be decimal 12 (binary 1100):</p> <pre>value = bit.set(8, 3) print(value) Output: 1.200000e+01</pre>

<b>bit.setfield</b>	
<b>Function</b>	Overwrites a bit field at a given index position.
<b>Usage</b>	<pre>value = bit.setfield(value1, index, width, fieldvalue)</pre> <p><code>value1</code>            Given number.  <code>index</code>             Index position of the least significant bit of the field; 1 to (33 – <code>width</code>).  <code>width</code>             Field width – number of bits in the field; 1 to 24.  <code>fieldvalue</code>       Value to write to the field.  <code>value</code>             Returned value of the new number.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function returns <code>value</code>, which is <code>value1</code> with a field of bits overwritten, starting at the given <code>index</code> position. The <code>index</code> specifies the position of the least significant bit of the given field. The <code>width</code> bits starting at the given <code>index</code> will be set to the value given by <code>fieldvalue</code>. The least significant bit in <code>value1</code> has an index of 1 and the most significant bit has an index of 32.</li> <li>• Prior to setting the field of bits, any fractional parts of <code>value1</code> and <code>fieldvalue</code> will be truncated to make them integers.</li> <li>• If the <code>fieldvalue</code> is wider than the <code>width</code>, the extra most significant bits of the <code>fieldvalue</code> will be truncated. For example, assume the <code>width</code> is 4 bits, and the binary value for <code>fieldwidth</code> is 11110 (5 bits). The most significant bit of <code>fieldwidth</code> will be truncated, and a binary value of 1110 will be used as the <code>fieldvalue</code>.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.clear</a> , <a href="#">bit.get</a> , <a href="#">bit.getfield</a> , <a href="#">bit.set</a> , <a href="#">bit.test</a> , <a href="#">bit.toggle</a>
<b>Example</b>	<p>The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at index position 2, the returned <code>value</code> will be decimal 11 (binary 1011):</p> <pre>value = bit.setfield(15, 2, 3, 5) print(value) Output: 1.100000e+01</pre>

<b>bit.test</b>	
<b>Function</b>	Returns the Boolean value (true or false) of a bit at a given index position.
<b>Usage</b>	<pre>value = bit.test(value1, index)</pre> <p>value1      Given number. index        Index position of the bit to be tested (1 to 32). value        Returned decimal value of the bit.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function returns <code>value</code>, which is the result of the tested bit. The least significant bit of the given number is at index 1. The most significant bit is at index 32.</li> <li>• Any fractional part of <code>value1</code> will be truncated to make it an integer. If the indexed bit for <code>value1</code> is set to 0, the returned value will be false. If the indexed bit for <code>value1</code> is set to 1, the returned value will be true.</li> <li>• If the index is bigger than the number of bits in <code>value1</code>, the result will be false.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.clear</a> , <a href="#">bit.get</a> , <a href="#">bit.getfield</a> , <a href="#">bit.set</a> , <a href="#">bit.setfield</a> , <a href="#">bit.toggle</a>
<b>Example</b>	<p>The binary equivalent of decimal 10 is 1010. Testing the bit at index position 4 will return a Boolean value of true:</p> <pre>value = bit.test(10, 4) print(value)</pre> <p>Output: true</p>

<b>bit.toggle</b>	
<b>Function</b>	Toggles the value of a bit at a given index position.
<b>Usage</b>	<pre>value = bit.toggle(value1, index)</pre> <p>value1      Given number. index        Index position of the bit to be toggled (1 to 32). value        Returned value of the new number.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function returns <code>value</code>, which is the result of toggling a bit in <code>value1</code>.</li> <li>• Any fractional part of <code>value1</code> is truncated to make it an integer. The returned decimal value is also an integer. The least significant bit of the given number is index 1. The most significant bit is index 32.</li> <li>• The indexed bit for <code>value1</code> is toggled from 0 to 1, or 1 to 0.</li> <li>• See “<a href="#">Logic and bit operations</a>” on <a href="#">page 12-12</a> for more information.</li> </ul>
<b>Also see</b>	<a href="#">bit.clear</a> , <a href="#">bit.get</a> , <a href="#">bit.getfield</a> , <a href="#">bit.set</a> , <a href="#">bit.setfield</a> , <a href="#">bit.test</a>
<b>Example</b>	<p>The binary equivalent of decimal 10 is 1010. Toggling the bit at index position 3 will return a decimal value of 14 (binary 1110).</p> <pre>value = bit.toggle(10, 3) print(value)</pre> <p>Output: 1.400000e+01</p>



## delay function

This function is used to hold up system operation for a specified period of time. It is typically used to soak a device at a specific voltage or current for a period of time.

<b>delay</b>	
<b>Function</b>	Delays system operation.
<b>Usage</b>	<code>delay(seconds)</code> seconds      Set delay in seconds (100000 seconds maximum).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will delay for the specified number of seconds. It is impossible to delay for zero seconds.</li> <li>• Delays smaller than 50<math>\mu</math>s will be dominated by overhead such that the actual delay might be as long as 50<math>\mu</math>s (typical). For delays longer than 50<math>\mu</math>s, the delay may be as much as 10<math>\mu</math>s (typical) more than the requested delay.</li> </ul>
<b>Example</b>	Sets the SMU A output to 1V, soaks the DUT for 50ms and then turns the output off: <pre>smua.source.levelv = 1.0 delay(0.050) smua.source.off()</pre>

## digio functions and attributes

The functions and attributes in this group are used to control read/write and trigger operations for the digital I/O Port.

**NOTE** The digital I/O lines can be used for both input and output. If a line is being driven low, then a “0” value will be read by a command for that line. You must write a “1” to all digital I/O lines that are to be used as inputs.

<b>digio.readbit</b>	
<b>Function</b>	Reads one digital I/O line.
<b>Usage</b>	<code>data = digio.readbit(line)</code> <code>line</code> Digital I/O line number to be read (1 to 14).
<b>Remarks</b>	A returned value of “0” indicates that the line is low. A returned value of “1” indicates that the line is high.
<b>Details</b>	See “Digital I/O port” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">digio.readport</a> , <a href="#">digio.writebit</a> , <a href="#">digio.writeport</a>
<b>Example</b>	Assume line 4 is set high, and it is then read: <code>data = digio.readbit(4)</code> <code>print(data)</code> Output: 1.000000e+00

<b>digio.readport</b>	
<b>Function</b>	Reads the digital I/O port.
<b>Usage</b>	<code>data = digio.readport()</code>
<b>Remarks</b>	The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and bit 14 corresponds to line 14. For example, a returned value of 170 has a binary equivalent of 0000010101010. Lines 2, 4, 6 and 8 are high (1), and the other 10 lines are low (0).
<b>Details</b>	See “Digital I/O port” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">digio.readbit</a> , <a href="#">digio.writebit</a> , <a href="#">digio.writeport</a>
<b>Example</b>	Assume lines 2, 4, 6 and 8 are set high, and the I/O port is then read: <code>data = digio.readport()</code> <code>print(data)</code> Output: 1.700000e+02 (binary 10101010)

<b>digio.trigger[N].assert</b>		N is a digital I/O trigger line: 1 to 14
<b>Function</b>	Asserts a trigger on one of the digital I/O lines.	
<b>Usage</b>	<code>digio.trigger[N].assert()</code>	
<b>Remarks</b>	The set pulsewidth determines how long the trigger is asserted.	
<b>Details</b>	See “ <a href="#">Controlling digital I/O lines</a> ” in <a href="#">Section 10</a> .	
<b>Also see</b>	<a href="#">digio.trigger[N].pulsewidth</a>	
<b>Example</b>	Asserts trigger on I/O line 2: <code>digio.trigger[2].assert()</code>	

<b>digio.trigger[N].clear</b>		N is a digital I/O trigger line: 1 to 14
<b>Function</b>	Clears a trigger event on a digital I/O line.	
<b>Usage</b>	<code>digio.trigger[N].clear()</code>	
<b>Remarks</b>	A trigger’s event detector remembers if a trigger event has been detected since the last <code>digio.trigger[line].wait</code> call. This function clears a trigger’s event detector and discards the previous history of the trigger line.	
<b>Details</b>	See “ <a href="#">Controlling digital I/O lines</a> ” in <a href="#">Section 10</a> .	
<b>Also see</b>	<a href="#">digio.trigger[N].wait</a>	
<b>Example</b>	Clears trigger event on I/O line 2: <code>digio.trigger[2].clear()</code>	

<b>digio.trigger[N].mode</b>		N is a digital I/O trigger line: 1 to 14								
<b>Attribute</b>	Trigger operation and detection mode.									
<b>Usage</b>	<pre> tmode = digio.trigger[N].mode    -- Reads trigger mode. digio.trigger[N].mode = tmode    -- Writes trigger mode.           </pre> <p>Set <code>tmode</code> to one of the following values:</p> <table border="0"> <tr> <td>1 or <code>digio.TRIG_FALLING</code></td> <td>Input: Detects falling edge triggers. Output: Asserts TTL-low trigger pulse.</td> </tr> <tr> <td>2 or <code>digio.TRIG_RISING</code></td> <td>Input: Detects rising edge triggers. Output: Asserts TTL-high trigger pulse.</td> </tr> <tr> <td>3 or <code>digio.TRIG_EITHER</code></td> <td>Input: Detects rising or falling edge triggers. Output: Asserts a TTL-low trigger pulse.</td> </tr> <tr> <td>5 or <code>digio.TRIG_SYNCHRONOUS</code></td> <td>Input: Detects falling edge-triggers, and then latch and drive them low. Output: Asserts a TTL-low trigger pulse.</td> </tr> </table>		1 or <code>digio.TRIG_FALLING</code>	Input: Detects falling edge triggers. Output: Asserts TTL-low trigger pulse.	2 or <code>digio.TRIG_RISING</code>	Input: Detects rising edge triggers. Output: Asserts TTL-high trigger pulse.	3 or <code>digio.TRIG_EITHER</code>	Input: Detects rising or falling edge triggers. Output: Asserts a TTL-low trigger pulse.	5 or <code>digio.TRIG_SYNCHRONOUS</code>	Input: Detects falling edge-triggers, and then latch and drive them low. Output: Asserts a TTL-low trigger pulse.
1 or <code>digio.TRIG_FALLING</code>	Input: Detects falling edge triggers. Output: Asserts TTL-low trigger pulse.									
2 or <code>digio.TRIG_RISING</code>	Input: Detects rising edge triggers. Output: Asserts TTL-high trigger pulse.									
3 or <code>digio.TRIG_EITHER</code>	Input: Detects rising or falling edge triggers. Output: Asserts a TTL-low trigger pulse.									
5 or <code>digio.TRIG_SYNCHRONOUS</code>	Input: Detects falling edge-triggers, and then latch and drive them low. Output: Asserts a TTL-low trigger pulse.									

<b>Remarks</b>	<ul style="list-style-type: none"> <li>• <code>tmode</code> can be expressed as a number (1, 2, 3 or 5) or as one of the pre-defined constants (see “<b>Usage</b>”).</li> <li>• When reading the trigger mode, it is returned as a number.</li> </ul>
<b>Details</b>	See “ <a href="#">Controlling digital I/O lines</a> ” in <a href="#">Section 10</a> .
<b>Example</b>	Sets the trigger mode for I/O line 4 to TRIG_RISING: <code>digio.trigger[4].mode = 2</code>

<b>digio.trigger[N].pulsewidth</b>		N is a digital I/O trigger line: 1 to 14
<b>Attribute</b>	The length of time that the trigger line will be asserted for output triggers.	
<b>Usage</b>	<code>width = digio.trigger[N].pulsewidth -- Reads pulsewidth.</code> <code>digio.trigger[N].pulsewidth = width -- Writes pulsewidth.</code> width           Pulsewidth length (seconds).	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The trigger line is guaranteed to be asserted for at least the specified time, and it might be asserted slightly longer.</li> <li>• Setting pulsewidth to 0 (seconds) asserts the trigger indefinitely.</li> <li>• The default pulsewidth time is 10<math>\mu</math>s.</li> </ul>	
<b>Details</b>	See “ <a href="#">Controlling digital I/O lines</a> ” in <a href="#">Section 10</a> .	
<b>Also see</b>	<a href="#">digio.trigger[N].release</a>	
<b>Example</b>	Sets pulsewidth for trigger line 4 to 20 $\mu$ s: <code>digio.trigger[4].pulsewidth = 20e-6</code>	

<b>digio.trigger[N].release</b>		N is a digital I/O trigger line: 1 to 14
<b>Function</b>	Releases an indefinite length or latched trigger.	
<b>Usage</b>	<code>digio.trigger[N].release()</code>	
<b>Remarks</b>	Releases a trigger that was asserted with an indefinite pulsewidth, as well as a trigger that was asserted in response to receiving a synchronous mode trigger. Only the specified trigger line (N) is affected.	
<b>Details</b>	See “ <a href="#">Controlling digital I/O lines</a> ” in <a href="#">Section 10</a> .	
<b>Also see</b>	<a href="#">digio.trigger[N].pulsewidth</a>	
<b>Example</b>	Releases trigger line 4: <code>digio.trigger[4].release()</code>	

<b>digio.trigger[N].wait</b>	
N is a digital I/O trigger line: 1 to 8	
<b>Function</b>	Waits for a trigger.
<b>Usage</b>	<pre>triggered = digio.trigger[N].wait(timeout) timeout      Set timeout in seconds. triggered    Returns 'true' if a trigger was detected, or 'false' if no triggers were               detected during the timeout period.</pre>
<b>Remarks</b>	This function will wait up to <code>timeout</code> seconds for an input trigger. If one or more trigger events were detected since the last time <code>digio.trigger[N].wait</code> or <code>digio.trigger[N].clear</code> was called, this function will return immediately. After waiting for a trigger with this function, the event detector will be automatically reset and re-armed. This is true regardless of the number of events detected.
<b>Details</b>	See “ <a href="#">Controlling digital I/O lines</a> ” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">digio.trigger[N].clear</a>
<b>Example</b>	<p>Waits up to three seconds for a trigger to be detected on trigger line 4, then displays if the trigger was detected:</p> <pre>triggered = digio.trigger[4].wait(3) print(triggered)</pre> <p>Output: <code>false</code> (no triggers detected)  <code>true</code> (trigger detected)</p>

<b>digio.writebit</b>	
<b>Function</b>	Sets a digital I/O line high or low.
<b>Usage</b>	<pre>digio.writebit(bit, data) bit           Digital I/O line number (1 to 14) data         Value to write to the bit; 0 (low) or 1 (high)</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If the output line is write protected, via the <a href="#">digio.writeprotect</a> attribute, the command will be ignored.</li> <li>• The <code>reset</code> function does not affect the present states of the digital I/O lines.</li> </ul>
<b>Details</b>	See “ <a href="#">Digital I/O port</a> ” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">digio.readbit</a> , <a href="#">digio.readport</a> , <a href="#">digio.writeport</a>
<b>Example</b>	<p>Sets digital I/O line 4 low (0):</p> <pre>digio.writebit(4, 0)</pre>

<b>digio.writeport</b>	
<b>Function</b>	Writes to all digital I/O lines.
<b>Usage</b>	<code>digio.writeport (data)</code> <code>data</code> Value to write to the port; 0 to 16383.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The binary representation of <code>data</code> indicates the output pattern to be written to the I/O port. For example, a <code>data</code> value of 170 has a binary equivalent of 0000010101010. Lines 2, 4, 6 and 8 are set high (1), and the other 10 lines are set low (0).</li> <li>• Write protected lines will not be changed (see <a href="#">digio.writeprotect</a>).</li> <li>• The <code>reset</code> function does not affect the present states of the digital I/O lines.</li> </ul>
<b>Details</b>	See “Digital I/O port” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">digio.readbit</a> , <a href="#">digio.readport</a> , <a href="#">digio.writebit</a>
<b>Example</b>	Sets digital I/O lines 1 through 8 high (binary 00000011111111): <code>digio.writeport (255)</code>

<b>digio.writeprotect</b>	
<b>Attribute</b>	Write protect mask that disables bits from being changed with the <code>digio.writebit</code> and <code>digio.writeport</code> functions.
<b>Usage</b>	<code>mask = digio.writeprotect</code> -- Reads write protect mask. <code>digio.writeprotect = mask</code> -- Writes write protect mask. <code>mask</code> Set to the value that specifies the bit pattern for write protect.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Bits that are set to one cause the corresponding line to be write protected.</li> <li>• The binary equivalent of <code>mask</code> indicates the mask to be set for the I/O port. For example, a <code>mask</code> value of 7 has a binary equivalent 00000000000111. This mask write protects lines 1, 2 and 3.</li> </ul>
<b>Details</b>	See “Digital I/O port” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">digio.writebit</a> , <a href="#">digio.writeport</a>
<b>Example</b>	Write protects lines 1, 2, 3 and 4: <code>digio.writeprotect = 15</code>

## display functions and attributes

The functions and attributes in this group are used for various display operations, which are explained in [Section 14](#).

<b>display.clear</b>	
<b>Function</b>	Clears all lines of the display.
<b>Usage</b>	<code>display.clear()</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>This function will switch to the user screen and then clear the display. Annunciators are not affected.</li> <li>The <code>display.clear()</code>, <code>display.setcursor</code>, and <code>display.settext</code> functions are overlapped, non-blocking commands. That is, the script will NOT wait for one of these commands to complete. These non-blocking functions do not immediately update the display. For performance considerations, they write to a shadow and will update the display as soon as processing time becomes available.</li> </ul>
<b>Details</b>	See “ <a href="#">Clearing the display</a> ” on <a href="#">page 14-5</a> .
<b>Also see</b>	<a href="#">display.setcursor</a> , <a href="#">display.settext</a>

<b>display.getannunciators</b>																																																							
<b>Function</b>	Reads the annunciators that are presently turned on.																																																						
<b>Usage</b>	<pre>annun = display.getannunciators() annun</pre> Returns the bitmap value for annunciators that are turned on.																																																						
<b>Remarks</b>	<p>This function returns a bimap value that indicates which annunciators are turned on. The 16-bit binary equivalent of the returned value is the bitmap. For example, assume the returned value is 1028. The binary equivalent for this value is as follows:</p> <pre>0000010000000100</pre> <p>The above bitmap indicates that bits 3 and 11 are set. From the chart below, bit 3 and bit 11 corresponds to the annunciators that are turned on (<b>4W</b> and <b>REM</b>). Notice that the sum of the weighted values for bits 3 and 11 is the returned value (1028).</p> <table border="1"> <thead> <tr> <th>Annunciator</th> <th>Bit</th> <th>Weighted Value</th> <th>Annunciator</th> <th>Bit</th> <th>Weighted Value</th> </tr> </thead> <tbody> <tr> <td>FILT</td> <td>1</td> <td>1</td> <td>EDIT</td> <td>9</td> <td>256</td> </tr> <tr> <td>MATH</td> <td>2</td> <td>2</td> <td>ERR</td> <td>10</td> <td>512</td> </tr> <tr> <td>4W</td> <td>3</td> <td>4</td> <td>REM</td> <td>11</td> <td>1024</td> </tr> <tr> <td>AUTO</td> <td>4</td> <td>8</td> <td>TALK</td> <td>12</td> <td>2048</td> </tr> <tr> <td>ARM</td> <td>5</td> <td>16</td> <td>LSTN</td> <td>13</td> <td>4096</td> </tr> <tr> <td>TRIG</td> <td>6</td> <td>32</td> <td>SRQ</td> <td>14</td> <td>8192</td> </tr> <tr> <td>* (star)</td> <td>7</td> <td>64</td> <td>REAR</td> <td>15</td> <td>16384</td> </tr> <tr> <td>SMPL</td> <td>8</td> <td>128</td> <td>REL</td> <td>16</td> <td>32768</td> </tr> </tbody> </table>	Annunciator	Bit	Weighted Value	Annunciator	Bit	Weighted Value	FILT	1	1	EDIT	9	256	MATH	2	2	ERR	10	512	4W	3	4	REM	11	1024	AUTO	4	8	TALK	12	2048	ARM	5	16	LSTN	13	4096	TRIG	6	32	SRQ	14	8192	* (star)	7	64	REAR	15	16384	SMPL	8	128	REL	16	32768
Annunciator	Bit	Weighted Value	Annunciator	Bit	Weighted Value																																																		
FILT	1	1	EDIT	9	256																																																		
MATH	2	2	ERR	10	512																																																		
4W	3	4	REM	11	1024																																																		
AUTO	4	8	TALK	12	2048																																																		
ARM	5	16	LSTN	13	4096																																																		
TRIG	6	32	SRQ	14	8192																																																		
* (star)	7	64	REAR	15	16384																																																		
SMPL	8	128	REL	16	32768																																																		

<b>Details</b>	See “ <a href="#">Annunciators</a> ” on <a href="#">page 14-11</a> .
<b>Example</b>	<p>Reads the annunciators that are turned on:</p> <pre>annun = display.getannunciators() print(annun)</pre> <p>Output: 1.280000e+03</p> <p>For the returned value of 1280, the binary equivalent is 0000010100000000. Bits 9 and 11 are set. Using the above chart in “<b>Remarks</b>”, the <b>REM</b> and <b>EDIT</b> annunciators are turned on.</p>

<b>display.getcursor</b>	
<b>Function</b>	Reads the present position of the cursor for the user display.
<b>Usage</b>	<pre>row, column, style = display.getcursor()</pre> <p><code>row</code> Returns the row for the present cursor position.  <code>column</code> Returns the column for the present cursor position.  <code>style</code> Returns the cursor style.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function switches the display to the user screen, and then returns values to indicate row and column position, and cursor style.</li> <li>• The <code>row</code> value is returned as 1 (top row) or 2 (bottom row).</li> <li>• With the cursor in the top row, the <code>column</code> is returned as a value from 1 to 20. With the cursor in the bottom row, the <code>column</code> is returned as a value from 1 to 32. Columns are numbered from left to right on the display.</li> <li>• The returned value for style is 0 (invisible) or 1 (blink).</li> </ul>
<b>Details</b>	See “ <a href="#">Cursor position</a> ” on <a href="#">page 14-5</a> .
<b>Also see</b>	<a href="#">display.gettext</a> , <a href="#">display.screen</a> , <a href="#">display.setcursor</a> , <a href="#">display.settext</a>
<b>Example</b>	<p>Reads cursor position (row and column):</p> <pre>row, column = display.getcursor() print(row, column)</pre> <p>Output: 1.000000e+00 3.000000e+00</p> <p>The above output indicates that the cursor is in Row 1 at Column 3.</p>



<b>display.getlastkey</b>																																			
<b>Function</b>	Retrieves the keycode for the last pressed key.																																		
<b>Usage</b>	<code>key = display.getlastkey()</code>																																		
<b>Remarks</b>	<ul style="list-style-type: none"> <li>This read-only function returns the keycode for the last pressed key. <code>key</code> returns one of the following values: <table border="0"> <tbody> <tr> <td>0 (<code>display.KEY_NONE</code>)</td> <td>82 (<code>display.KEY_ENTER</code>)</td> </tr> <tr> <td>65 (<code>display.KEY_RANGEUP</code>)</td> <td>83 (<code>display.KEY_MEASB</code>)</td> </tr> <tr> <td>67 (<code>display.KEY_RELB</code>)</td> <td>84 (<code>display.KEY_DIGITSB</code>)</td> </tr> <tr> <td>68 (<code>display.KEY_MENU</code>)</td> <td>85 (<code>display.KEY_RECALL</code>)</td> </tr> <tr> <td>69 (<code>display.KEY_MODEA</code>)</td> <td>86 (<code>display.KEY_MEASA</code>)</td> </tr> <tr> <td>70 (<code>display.KEY_RELA</code>)</td> <td>87 (<code>display.KEY_DIGITSA</code>)</td> </tr> <tr> <td>71 (<code>display.KEY_RUN</code>)</td> <td>90 (<code>display.KEY_LIMITB</code>)</td> </tr> <tr> <td>72 (<code>display.KEY_DISPLAY</code>)</td> <td>91 (<code>display.KEY_SPEEDB</code>)</td> </tr> <tr> <td>73 (<code>display.KEY_AUTO</code>)</td> <td>92 (<code>display.KEY_TRIG</code>)</td> </tr> <tr> <td>74 (<code>display.KEY_FILTERB</code>)</td> <td>93 (<code>display.KEY_LIMITA</code>)</td> </tr> <tr> <td>75 (<code>display.KEY_EXIT</code>)</td> <td>94 (<code>display.KEY_SPEEDA</code>)</td> </tr> <tr> <td>76 (<code>display.KEY_SRCB</code>)</td> <td>95 (<code>display.KEY_LOAD</code>)</td> </tr> <tr> <td>77 (<code>display.KEY_FILTERA</code>)</td> <td>97 (<code>display.WHEEL_ENTER</code>)</td> </tr> <tr> <td>78 (<code>display.KEY_STORE</code>)</td> <td>103 (<code>display.KEY_RIGHT</code>)</td> </tr> <tr> <td>79 (<code>display.KEY_SRCB</code>)</td> <td>104 (<code>display.KEY_LEFT</code>)</td> </tr> <tr> <td>80 (<code>display.KEY_CONFIG</code>)</td> <td>114 (<code>display.WHEEL_RIGHT</code>)</td> </tr> <tr> <td>81 (<code>display.KEY_RANGEDOWN</code>)</td> <td></td> </tr> </tbody> </table> </li> <li>A history of the keycode for the last pressed front panel key is maintained by the Model 260x. When the instrument is powered-on, (or when transitioning from local to remote), the keycode is set to 0 (<code>display.KEY_NONE</code>).</li> <li>The <b>OUTPUT ON/OFF</b> keys for SMU A and SMU B cannot be tracked by this function.</li> <li>Pressing the <b>EXIT/LOCAL</b> key normally aborts a script. In order to use this function with the <b>EXIT</b> key, <code>display.locallockout</code> must be used.</li> </ul>	0 ( <code>display.KEY_NONE</code> )	82 ( <code>display.KEY_ENTER</code> )	65 ( <code>display.KEY_RANGEUP</code> )	83 ( <code>display.KEY_MEASB</code> )	67 ( <code>display.KEY_RELB</code> )	84 ( <code>display.KEY_DIGITSB</code> )	68 ( <code>display.KEY_MENU</code> )	85 ( <code>display.KEY_RECALL</code> )	69 ( <code>display.KEY_MODEA</code> )	86 ( <code>display.KEY_MEASA</code> )	70 ( <code>display.KEY_RELA</code> )	87 ( <code>display.KEY_DIGITSA</code> )	71 ( <code>display.KEY_RUN</code> )	90 ( <code>display.KEY_LIMITB</code> )	72 ( <code>display.KEY_DISPLAY</code> )	91 ( <code>display.KEY_SPEEDB</code> )	73 ( <code>display.KEY_AUTO</code> )	92 ( <code>display.KEY_TRIG</code> )	74 ( <code>display.KEY_FILTERB</code> )	93 ( <code>display.KEY_LIMITA</code> )	75 ( <code>display.KEY_EXIT</code> )	94 ( <code>display.KEY_SPEEDA</code> )	76 ( <code>display.KEY_SRCB</code> )	95 ( <code>display.KEY_LOAD</code> )	77 ( <code>display.KEY_FILTERA</code> )	97 ( <code>display.WHEEL_ENTER</code> )	78 ( <code>display.KEY_STORE</code> )	103 ( <code>display.KEY_RIGHT</code> )	79 ( <code>display.KEY_SRCB</code> )	104 ( <code>display.KEY_LEFT</code> )	80 ( <code>display.KEY_CONFIG</code> )	114 ( <code>display.WHEEL_RIGHT</code> )	81 ( <code>display.KEY_RANGEDOWN</code> )	
0 ( <code>display.KEY_NONE</code> )	82 ( <code>display.KEY_ENTER</code> )																																		
65 ( <code>display.KEY_RANGEUP</code> )	83 ( <code>display.KEY_MEASB</code> )																																		
67 ( <code>display.KEY_RELB</code> )	84 ( <code>display.KEY_DIGITSB</code> )																																		
68 ( <code>display.KEY_MENU</code> )	85 ( <code>display.KEY_RECALL</code> )																																		
69 ( <code>display.KEY_MODEA</code> )	86 ( <code>display.KEY_MEASA</code> )																																		
70 ( <code>display.KEY_RELA</code> )	87 ( <code>display.KEY_DIGITSA</code> )																																		
71 ( <code>display.KEY_RUN</code> )	90 ( <code>display.KEY_LIMITB</code> )																																		
72 ( <code>display.KEY_DISPLAY</code> )	91 ( <code>display.KEY_SPEEDB</code> )																																		
73 ( <code>display.KEY_AUTO</code> )	92 ( <code>display.KEY_TRIG</code> )																																		
74 ( <code>display.KEY_FILTERB</code> )	93 ( <code>display.KEY_LIMITA</code> )																																		
75 ( <code>display.KEY_EXIT</code> )	94 ( <code>display.KEY_SPEEDA</code> )																																		
76 ( <code>display.KEY_SRCB</code> )	95 ( <code>display.KEY_LOAD</code> )																																		
77 ( <code>display.KEY_FILTERA</code> )	97 ( <code>display.WHEEL_ENTER</code> )																																		
78 ( <code>display.KEY_STORE</code> )	103 ( <code>display.KEY_RIGHT</code> )																																		
79 ( <code>display.KEY_SRCB</code> )	104 ( <code>display.KEY_LEFT</code> )																																		
80 ( <code>display.KEY_CONFIG</code> )	114 ( <code>display.WHEEL_RIGHT</code> )																																		
81 ( <code>display.KEY_RANGEDOWN</code> )																																			
<b>Details</b>	See “ <a href="#">Sending keycodes</a> ” on <a href="#">page 14-16</a> .																																		
<b>Also see</b>	<a href="#">display.sendkey</a> , <a href="#">display.locallockout</a>																																		
<b>Example</b>	<p>On the front panel, press the <b>MENU</b> key and then send the following code:</p> <pre>key = display.getlastkey() print(key)</pre> <p>Output: 6.800000e+01</p>																																		

<b>display.gettext</b>	
<b>Function</b>	Reads the text presently displayed.
<b>Usage</b>	<p>There are five ways to use this function:</p> <pre>text = display.gettext() text = display.gettext(embellished) text = display.gettext(embellished, row) text = display.gettext(embellished, row, column_start) text = display.gettext(embellished, row, column_start, column_end)</pre> <p><code>embellished</code>      Set to <code>false</code> to return text as a simple character string. Set to <code>true</code> to include all character codes.</p> <p><code>row</code>                Set to <code>1</code> or <code>2</code> to select which row to read text. If not included, text from both rows are read.</p> <p><code>column_start</code>      Set to starting column for reading text. Default is <code>1</code>.</p> <p><code>column_end</code>        Set to ending column for reading text. Default is <code>20</code> (Row 1) or <code>32</code> (Row 2).</p> <p>Note: The range of valid column numbers depends on which row is specified. For Row 1, valid column numbers are 1 to 20. For Row 2, valid column numbers are 1 to 32.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Sending the command without any parameters returns both lines of the display. The <code>\$N</code> character code will be included to show where the top line ends and the bottom line begins.</li> <li>• With <code>embellished</code> set to <code>true</code>, all other character codes will be returned along with the message. With <code>embellished</code> set to <code>false</code>, only the message and the <code>\$N</code> character code will be returned. See the <a href="#">display.settext</a> function for details on the character codes.</li> <li>• The display will not be switched to the user screen. Text will be read from the active screen.</li> </ul>
<b>Details</b>	See “ <a href="#">Displaying text messages</a> ” on <a href="#">page 14-6</a> .
<b>Also see</b>	<a href="#">display.getcursor</a> , <a href="#">display.setcursor</a> , <a href="#">display.settext</a>
<b>Example</b>	<p>Returns all text in both lines of the display:</p> <pre>text = display.gettext() print(text)</pre> <p>Output: User Screen                      \$N</p> <p>The above output indicates that the message “User Screen” is on the top line. The bottom line is blank.</p>

<b>display.inputvalue</b>	
<b>Function</b>	Displays a formatted input field that the operator can edit.
<b>Usage</b>	<p>There are four ways to use this function:</p> <pre>value = display.inputvalue(format) value = display.inputvalue(format, default) value = display.inputvalue(format, default, min) value = display.inputvalue(format, default, min, max)</pre> <p><code>format</code>      Define format string for the input field using '0's, the decimal point (.), polarity sign (+) and 'E' for exponent.</p> <p><code>default</code>      Set the default value for the parameter.</p> <p><code>min</code>            Set the minimum input value that can be set.</p> <p><code>max</code>            Set the maximum input value that can be set.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will make use of text to create an editable input field on the user screen at the present cursor position. The first write to the display after poweron will clear the user screen.</li> <li>• Examples of the input field:  +0.00    00    +00.0000E+00    0.00000E+0</li> </ul> <p><b>Value field:</b></p> <ul style="list-style-type: none"> <li>+      Include a "+" sign for positive/negative value entry. Not including the "+" sign prevents negative value entry.</li> <li>0's'    Defines the digit positions for the value. Up to six '0's can be used for the value (as shown above in the third and fourth examples).</li> <li>.      If used, include the decimal point (.) where needed for the value.</li> </ul> <p><b>Exponent field (optional):</b></p> <ul style="list-style-type: none"> <li>E      Include the "E" for exponent entry.</li> <li>+      Include a "+" sign for positive/negative exponent entry. Not including the "+" sign prevents negative exponent entry.</li> <li>0's'    Defines the digit positions for the exponent.</li> </ul> <ul style="list-style-type: none"> <li>• Along with specifying the <code>format</code> for the input field, there are options to specify minimum and maximum limits for the input field. When NOT using the "+" sign for the value field, the minimum limit cannot be set to less than zero. When using the "+" sign, the minimum limit can set to less than zero (e.g., -2).</li> <li>• There is also an option to specify a default value. When this command is executed, the initially displayed value for the field will be the <code>default</code> value.</li> <li>• Message prompts to instruct the operator should be displayed prior to calling this function. Make sure to position the cursor where the edit field should appear.</li> <li>• The input value is limited to <math>\pm 1e37</math>.</li> </ul>

<b>Remarks (cont.)</b>	<ul style="list-style-type: none"> <li>• After sending this command, script execution holds up and waits for the operator to enter a value and press <b>ENTER</b>: <ul style="list-style-type: none"> <li>• If limits are used, the operator will not be able to input values outside the minimum and maximum limits.</li> <li>• For positive and negative entry (“+” sign used for the value field and/or the exponent field), polarity of a non-zero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the wheel. Polarity will also toggle when using the wheel to decrease or increase the value or exponent past zero. A zero value or exponent (e.g. +00) is always positive and cannot be toggled to negative polarity.</li> </ul> </li> <li>• After sending this command and pressing the <b>EXIT</b> key, <code>value</code> will return <code>nil</code>.</li> </ul>
<b>Details</b>	See “ <a href="#">Parameter value prompting</a> ” on <a href="#">page 14-10</a> .
<b>Also see</b>	<a href="#">display.prompt</a> , <a href="#">display.setcursor</a> , <a href="#">display.settext</a>
<b>Example</b>	Displays an editable field (“+0.50”) for operator input – Valid input range is -0.10 to +2.00, with a default of 0.50: <pre>display.clear() value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)</pre>

## display.loadmenu.add

<b>Function</b>	Adds an entry to the “USER TESTS” submenu of the “LOAD TEST” menu.
<b>Usage</b>	<p>There are two ways to use this function:</p> <pre>display.loadmenu.add(displayname, chunk) display.loadmenu.add(displayname, chunk, memory)</pre> <p><code>displayname</code>      Name to display in the menu.  <code>chunk</code>                Chunk is the code to be executed.  <code>memory</code>              Save or don't save <code>chunk</code> and <code>displayname</code> in non-volatile memory.</p> <p>Set <code>memory</code> to one of the following values:  0 or <code>display.DONT_SAVE</code>  1 or <code>display.SAVE</code>  The default <code>memory</code> setting is <code>display.SAVE</code>.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function adds an entry to the “USER TESTS” submenu of the “LOAD TEST” menu. If the given item is subsequently selected via the front panel, the <code>chunk</code> will be executed when the <b>RUN</b> key is pressed.</li> <li>• The <code>chunk</code> can be made up of scripts, functions, variables and commands. With <code>memory</code> set to <code>display.SAVE</code>, commands are saved with the <code>chunk</code> in non-volatile memory. Scripts, functions and variables used in the <code>chunk</code> are not saved by <code>display.SAVE</code>. Functions and variables need to be saved along with the script (see “<a href="#">Saving a user script</a>” <a href="#">page 14-13</a>). If the script is not saved in non-volatile memory, it will be lost when the Model 260x is turned off. See <b>Example 1</b> below.</li> <li>• It does not matter what order the menu items are added. They will be displayed in alphabetical order when the “USER TESTS” menu is selected.</li> </ul>

<b>Details</b>	See “Load test menu” on page 14-13.
<b>Also see</b>	<a href="#">display.loadmenu.delete</a>
<b>Examples</b>	<p><b>Example 1</b> – Assume a script with a function named “DUT1” has already been loaded into the Model 260x, and the script has NOT been saved in non-volatile memory. Now assume you want to add a test named “Test” to the USER TESTS menu. You want the test to run the function named “DUT1” and sound the beeper. The following command will add “Test” to the menu, define the chunk, and then save <code>displayname</code> and chunk in non-volatile memory:</p> <pre>display.loadmenu.add("Test", "DUT1() beeper.beep(2, 500)", display.SAVE)</pre> <p>When “Test” is run from the front panel USER TESTS menu, the function named “DUT1” will execute and the beeper will beep for two seconds.</p> <p>Now assume you cycle power on the Model 260x. Since the script was not saved in non-volatile memory, the function named “DUT1” is lost. When “Test” is again run from the front panel, the beeper will beep, but “DUT1” will not execute because it no longer exists in the chunk.</p> <p><b>Example 2</b> – Adds entry called “Part1” to the front panel “USER TESTS” load menu for the chunk “testpart([[Part1]], 5.0)”, and saves it in non-volatile memory:</p> <pre>display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)", display.SAVE)</pre>

<b>display.loadmenu.delete</b>	
<b>Function</b>	Deletes an entry from the “USER” submenu of the “LOAD TEST” menu.
<b>Usage</b>	<code>display.loadmenu.delete(displayname)</code> <code>displayname</code> Name to remove from the menu.
<b>Remarks</b>	This function is used to delete an entry ( <code>displayname</code> ) from the front panel USER TESTS submenu of the LOAD TEST menu.
<b>Details</b>	See “Load test menu” on page 14-13.
<b>Also see</b>	<a href="#">display.loadmenu.add</a>
<b>Example</b>	Removes the entry named “Part1” from the front panel “USER TESTS” load menu: <code>display.loadmenu.delete("Part1")</code>

<b>display.locallockout</b>	
<b>Attribute</b>	<b>LOCAL</b> key disabled.
<b>Usage</b>	<pre>lockout = display.locallockout    -- Reads state of lockout. display.locallockout = lockout    -- Writes state of lockout.  Set lockout to one of the following values: 0 or display.UNLOCK    Unlocks <b>LOCAL</b> key. 1 or display.LOCK     Locks out <b>LOCAL</b> key.</pre>
<b>Remarks</b>	Setting <code>display.locallockout</code> to <code>display.LOCK</code> prevents the user from interrupting remote operation by pressing the <b>LOCAL</b> key. Set this attribute to <code>display.UNLOCK</code> to allow the <b>LOCAL</b> key to abort script/remote operation.
<b>Details</b>	See “ <a href="#">LOCAL lockout</a> ” on page 14-12.
<b>Example</b>	Disables the front panel <b>LOCAL</b> key: <pre>display.locallockout = display.LOCK</pre>

<b>display.menu</b>	
<b>Function</b>	Presents a menu on the front panel display.
<b>Usage</b>	<pre>selection = display.menu(name, items)   name           Menu name to display on the top line.   items          Menu items to display on the bottom line.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The menu consists of the menu <code>name</code> string on the top line, and a selectable list of <code>items</code> on the bottom line. The menu items must be a single string with each item separated by whitespace. The <code>name</code> for the top line is limited to 20 characters.</li> <li>• After sending this command, script execution holds up and waits for the operator to select a menu item. An item is selected by rotating the wheel (or using the cursor keys) to place the blinking cursor on the item, and then pressing the wheel (or <b>ENTER</b> key). When an item is selected, the text of that selection is returned.</li> <li>• Pressing the <b>EXIT</b> key will not abort the script while the menu is displayed, but it will return <code>nil</code>. The script can be aborted by calling the <code>exit</code> function when <code>nil</code> is returned.</li> </ul>
<b>Details</b>	See “ <a href="#">Menu</a> ” on page 14-8.
<b>Example</b>	Displays a menu with three menu items. If the second menu item is selected, <code>selection</code> will be given the value <code>Test2</code> : <pre>selection = display.menu("Sample Menu", "Test1 Test2 Test3") print(selection) Output: Test2</pre>

<b>display.prompt</b>	
<b>Function</b>	Prompts the user to enter a parameter from the front panel.
<b>Usage</b>	<p>There are four ways to use this function:</p> <pre>value = display.prompt(format, units, help) value = display.prompt(format, units, help, default) value = display.prompt(format, units, help, default, min) value = display.prompt(format, units, help, default, min, max)</pre> <p><code>format</code>        Define format string for the input field using '0's, the decimal point (.), polarity sign (+) and 'E' for exponent.</p> <p><code>units</code>         Set units text string for top line (8 characters maximum).</p> <p><code>help</code>          Text string to display on the bottom line (32 characters maximum).</p> <p><code>default</code>       Set the default value for the parameter.</p> <p><code>min</code>          Set the minimum input value that can be set.</p> <p><code>max</code>          Set the maximum input value that can be set.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will create an editable input field at the present cursor position, and an input prompt message on the bottom line. Example of a displayed input field and prompt: 0.00V Input 0 to +2V</li> <li>• The <code>format</code> configures the editable input field. Four examples for the format: +0.00    00    +00.0000E+00    0.00000E+0</li> </ul> <p><b>Value field:</b></p> <ul style="list-style-type: none"> <li><code>+</code>    Include a "+" sign for positive/negative value entry. Not including the "+" sign prevents negative value entry.</li> <li><code>0's'</code>    Defines the digit positions for the value. Up to six '0's can be used for the value (as shown above in the third and fourth examples).</li> <li><code>.</code>    If used, include the decimal point (.) where needed for the value.</li> </ul> <p><b>Exponent field (optional):</b></p> <ul style="list-style-type: none"> <li><code>E</code>    Include the "E" for exponent entry.</li> <li><code>+</code>    Include a "+" sign for positive/negative exponent entry. Not including the "+" sign prevents negative exponent entry.</li> <li><code>0's'</code>    Defines the digit positions for the exponent.</li> </ul>

<b>Remarks (cont.)</b>	<ul style="list-style-type: none"> <li>• <code>units</code> is a string that indicates the units (e.g., “V” or “A”) for the value and <code>help</code> provides a message prompt on the bottom line.</li> <li>• Along with specifying the <code>format</code> for the input field, there are options to specify minimum and maximum limits for the input field. When NOT using the “+” sign for the value field, the minimum limit cannot be set to less than zero. When using the “+” sign, the minimum limit can set to less than zero (e.g., -2).</li> <li>• There is also an option to specify a default value. When this command is executed, the initially displayed value for the field will be the <code>default</code> value.</li> <li>• Message prompts to instruct the operator should be displayed prior to calling this function. Make sure to position the cursor where the edit field should appear.</li> <li>• The input value is limited to <math>\pm 1e37</math>.</li> <li>• After sending this command, script execution holds and waits for the operator to enter a value and press <b>ENTER</b>: <ul style="list-style-type: none"> <li>• If limits are used, the operator will not be able to input values outside the minimum and maximum limits.</li> <li>• For positive and negative entry (“+” sign used for the value field and/or the exponent field), polarity of a non-zero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the wheel. Polarity will also toggle when using the wheel to decrease or increase the value or exponent past zero. A zero value or exponent (e.g. +00) is always positive and cannot be toggled to negative polarity.</li> </ul> </li> <li>• After sending this command and pressing the <b>EXIT</b> key, <code>value</code> will return <code>nil</code>.</li> </ul>
<b>Details</b>	See “ <a href="#">Parameter value prompting</a> ” on <a href="#">page 14-10</a> .
<b>Also see</b>	<a href="#">display.inputvalue</a>
<b>Example</b>	<p>Prompts the operator to enter a voltage value – Valid input range is 0 to +2.00, with a default of 0.50:</p> <pre>value = display.prompt("0.00", "V", "Input 0 to +2V" 0.5, 0, 2)</pre> <p>The above command will display the following input prompt:</p> <pre>0.50V Input 0 to +2V</pre>

## display.screen

<b>Attribute</b>	The selected display screen.								
<b>Usage</b>	<pre>displayid = display.screen      -- Reads display screen. display.screen = displayid     -- Writes display screen.</pre> <p>Set <code>displayid</code> to one of the following values:</p> <table data-bbox="386 1447 1308 1567"> <tr> <td>0 or <code>display.SMUA</code></td> <td>Displays source-measure and compliance for SMU A.</td> </tr> <tr> <td>1 or <code>display.SMUB</code></td> <td>Displays source-measure and compliance for SMU B.</td> </tr> <tr> <td>2 or <code>display.SMUA_SMUB</code></td> <td>Displays source-measure for SMU A and SMU B.</td> </tr> <tr> <td>3 or <code>display.USER</code></td> <td>Displays the user screen.</td> </tr> </table>	0 or <code>display.SMUA</code>	Displays source-measure and compliance for SMU A.	1 or <code>display.SMUB</code>	Displays source-measure and compliance for SMU B.	2 or <code>display.SMUA_SMUB</code>	Displays source-measure for SMU A and SMU B.	3 or <code>display.USER</code>	Displays the user screen.
0 or <code>display.SMUA</code>	Displays source-measure and compliance for SMU A.								
1 or <code>display.SMUB</code>	Displays source-measure and compliance for SMU B.								
2 or <code>display.SMUA_SMUB</code>	Displays source-measure for SMU A and SMU B.								
3 or <code>display.USER</code>	Displays the user screen.								



<b>Remarks</b>	Setting this attribute selects the display screen for the front panel. This attribute can be read to determine which of the four available display screens was last selected by the user. The user can select the screen by value or one of the enumerations.
<b>Details</b>	See “ <a href="#">Display screen</a> ” on <a href="#">page 14-3</a> .
<b>Example</b>	Selects the source-measure and compliance limit display for SMUA: <code>display.screen = display.SMUA</code>

## display.sendkey

<b>Function</b>	Sends a keycode to simulate the action of a front panel control.																																																																																																												
<b>Usage</b>	<p><code>display.sendkey(keycode)</code></p> <p>Set <code>keycode</code> to one of the values shown below:</p> <table> <tbody> <tr> <td><code>display.KEY_AUTO</code></td> <td>or</td> <td>73</td> <td><code>display.KEY_OUTPUTA</code></td> <td>or</td> <td>88</td> </tr> <tr> <td><code>display.KEY_CONFIG</code></td> <td>or</td> <td>80</td> <td><code>display.KEY_OUTPUTB</code></td> <td>or</td> <td>96</td> </tr> <tr> <td><code>display.KEY_DIGITSA</code></td> <td>or</td> <td>87</td> <td><code>display.KEY_RANGEDOWN</code></td> <td>or</td> <td>81</td> </tr> <tr> <td><code>display.KEY_DIGITSB</code></td> <td>or</td> <td>84</td> <td><code>display.KEY_RANGEUP</code></td> <td>or</td> <td>65</td> </tr> <tr> <td><code>display.KEY_DISPLAY</code></td> <td>or</td> <td>72</td> <td><code>display.KEY_RECALL</code></td> <td>or</td> <td>85</td> </tr> <tr> <td><code>display.KEY_ENTER</code></td> <td>or</td> <td>82</td> <td><code>display.KEY_RELA</code></td> <td>or</td> <td>70</td> </tr> <tr> <td><code>display.KEY_EXIT</code></td> <td>or</td> <td>75</td> <td><code>display.KEY_RELB</code></td> <td>or</td> <td>67</td> </tr> <tr> <td><code>display.KEY_FILTERA</code></td> <td>or</td> <td>77</td> <td><code>display.KEY_RIGHT</code></td> <td>or</td> <td>103</td> </tr> <tr> <td><code>display.KEY_FILTERB</code></td> <td>or</td> <td>74</td> <td><code>display.KEY_RUN</code></td> <td>or</td> <td>71</td> </tr> <tr> <td><code>display.KEY_LEFT</code></td> <td>or</td> <td>104</td> <td><code>display.KEY_SPEEDA</code></td> <td>or</td> <td>94</td> </tr> <tr> <td><code>display.KEY_LIMITA</code></td> <td>or</td> <td>93</td> <td><code>display.KEY_SPEEDB</code></td> <td>or</td> <td>91</td> </tr> <tr> <td><code>display.KEY_LIMITB</code></td> <td>or</td> <td>90</td> <td><code>display.KEY_SRC A</code></td> <td>or</td> <td>79</td> </tr> <tr> <td><code>display.KEY_LOAD</code></td> <td>or</td> <td>95</td> <td><code>display.KEY_SRCB</code></td> <td>or</td> <td>76</td> </tr> <tr> <td><code>display.KEY_MEASA</code></td> <td>or</td> <td>86</td> <td><code>display.KEY_STORE</code></td> <td>or</td> <td>78</td> </tr> <tr> <td><code>display.KEY_MEASB</code></td> <td>or</td> <td>83</td> <td><code>display.KEY_TRIG</code></td> <td>or</td> <td>92</td> </tr> <tr> <td><code>display.KEY_MENU</code></td> <td>or</td> <td>68</td> <td><code>display.WHEEL_ENTER</code></td> <td>or</td> <td>97</td> </tr> <tr> <td><code>display.KEY_MODEA</code></td> <td>or</td> <td>69</td> <td><code>display.WHEEL_LEFT</code></td> <td>or</td> <td>107</td> </tr> <tr> <td><code>display.KEY_MODEB</code></td> <td>or</td> <td>66</td> <td><code>display.WHEEL_RIGHT</code></td> <td>or</td> <td>114</td> </tr> </tbody> </table>	<code>display.KEY_AUTO</code>	or	73	<code>display.KEY_OUTPUTA</code>	or	88	<code>display.KEY_CONFIG</code>	or	80	<code>display.KEY_OUTPUTB</code>	or	96	<code>display.KEY_DIGITSA</code>	or	87	<code>display.KEY_RANGEDOWN</code>	or	81	<code>display.KEY_DIGITSB</code>	or	84	<code>display.KEY_RANGEUP</code>	or	65	<code>display.KEY_DISPLAY</code>	or	72	<code>display.KEY_RECALL</code>	or	85	<code>display.KEY_ENTER</code>	or	82	<code>display.KEY_RELA</code>	or	70	<code>display.KEY_EXIT</code>	or	75	<code>display.KEY_RELB</code>	or	67	<code>display.KEY_FILTERA</code>	or	77	<code>display.KEY_RIGHT</code>	or	103	<code>display.KEY_FILTERB</code>	or	74	<code>display.KEY_RUN</code>	or	71	<code>display.KEY_LEFT</code>	or	104	<code>display.KEY_SPEEDA</code>	or	94	<code>display.KEY_LIMITA</code>	or	93	<code>display.KEY_SPEEDB</code>	or	91	<code>display.KEY_LIMITB</code>	or	90	<code>display.KEY_SRC A</code>	or	79	<code>display.KEY_LOAD</code>	or	95	<code>display.KEY_SRCB</code>	or	76	<code>display.KEY_MEASA</code>	or	86	<code>display.KEY_STORE</code>	or	78	<code>display.KEY_MEASB</code>	or	83	<code>display.KEY_TRIG</code>	or	92	<code>display.KEY_MENU</code>	or	68	<code>display.WHEEL_ENTER</code>	or	97	<code>display.KEY_MODEA</code>	or	69	<code>display.WHEEL_LEFT</code>	or	107	<code>display.KEY_MODEB</code>	or	66	<code>display.WHEEL_RIGHT</code>	or	114
<code>display.KEY_AUTO</code>	or	73	<code>display.KEY_OUTPUTA</code>	or	88																																																																																																								
<code>display.KEY_CONFIG</code>	or	80	<code>display.KEY_OUTPUTB</code>	or	96																																																																																																								
<code>display.KEY_DIGITSA</code>	or	87	<code>display.KEY_RANGEDOWN</code>	or	81																																																																																																								
<code>display.KEY_DIGITSB</code>	or	84	<code>display.KEY_RANGEUP</code>	or	65																																																																																																								
<code>display.KEY_DISPLAY</code>	or	72	<code>display.KEY_RECALL</code>	or	85																																																																																																								
<code>display.KEY_ENTER</code>	or	82	<code>display.KEY_RELA</code>	or	70																																																																																																								
<code>display.KEY_EXIT</code>	or	75	<code>display.KEY_RELB</code>	or	67																																																																																																								
<code>display.KEY_FILTERA</code>	or	77	<code>display.KEY_RIGHT</code>	or	103																																																																																																								
<code>display.KEY_FILTERB</code>	or	74	<code>display.KEY_RUN</code>	or	71																																																																																																								
<code>display.KEY_LEFT</code>	or	104	<code>display.KEY_SPEEDA</code>	or	94																																																																																																								
<code>display.KEY_LIMITA</code>	or	93	<code>display.KEY_SPEEDB</code>	or	91																																																																																																								
<code>display.KEY_LIMITB</code>	or	90	<code>display.KEY_SRC A</code>	or	79																																																																																																								
<code>display.KEY_LOAD</code>	or	95	<code>display.KEY_SRCB</code>	or	76																																																																																																								
<code>display.KEY_MEASA</code>	or	86	<code>display.KEY_STORE</code>	or	78																																																																																																								
<code>display.KEY_MEASB</code>	or	83	<code>display.KEY_TRIG</code>	or	92																																																																																																								
<code>display.KEY_MENU</code>	or	68	<code>display.WHEEL_ENTER</code>	or	97																																																																																																								
<code>display.KEY_MODEA</code>	or	69	<code>display.WHEEL_LEFT</code>	or	107																																																																																																								
<code>display.KEY_MODEB</code>	or	66	<code>display.WHEEL_RIGHT</code>	or	114																																																																																																								
<b>Remarks</b>	Sending this command simulates the pressing of a front panel key or wheel, or turning the Wheel one click to the left or right.																																																																																																												
<b>Details</b>	See “ <a href="#">Sending keycodes</a> ” on <a href="#">page 14-16</a> .																																																																																																												
<b>Example</b>	“Press” the RUN key: <code>display.sendkey(display.KEY_RUN)</code>																																																																																																												

<b>display.setcursor</b>	
<b>Function</b>	Sets the position of the cursor.
<b>Usage</b>	<p>There are two ways to use this function:</p> <pre>display.setcursor(row, column) display.setcursor(row, column, style)</pre> <p><code>row</code>               Set row number for the cursor (1 or 2).</p> <p><code>column</code>           Set column number for the cursor. For row 1, column can be set from 1 to 20. For row 2, column can be set from 1 to 32.</p> <p><code>style</code>             Set cursor style to be invisible (0) or blink (1).</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Sending this command selects the user screen and then moves the cursor to the given location.</li> <li>• An out of range parameter for <code>row</code> will set the cursor to row 2. An out of range parameter for <code>column</code> will set the cursor to column 20 (for row 1) or 32 (for row 2).</li> <li>• An out of range parameter for <code>style</code> sets it to 0 (invisible).</li> <li>• A blinking cursor will only be visible when it is positioned over displayed text. It cannot be seen when positioned over a space character.</li> <li>• The <code>display.clear</code>, <code>display.setcursor</code>, and <code>display.settext</code> functions are overlapped, non-blocking commands. That is, the script will NOT wait for one of these commands to complete. These non-blocking functions do not immediately update the display. For performance considerations, they write to a shadow and will update the display as soon as processing time becomes available.</li> </ul>
<b>Details</b>	See “ <a href="#">Cursor position</a> ” on <a href="#">page 14-5</a> .
<b>Also see</b>	<a href="#">display.clear</a> , <a href="#">display.getcursor</a> , <a href="#">display.gettext</a> , <a href="#">display.settext</a>
<b>Example</b>	Positions cursor on row 2 column 1: <code>display.setcursor(2, 1)</code>

<b>display.settext</b>	
<b>Function</b>	Displays text on the user screen.
<b>Usage</b>	<code>display.settext(text)</code> <code>text</code> Text message string to be displayed.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function selects the user display screen, and displays the given text. The first write to the display after poweron will clear the user screen.</li> <li>• The text starts at the present cursor position. After the text is displayed, the cursor will be located after the last character in the display message.</li> <li>• Top line text will not wrap to the bottom line of the display automatically. Any text that does not fit on the current line will be truncated. If the text is truncated, the cursor will be left at the end of the line.</li> <li>• The text remains on the display until replaced or cleared.</li> <li>• The following character codes can be also be included in the <code>text</code> string: <ul style="list-style-type: none"> <li>\$N Newline – Starts text on the next line. If the cursor is already on line 2, text will be ignored after the '\$N' is received.</li> <li>\$R Sets text to Normal.</li> <li>\$B Sets text to Blink.</li> <li>\$D Sets text to Dim intensity.</li> <li>\$F Sets text to background blink.</li> <li>\$\$ Escape sequence to display a single "\$".</li> </ul> </li> <li>• The <code>display.clear</code>, <code>display.setcursor</code>, and <code>display.settext</code> functions are overlapped, non-blocking commands. That is, the script will NOT wait for one of these commands to complete. These non-blocking functions do not immediately update the display. For performance considerations, they write to a shadow and will update the display as soon as processing time becomes available.</li> </ul>
<b>Details</b>	See <a href="#">“Displaying text messages”</a> on <a href="#">page 14-6</a> .
<b>Also see</b>	<a href="#">display.clear</a> , <a href="#">display.getcursor</a> , <a href="#">display.gettext</a> , <a href="#">display.setcursor</a>
<b>Example</b>	<p>Displays a message on the user screen:</p> <pre>display.clear() display.settext("Message Test \$N\$Bwith Row 2 Blinking")</pre> <p>The top line displays “Message Test” and the bottom line displays the blinking message “with Row 2 Blinking”.</p>

<b>display.smuX.digits</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	The selected measurement display resolution.
<b>Usage</b>	<pre>digits = display.smuX.digits      -- Reads resolution. display.smuX.digits = digits      -- Writes resolution.</pre> <p>Set <code>digits</code> to one of the following values:</p> <pre>4 or display.DIGITS_4_5      Selects 4-1/2d resolution. 5 or display.DIGITS_5_5      Selects 5-1/2d resolution. 6 or display.DIGITS_6_5      Selects 6-1/2d resolution.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute selects the measurement display resolution; 4-1/2 digit, 5-1/2 digit or 6-1/2 digit.</li> <li>• SMU A and SMU B can be set for a different measurement display resolution.</li> </ul>
<b>Details</b>	See “ <a href="#">Display resolution</a> ” on <a href="#">page 14-4</a> .
<b>Example</b>	Selects 5-1/2d resolution for SMU A: <pre>display.smua.digits = display.DIGITS_5_5</pre>

<b>display.smuX.measure.func</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	The type of measurement being displayed.
<b>Usage</b>	<pre>func = display.smuX.measure.func  -- Reads function. display.smuX.measure.func = func  -- Writes function.</pre> <p>Set <code>func</code> to one of the following values:</p> <pre>0 or display.MEASURE_DCAMPS      Selects current measure function. 1 or display.MEASURE_DCVOLTS     Selects volts measure function. 2 or display.MEASURE_OHMS        Selects ohms measure function. 3 or display.MEASURE_WATTS       Selects power measure function.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Selects the displayed measurement function: Amps, volts, ohms or watts.</li> <li>• SMU A and SMU B can be set for different measurement functions.</li> </ul>
<b>Details</b>	See “ <a href="#">Measurement functions</a> ” on <a href="#">page 14-3</a> .
<b>Example</b>	Selects the current measure function for SMU A: <pre>display.smua.measure.func = display.MEASURE_DCAMPS</pre>

<b>display.trigger.clear</b>	
<b>Function</b>	Clears the front panel trigger event detector.
<b>Usage</b>	<code>display.trigger.clear()</code>
<b>Remarks</b>	The trigger event detector remembers if an event has been detected since the last <code>display.trigger.wait</code> call. This function clears the trigger's event detector and discards the previous history of <b>TRIG</b> key presses.
<b>Details</b>	See “Display triggering” on page 14-15.
<b>Also see</b>	<a href="#">display.trigger.wait</a>

<b>display.trigger.wait</b>	
<b>Function</b>	Waits for the <b>TRIG</b> key on the front panel to be pressed.
<b>Usage</b>	<pre>triggered = display.trigger.wait(timeout) timeout      Set timeout in seconds. triggered    Returns a true if a trigger was detected. Returns a false if the               operation timed out.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will wait for the <b>TRIG</b> key on the front panel to be pressed. If the trigger key was previously pressed and one or more trigger events were detected, this function will return immediately.</li> <li>• After waiting for a trigger with this function, the event detector will be automatically reset and rearmed. This is true regardless of the number of events detected.</li> <li>• Use the <code>display.trigger.clear</code> call to clear the trigger event detector.</li> </ul>
<b>Details</b>	See “Display triggering” on page 14-15.
<b>Also see</b>	<a href="#">display.trigger.clear</a>
<b>Example</b>	<p>Waits up to five seconds for the <b>TRIG</b> key to be pressed:</p> <pre>triggered = display.trigger.wait(5) print(triggered)</pre> <p>Output: true</p> <p>The above output indicates that the <b>TRIG</b> key was pressed (trigger detected) before the five second timeout expired.</p>

<b>display.waitkey</b>																																																																															
<b>Function</b>	Captures the keycode value for the next key press.																																																																														
<b>Usage</b>	<code>key = display.waitkey()</code>																																																																														
<b>Remarks</b>	<ul style="list-style-type: none"> <li>After sending this function, script execution will hold up until a front panel key or the wheel is pressed, or the wheel is turned to the right or left. After pressing a control or turning the wheel, the keycode value for that key will be returned. The chart shown below lists the keycode value for each front panel control. The controls are listed alphabetically.</li> <li>If the <b>EXIT</b> key is pressed while this function is waiting for a keypress, the script will not be aborted.</li> <li>A typical use for this function is to prompt the user to press <b>EXIT</b> to abort the script or press any other key to continue. If keycode value 75 is returned (<b>EXIT</b> key pressed), then the <code>exit()</code> function can be called to abort the script. Sample code for this process is provided in “<a href="#">Capturing key-press codes</a>” on <a href="#">page 14-17</a>.</li> </ul> <table border="1"> <thead> <tr> <th>Control</th> <th>Keycode</th> <th>Control</th> <th>Keycode</th> <th>Control</th> <th>Keycode</th> </tr> </thead> <tbody> <tr> <td>AUTO</td> <td>73</td> <td>LIMIT (B)</td> <td>90</td> <td>REL (A)</td> <td>70</td> </tr> <tr> <td>CONFIG</td> <td>80</td> <td>LOAD</td> <td>95</td> <td>REL (B)</td> <td>67</td> </tr> <tr> <td>CURSOR (left)</td> <td>104</td> <td>MEAS (A)</td> <td>86</td> <td>RUN</td> <td>71</td> </tr> <tr> <td>CURSOR (right)</td> <td>103</td> <td>MEAS (B)</td> <td>83</td> <td>SPEED (A)</td> <td>94</td> </tr> <tr> <td>DIGITS (A)</td> <td>87</td> <td>MENU</td> <td>68</td> <td>SPEED (B)</td> <td>91</td> </tr> <tr> <td>DIGITS (B)</td> <td>84</td> <td>MODE (A)</td> <td>69</td> <td>SRC (A)</td> <td>79</td> </tr> <tr> <td>DISPLAY</td> <td>72</td> <td>MODE (B)</td> <td>66</td> <td>SRC (B)</td> <td>76</td> </tr> <tr> <td>ENTER</td> <td>82</td> <td>OUTPUT (A)</td> <td>88</td> <td>STORE</td> <td>78</td> </tr> <tr> <td>EXIT</td> <td>75</td> <td>OUTPUT (B)</td> <td>96</td> <td>TRIG</td> <td>92</td> </tr> <tr> <td>FILTER (A)</td> <td>77</td> <td>RANGE (down)</td> <td>81</td> <td>Wheel (press)</td> <td>97</td> </tr> <tr> <td>FILTER (B)</td> <td>74</td> <td>RANGE (up)</td> <td>65</td> <td>Wheel (left)</td> <td>107</td> </tr> <tr> <td>LIMIT (A)</td> <td>93</td> <td>RECALL</td> <td>85</td> <td>Wheel (right)</td> <td>114</td> </tr> </tbody> </table> <p>The above chart lists the numeric keycode values for the front panel controls. The key-code value identifiers are listed in the documentation for <a href="#">display.sendkey</a> (e.g., <code>display.KEY_RUN</code> is the identifier for the <b>RUN</b> key)</p>	Control	Keycode	Control	Keycode	Control	Keycode	AUTO	73	LIMIT (B)	90	REL (A)	70	CONFIG	80	LOAD	95	REL (B)	67	CURSOR (left)	104	MEAS (A)	86	RUN	71	CURSOR (right)	103	MEAS (B)	83	SPEED (A)	94	DIGITS (A)	87	MENU	68	SPEED (B)	91	DIGITS (B)	84	MODE (A)	69	SRC (A)	79	DISPLAY	72	MODE (B)	66	SRC (B)	76	ENTER	82	OUTPUT (A)	88	STORE	78	EXIT	75	OUTPUT (B)	96	TRIG	92	FILTER (A)	77	RANGE (down)	81	Wheel (press)	97	FILTER (B)	74	RANGE (up)	65	Wheel (left)	107	LIMIT (A)	93	RECALL	85	Wheel (right)	114
Control	Keycode	Control	Keycode	Control	Keycode																																																																										
AUTO	73	LIMIT (B)	90	REL (A)	70																																																																										
CONFIG	80	LOAD	95	REL (B)	67																																																																										
CURSOR (left)	104	MEAS (A)	86	RUN	71																																																																										
CURSOR (right)	103	MEAS (B)	83	SPEED (A)	94																																																																										
DIGITS (A)	87	MENU	68	SPEED (B)	91																																																																										
DIGITS (B)	84	MODE (A)	69	SRC (A)	79																																																																										
DISPLAY	72	MODE (B)	66	SRC (B)	76																																																																										
ENTER	82	OUTPUT (A)	88	STORE	78																																																																										
EXIT	75	OUTPUT (B)	96	TRIG	92																																																																										
FILTER (A)	77	RANGE (down)	81	Wheel (press)	97																																																																										
FILTER (B)	74	RANGE (up)	65	Wheel (left)	107																																																																										
LIMIT (A)	93	RECALL	85	Wheel (right)	114																																																																										
<b>Details</b>	See “ <a href="#">Capturing key-press codes</a> ” on <a href="#">page 14-17</a> .																																																																														
<b>Also see</b>	<a href="#">display.sendkey</a> , <a href="#">display.settext</a> , <a href="#">display.getlastkey</a>																																																																														
<b>Example</b>	<p>The following code will hold up script execution and wait for the operator to press a key or the wheel, or rotate the wheel:</p> <pre>key = display.waitkey() print(key)</pre> <p>Output: 8.600000e+01</p> <p>The above output (86) indicates that the <b>MEAS (A)</b> key was pressed.</p>																																																																														

## errorqueue functions and attribute

The functions and attribute in this group are used to read the entries in the error/event queue.

<b>errorqueue.clear</b>	
<b>Function</b>	Clears all entries out of the error/event queue.
<b>Usage</b>	<code>errorqueue.clear()</code>
<b>Remarks</b>	This function removes all entries from the error/event queue.
<b>Details</b>	See <a href="#">Appendix B</a> (error and status messages) and <a href="#">Appendix D</a> (status model).
<b>Also see</b>	<a href="#">errorqueue.count</a> , <a href="#">errorqueue.next</a>

<b>errorqueue.count</b>	
<b>Attribute</b>	The number of entries in the error/event queue.
<b>Usage</b>	<code>count = errorqueue.count</code>
<b>Remarks</b>	This attribute can be read to determine the number of messages in the error/event queue. This is a read-only attribute. Writing to this attribute will generate an error.
<b>Details</b>	See <a href="#">Appendix B</a> (error and status messages) and <a href="#">Appendix D</a> (status model).
<b>Also see</b>	<a href="#">errorqueue.clear</a> , <a href="#">errorqueue.next</a>
<b>Example</b>	<p>Reads number of entries in the error/event queue:</p> <pre>count = errorqueue.count print(count)</pre> <p>Output: 4.000000e+00</p> <p>The above output indicates that there are four entries in the event/error queue.</p>

<b>errorqueue.next</b>	
<b>Function</b>	Reads an entry from the error/event queue.
<b>Usage</b>	<pre>errorcode, message, severity, node = errorqueue.next()</pre> <p><code>errorcode</code> Returns the error code number for the entry.  <code>message</code> Returns the message that describes the entry.  <code>severity</code> Returns the severity level (0, 10, 20, 30 or 40).  <code>node</code> Returns the node number where the error originated.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Entries are stored in a first-in, first-out (FIFO) queue. This function reads the oldest entry and removes it from the queue.</li> <li>• Error codes and messages are listed in <a href="#">Table B-2</a> in <a href="#">Section B</a>.</li> <li>• If there are no entries in the queue, code 0, “Queue Is Empty” is returned.</li> <li>• Returned <code>severity</code> levels include the following: <ul style="list-style-type: none"> <li>0 Informational – Indicates no error: “Queue is Empty”.</li> <li>10 Informational – Indicates an event or a minor error. Examples: “Reading Available” and “Reading Overflow”.</li> <li>20 Recoverable – Indicates possible invalid user input. Operation will continue but action should be taken to correct the error. Examples: “Exponent Too Large” and “Numeric Data Not Allowed”.</li> <li>30 Serious – Indicates a serious error and may require technical assistance. Example: “Saved calibration constants corrupted”.</li> <li>40 Fatal – Indicates that the Model 260x is non-operational and will require service. Contact information for service is provided in <a href="#">Section 1</a>. Examples: “Bad SMU AFPGA image size”, “SMU is unresponsive” and “Communication Timeout with DFPGA”.</li> </ul> </li> <li>• In an expanded system, each TSP-Link enabled instrument is assigned a node number. <code>node</code> returns the node number where the error originated.</li> </ul>
<b>Details</b>	See <a href="#">Appendix B</a> (error and status messages) and <a href="#">Appendix D</a> (status model).
<b>Also see</b>	<a href="#">errorqueue.clear</a> , <a href="#">errorqueue.count</a>
<b>Example</b>	<p>Reads the oldest entry in the error/event queue:</p> <pre>errorcode, message = errorqueue.next() print(errorcode, message)</pre> <p>Output: 0.000000e+00 Queue Is Empty</p> <p>The above output indicates that the queue is empty.</p>



## exit function

This function is used to terminate a script that is presently running.

exit	
<b>Function</b>	Stops execution of a script.
<b>Usage</b>	<code>exit()</code>
<b>Remarks</b>	Aborts script execution. This command will not wait for overlapped commands to complete before terminating script execution. If overlapped commands are required to finish, use the <code>waitcomplete</code> function prior to calling <code>exit</code> .

## format attributes

The format attributes are used to configure the output formats used by the `printnumber` and `printbuffer` functions. These attributes are used to set the data format (ASCII or binary), ASCII precision (number of digits) and binary byte order (normal or swapped).

format.asciiprecision	
<b>Attribute</b>	The precision (number of digits) for all numbers printed with the ASCII format.
<b>Usage</b>	<pre>precision = format.asciiprecision    -- Reads precision. format.asciiprecision = precision    -- Writes precision. precision    Set from 1 to 16.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute selects the precision (number of digits) for data printed with the <code>print</code>, <code>printnumber</code> and <code>printbuffer</code> functions. The precision attribute is only used with the ASCII format. The precision must be a number between 1 and 16.</li> <li>• Note that the precision is the number of significant digits printed. There will always be one digit to the left of the decimal point. Be sure to include this digit when setting the precision.</li> <li>• The default (<code>reset</code>) precision is 6.</li> </ul>
<b>Also see</b>	<a href="#">format.byteorder</a> , <a href="#">format.data</a> , <a href="#">printbuffer</a> , <a href="#">printnumber</a>
<b>Example</b>	<pre>Sets the ASCII precision to 7 digits and prints a number: format.asciiprecision = 7 print(2.5) Output: 2.500000E+00</pre>

<b>format.byteorder</b>											
<b>Attribute</b>	The binary byte order for data printed using the <code>printnumber</code> and <code>printbuffer</code> functions.										
<b>Usage</b>	<pre>order = format.byteorder      -- Reads byte order. format.byteorder = order      -- Writes byte order.</pre> <p>Set <code>order</code> to one of the following values:</p> <table> <tr> <td>0 or <code>format.NORMAL</code></td> <td>Most significant byte first.</td> </tr> <tr> <td>0 or <code>format.BIGENDIAN</code></td> <td>Most significant byte first.</td> </tr> <tr> <td>0 or <code>format.NETWORK</code></td> <td>Most significant byte first.</td> </tr> <tr> <td>1 or <code>format.SWAPPED</code></td> <td>Least significant byte first.</td> </tr> <tr> <td>1 or <code>format.LITTLEENDIAN</code></td> <td>Least significant byte first.</td> </tr> </table>	0 or <code>format.NORMAL</code>	Most significant byte first.	0 or <code>format.BIGENDIAN</code>	Most significant byte first.	0 or <code>format.NETWORK</code>	Most significant byte first.	1 or <code>format.SWAPPED</code>	Least significant byte first.	1 or <code>format.LITTLEENDIAN</code>	Least significant byte first.
0 or <code>format.NORMAL</code>	Most significant byte first.										
0 or <code>format.BIGENDIAN</code>	Most significant byte first.										
0 or <code>format.NETWORK</code>	Most significant byte first.										
1 or <code>format.SWAPPED</code>	Least significant byte first.										
1 or <code>format.LITTLEENDIAN</code>	Least significant byte first.										
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute selects the byte order that data is written when printing data values with the <code>printnumber</code> and the <code>printbuffer</code> functions. The byte order attribute is only used with the <code>SREAL</code>, <code>REAL</code>, <code>REAL32</code>, and <code>REAL64</code> data formats.</li> <li>• <code>NORMAL</code>, <code>BIGENDIAN</code>, and <code>NETWORK</code> select the same byte order. <code>SWAPPED</code> and <code>LITTLEENDIAN</code> select the same byte order. They are alternative identifiers. Selecting which to use is a matter of preference.</li> <li>• Select the <code>SWAPPED</code> or <code>LITTLEENDIAN</code> byte order when sending data to an IBM PC compatible computer.</li> </ul>										
<b>Also see</b>	<a href="#">format.asciiprecision</a> , <a href="#">format.data</a> , <a href="#">printbuffer</a> , <a href="#">printnumber</a>										
<b>Example</b>	<p>Selects the <code>SWAPPED</code> byte order:</p> <pre>format.byteorder = format.SWAPPED</pre>										

<b>format.data</b>											
<b>Attribute</b>	The data format for data printed using the <code>printnumber</code> and <code>printbuffer</code> functions.										
<b>Usage</b>	<pre>fmt = format.data      -- Reads data format. format.data = fmt     -- Writes data format.</pre> <p>Set <code>fmt</code> to one of the following values:</p> <table> <tr> <td>1 or <code>format.ASCII</code></td> <td>ASCII format.</td> </tr> <tr> <td>2 or <code>format.SREAL</code></td> <td>Single precision IEEE-754 binary format.</td> </tr> <tr> <td>2 or <code>format.REAL32</code></td> <td>Single precision IEEE-754 binary format.</td> </tr> <tr> <td>3 or <code>format.REAL</code></td> <td>Double precision IEEE-754 binary format.</td> </tr> <tr> <td>3 or <code>format.REAL64</code></td> <td>Double precision IEEE-754 binary format.</td> </tr> </table>	1 or <code>format.ASCII</code>	ASCII format.	2 or <code>format.SREAL</code>	Single precision IEEE-754 binary format.	2 or <code>format.REAL32</code>	Single precision IEEE-754 binary format.	3 or <code>format.REAL</code>	Double precision IEEE-754 binary format.	3 or <code>format.REAL64</code>	Double precision IEEE-754 binary format.
1 or <code>format.ASCII</code>	ASCII format.										
2 or <code>format.SREAL</code>	Single precision IEEE-754 binary format.										
2 or <code>format.REAL32</code>	Single precision IEEE-754 binary format.										
3 or <code>format.REAL</code>	Double precision IEEE-754 binary format.										
3 or <code>format.REAL64</code>	Double precision IEEE-754 binary format.										
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute selects the data format used to print data values with the <code>printnumber</code> and <code>printbuffer</code> functions.</li> <li>• The precision of the ASCII format can be controlled with the <a href="#">format.asciiprecision</a> attribute. The byte order of SREAL, REAL, REAL32, and REAL64 can be selected with the <a href="#">format.byteorder</a> attribute.</li> <li>• REAL32 and SREAL select the same single precision format. REAL and REAL64 select the same double precision format. They are alternative identifiers. Selecting which to use is a matter of preference.</li> <li>• The IEEE-754 binary formats use 4 bytes each for single precision values and 8 bytes each for double precision values.</li> <li>• When data is written with any of the binary formats, the response message will start with “#0” and end with a new line. When data is written with the ASCII format, elements will be separated with a comma and space.</li> </ul>										
<b>Also see</b>	<a href="#">format.asciiprecision</a> , <a href="#">format.byteorder</a> , <a href="#">printbuffer</a> , <a href="#">printnumber</a>										
<b>Example</b>	<p>Selects the ASCII data format:</p> <pre>format.data = format.ASCII</pre>										

## gpib attribute

The following attribute is used to set the GPIB address.

<b>gpib.address</b>	
<b>Attribute</b>	GPIB address.
<b>Usage</b>	<pre>address = gpib.address      -- Reads address. gpib.address = address      -- Writes address. address          Set from 0 to 30.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A new GPIB address takes effect when the command is processed. If there are response messages in the output queue when this command is processed they must be read at the new address.</li> <li>• The user 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.</li> <li>• The GPIB address is stored in non-volatile memory. The <code>reset</code> function has no effect on the address.</li> </ul>
<b>Details</b>	See “ <a href="#">GPIB operation</a> ” in <a href="#">Section 11</a> .
<b>Example</b>	<p>Sets the GPIB address of the Model 260x to 26 and then reads the address:</p> <pre>gpib.address = 26 address = gpib.address print(address)</pre> <p>Output: 2.600000e+01</p>

## localnode attributes

The attributes in this group are used to set the power line frequency, control (on/off) prompting and control (hide/show) error messages on the display.

<b>localnode.linefreq</b>	
<b>Attribute</b>	Power line frequency.
<b>Usage</b>	<pre>frequency = localnode.linefreq      -- Reads line frequency. localnode.linefreq = frequency      -- Writes line frequency. frequency      Set to 50 or 60.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• To achieve optimum noise rejection when performing measurements at integer NPLC apertures, the line frequency setting must match the frequency (50Hz or 60Hz) of the AC power line.</li> <li>• When used in an expanded system (TSP-Link), <code>localnode.linefreq</code> is sent to the Remote Master node only. Use <code>node[N].linefreq</code> (where N is the node number) to send the command to any node in the system. See <a href="#">Section 9</a> for details on TSP-Link.</li> </ul>
<b>Example</b>	Sets the line frequency to 60Hz: <pre>localnode.linefreq = 60</pre>

<b>localnode.prompts</b>	
<b>Attribute</b>	Prompting mode.
<b>Usage</b>	<pre>prompting = localnode.prompts      -- Reads prompting state. localnode.prompts = prompting      -- Writes prompting state. prompting      Set to 0 to disable or 1 to enable.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute controls prompting. When it is set to 1, prompts are issued after each command message is processed by the instrument. When it is set to 0, prompts are not issued.</li> <li>• The command messages do not generate prompts. The Model 260x generates prompts in response to command messages.</li> <li>• When the prompting mode is enabled, the Model 260x generates prompts in response to command messages. There are three prompts that might be returned: <ul style="list-style-type: none"> <li>• “TSP&gt;” is the standard prompt. This prompt indicates that everything is normal and the command is done processing.</li> <li>• “TSP?” is issued if there are entries in the error queue when the prompt is issued. Like the “TSP&gt;” prompt, it indicates the command is done processing. It does not mean the previous command generated an error, only that there are still errors in the queue when the command was done processing.</li> <li>• “&gt;&gt;&gt;&gt;” is the continuation prompt. This prompt is used when downloading scripts or flash images. When downloading scripts or flash images, many command messages must be sent as a unit. The continuation prompt indicates that the instrument is expecting more messages as part of the current command.</li> </ul> </li> <li>• Test Script Builder (TSB) requires prompts. It sets the prompting mode behind the scenes. If you disable prompting, use of the TSB will hang because it will be waiting for the prompt that lets it know that the command is done executing. DO NOT disable prompting with the use of the TSB.</li> <li>• When used in an expanded system (TSP-Link), <code>localnode.prompt</code> is sent to the Remote Master node only. Use <code>node[N].prompt</code> (where N is the node number) to send the command to any node in the system. See <a href="#">Section 9</a> for details on TSP-Link.</li> </ul>
<b>Also see</b>	<a href="#">localnode.showerrors</a>
<b>Example</b>	Enables prompting: <pre>localnode.prompts = 1</pre>

<b>localnode.showerrors</b>	
<b>Attribute</b>	Automatic display of errors.
<b>Usage</b>	<pre> errormode = localnode.showerrors    -- Reads show errors state. localnode.showerrors = errormode    -- Writes show errors state. errormode    Set to 0 or 1. </pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If this attribute is set to 1, for any errors that are generated, the unit will automatically display the errors stored in the error queue, and then clear the queue. Errors will be processed at the end of executing a command message (just prior to issuing a prompt if prompts are enabled).</li> <li>• If this attribute is set to 0, errors will be left in the error queue and must be explicitly read or cleared.</li> <li>• When used in an expanded system (TSP-Link), <code>localnode.showerrors</code> is sent to the Remote Master node only. Use <code>node [N] .showerrors</code> (where <code>N</code> is the node number) to send the command to any node in the system. See <a href="#">Section 9</a> for details on TSP-Link.</li> </ul>
<b>Details</b>	See “ <a href="#">errorqueue functions and attribute</a> ”
<b>Also see</b>	<a href="#">localnode.prompts</a>
<b>Example</b>	Displays errors: <code>localnode.showerrors = 1</code>

## makegetter functions

These functions are used to set and retrieve a value for an attribute.

<b>makegetter</b>							
<b>Function</b>	Creates a function to set the value of an attribute.						
<b>Usage</b>	<pre> getter = makegetter(table, attributename) </pre> <p> <table> <tr> <td><code>table</code></td> <td>Read-only table where the attribute is located.</td> </tr> <tr> <td><code>attributename</code></td> <td>The string name of the attribute.</td> </tr> <tr> <td><code>getter</code></td> <td>Function that returns the value of the given attribute.</td> </tr> </table> </p>	<code>table</code>	Read-only table where the attribute is located.	<code>attributename</code>	The string name of the attribute.	<code>getter</code>	Function that returns the value of the given attribute.
<code>table</code>	Read-only table where the attribute is located.						
<code>attributename</code>	The string name of the attribute.						
<code>getter</code>	Function that returns the value of the given attribute.						
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function creates a function that when called returns the value of the attribute. This function is useful for aliasing attributes to improve execution speed. Calling the <code>getter</code> function will execute faster than accessing the attribute directly.</li> <li>• Creating a <code>getter</code> function is only useful if it is going to be called several times. Otherwise the overhead of creating the <code>getter</code> function outweighs the overhead of accessing the attribute directly.</li> </ul>						

<b>Example</b>	<p>Creates a getter function called <code>getlevel</code>:</p> <pre>getlevel = makegetter(smua.source, "levelv") ... v = getlevel()</pre> <p>When <code>getlevel</code> is called, it returns the value of <code>smua.source.levelv</code>.</p>
----------------	---

<b>makesetter</b>							
<b>Function</b>	Creates a function to set the value of an attribute.						
<b>Usage</b>	<pre>setter = makesetter(table, attributename)</pre> <table style="width: 100%; border: none;"> <tr> <td style="padding-right: 20px;"><code>table</code></td> <td>Read-only table where the attribute is located.</td> </tr> <tr> <td><code>attributename</code></td> <td>The string name of the attribute.</td> </tr> <tr> <td><code>setter</code></td> <td>Function that sets the value of the given attribute.</td> </tr> </table>	<code>table</code>	Read-only table where the attribute is located.	<code>attributename</code>	The string name of the attribute.	<code>setter</code>	Function that sets the value of the given attribute.
<code>table</code>	Read-only table where the attribute is located.						
<code>attributename</code>	The string name of the attribute.						
<code>setter</code>	Function that sets the value of the given attribute.						
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function creates a function that when called sets the value of the attribute. This function is useful for aliasing attributes to improve execution speed. Calling the <code>setter</code> function will execute faster than accessing the attribute directly.</li> <li>• Creating a <code>setter</code> function is only useful if it is going to be called several times. Otherwise the overhead of creating the <code>setter</code> function outweighs the overhead of accessing the attribute directly.</li> </ul>						
<b>Example</b>	<p>Creates a <code>setter</code> function called <code>setlevel</code>:</p> <pre>setlevel = makesetter(smua.source, "levelv") for v = 1, 10 do     setlevel(v) end</pre> <p>Using <code>setlevel</code> in the loop sets the value of <code>smua.source.levelv</code>, thereby performing a source sweep.</p>						



## opc function

This function sets the OPC bit in the status register when all overlapped commands are completed.

opc	
<b>Function</b>	Sets the Operation Complete status bit when all overlapped commands are completed.
<b>Usage</b>	<code>opc ( )</code>
<b>Remarks</b>	<ul style="list-style-type: none"><li>• This function will cause the Operation Complete bit in the Standard Event Status Register to be set when all previously started local overlapped commands are complete. Note that each node will independently set their Operation Complete bits in their own status models.</li><li>• Any nodes not actively performing overlapped commands will set their bits immediately. All remaining nodes will set their own bits as they complete their own overlapped commands.</li></ul>
<b>Details</b>	See “ <a href="#">Status Model</a> ” in <a href="#">Section D</a> .
<b>Also see</b>	<a href="#">waitcomplete</a>

# printbuffer and printnumber functions

These functions are used to print data and numbers.

<b>printbuffer</b>	
<b>Function</b>	Prints data from reading buffer sub-tables.
<b>Usage</b>	<p>There are multiple ways to use this function, depending on how many sub-tables are used:</p> <pre>printbuffer(start_index, end_index, st_1) printbuffer(start_index, end_index, st_1, st_2) printbuffer(start_index, end_index, st_1, st_2, ..., st_n)</pre> <p>start_index                      Starting index of values to print.  end_index                            Ending index of values to print.  st_1, st_2, ... st_n              Sub-tables from which to print values.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Correct usage when there are no outstanding overlapped commands to acquire data: <pre>1 &lt;= start_index &lt;= end_index &lt;= n</pre> <p>Where <i>n</i> refers to the index of the last entry in the tables to be printed.</p> <p>If <i>end_index</i> &lt; <i>start_index</i> or <i>n</i> &lt; <i>start_index</i>, no data will be printed. If <i>start_index</i> &lt; 1, 1 will be used as the first index. If <i>n</i> &lt; <i>end_index</i>, <i>n</i> will be used as the last index.</p> </li> <li>• When any of the given reading buffers are being used in overlapped commands that have not yet completed at least to the desired index, this function will return data as it becomes available.</li> <li>• When there are outstanding overlapped commands to acquire data, <i>n</i> refers to the index that the last entry in the table will have after all the measurements have completed.</li> <li>• This function prints values from reading buffers. If a specific sub-table is not specified (e.g., “rb1” rather than “rb1.statuses”), the default “readings” sub-table will be used.</li> <li>• At least one sub-table must be specified. There is an upper limit that is dictated by the output format and the maximum output message length. Values will be interleaved in one message. Care must be taken not to exceed the maximum output message length.</li> <li>• All the data will be put in one response message. The response message will be formatted as dictated by <code>format.data</code> and other associated attributes.</li> </ul>
<b>Also see</b>	<a href="#">format.data</a> , <a href="#">printnumber</a>

<b>Example</b>	<p>This example prints all timestamps and readings in one buffer and all readings from another buffer, where n is 4:</p> <pre>format.data = format.ASCII printbuffer(1, rb1.n, rb1.timestamps, rb1, rb2)</pre> <p>Example of returned data (timestamps, rb1.readings, rb2.readings):</p> <pre>1.02345E-04, 8.76542E-04, 5.29372E-01, 1.02445E-04, 8.66543E-04, 5.24242E-01, 1.02545E-04, 8.56547E-04, 5.19756E-01, 1.02645E-04, 8.44546E-04, 5.14346E-01</pre>
----------------	--

<b>printnumber</b>	
<b>Function</b>	Prints numbers using the format selected for printing reading buffers.
<b>Usage</b>	<p>There are multiple ways to use this function, depending on how many numbers are to be printed:</p> <pre>printnumber(v1) printnumber(v1 ,v2) printnumber(v1 ,v2, ..., vn) v1, v2, ..., vn    Numbers to print.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will print the given numbers using the data format specified by <code>format.data</code> and other associated attributes.</li> <li>• At least one number must be given. There is an upper limit that is dictated by the output format and the maximum output message length. All values will be written in a single message. Care must be taken not to exceed the maximum output message length.</li> </ul>
<b>Also see</b>	<a href="#">printbuffer</a> , <a href="#">format.data</a>
<b>Example</b>	<p>Prints three measurements that were previously performed:</p> <pre>format.data = format.ASCII printnumber(i, v, t)</pre> <p>Example of returned data (i, v, t):</p> <pre>1.02345E-04, 8.76542E-02, 5.29372E-01</pre>

## reset function

This function is used to return all logical instruments to the default settings.

reset	
<b>Function</b>	Resets the logical instruments to the default settings.
<b>Usage</b>	<code>reset()</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function resets all logical instruments in the system. This function is equivalent to iterating over all the logical instruments in the system and calling the reset method of each.</li> <li>• Default settings are listed in <a href="#">Table 1-3</a>.</li> </ul>
<b>Details</b>	See “ <a href="#">Default settings</a> ” in <a href="#">Section 1</a> .

## serial functions and attributes

The functions and attributes in this group are used to configure the RS-232 Interface:

serial.baud	
<b>Attribute</b>	Baud rate for the RS-232 port.
<b>Usage</b>	<pre> baud = serial.baud      -- Reads baud rate. serial.baud = baud     -- Writes baud rate. baud      Set to 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 or 115200. </pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A new baud rate setting takes effect when the command to change it is processed.</li> <li>• The user should allow ample time for the command to be processed before attempting to communicate with the instrument again. It is recommended that the baud rate be set from the GPIB interface or from the front panel.</li> <li>• The baud rate is stored in non-volatile memory. The <code>reset</code> function has no effect on the baud rate.</li> </ul>
<b>Details</b>	See “ <a href="#">RS-232 interface operation</a> ” in <a href="#">Section 11</a> .
<b>Also see</b>	<a href="#">serial.databits</a> , <a href="#">serial.flowcontrol</a> , <a href="#">serial.parity</a>
<b>Example</b>	Sets the baud rate to 1200: <code>serial.baud = 1200</code>

<b>serial.databits</b>	
<b>Attribute</b>	Character width (data bits) for the RS-232 port.
<b>Usage</b>	<pre>bits = serial.databits    -- Reads data width. serial.databits = bits    -- Writes data width. bits      Set to 7 or 8.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A new data width setting takes effect when the command to change it is processed.</li> <li>• The user should allow ample time for the command to be processed before attempting to communicate with the instrument again. It is recommended that the data width be set from the GPIB interface or from the front panel.</li> <li>• The data bits value is stored in non-volatile memory. The <code>reset</code> function has no effect on data bits.</li> </ul>
<b>Details</b>	See “ <a href="#">RS-232 interface operation</a> ” in <a href="#">Section 11</a> .
<b>Also see</b>	<a href="#">serial.baud</a> , <a href="#">serial.flowcontrol</a> , <a href="#">serial.parity</a>
<b>Example</b>	Sets data width to 8: <pre>serial.databits = 8</pre>

<b>serial.flowcontrol</b>	
<b>Attribute</b>	Flow control for the RS-232 port.
<b>Usage</b>	<pre>flow = serial.flowcontrol    -- Reads flow control. serial.flowcontrol = flow    -- Writes flow control.</pre> <p>Set <code>flow</code> to one of the following values:</p> <pre>"none"      or serial.FLOW_NONE          Selects no flow control. "hardware" or serial.FLOW_HARDWARE      Selects hardware flow control.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A new flow control setting takes effect when the command to change it is processed.</li> <li>• The user should allow ample time for the command to be processed before attempting to communicate with the instrument again. It is recommended that the flow control be set from the GPIB interface or from the front panel.</li> <li>• The flow control value is stored in non-volatile memory. The <code>reset</code> function has no effect on flow control.</li> </ul>
<b>Details</b>	See “ <a href="#">RS-232 interface operation</a> ” in <a href="#">Section 11</a> .
<b>Also see</b>	<a href="#">serial.baud</a> , <a href="#">serial.databits</a> , <a href="#">serial.parity</a>
<b>Example</b>	Sets flow control to none: <pre>serial.flowcontrol = serial.FLOW_NONE</pre>

<b>serial.parity</b>	
<b>Attribute</b>	Parity for the RS-232 port.
<b>Usage</b>	<pre>parity = serial.parity    -- Reads parity. serial.parity = parity    -- Writes parity.</pre> <p>Set <code>parity</code> to one of the following values:</p> <pre>"none" or serial.PARITY_NONE    Selects no parity. "even" or serial.PARITY_EVEN    Selects even parity. "odd" or serial.PARITY_ODD     Selects odd parity.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A new parity setting takes effect when the command to change it is processed.</li> <li>• The user should allow ample time for the command to be processed before attempting to communicate with the instrument again. It is recommended that the parity be set from the GPIB interface or from the front panel.</li> <li>• The parity setting is stored in non-volatile memory. The <code>reset</code> function has no effect on parity.</li> </ul>
<b>Details</b>	See “ <a href="#">RS-232 interface operation</a> ” in <a href="#">Section 11</a> .
<b>Also see</b>	<a href="#">serial.baud</a> , <a href="#">serial.databits</a> , <a href="#">serial.flowcontrol</a>
<b>Example</b>	Sets parity to none: <pre>serial.parity = serial.PARITY_NONE</pre>

<b>serial.read</b>	
<b>Function</b>	Reads data from the serial port.
<b>Usage</b>	<pre>data = serial.read(maxchars)</pre> <p><code>maxchars</code>            Sets the maximum number of characters to read.  <code>data</code>                 Returns a string consisting of all data read from the serial port.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will read available characters from the serial port. It will not wait for new characters to arrive. As long as <code>maxchars</code> is a relatively small number (less than several hundred characters), all characters received by the serial port prior to the call will be returned. This might be less than <code>maxchars</code>. If too many characters are received in between calls to this function, the RS-232 buffers will overflow and some characters may be lost.</li> <li>• This function can be called as many times as necessary to receive the required number of characters. For optimal performance, it is suggested that a small delay be used between repeat calls to this function.</li> <li>• The data returned is the raw data stream read from the port. Control characters, terminator characters, etc. will not be processed nor will the data stream be altered in any way.</li> <li>• This function cannot be used if the serial port is enabled as a command interface. A settings conflict error will be generated if the serial port is enabled as a command interface when this function is called.</li> </ul>

<b>Also see</b>	<a href="#">serial.write</a>
<b>Example</b>	<p>Reads data from the serial port:</p> <pre>data = serial.read(200) print(data)</pre> <p>Output: John Doe</p> <p>The above output indicates that the string “John Doe” was read from the serial port.</p>

<b>serial.write</b>	
<b>Function</b>	Writes data to the serial port.
<b>Usage</b>	<pre>serial.write(data) data</pre> <p>Specify the data string to write.</p>
<b>Remarks</b>	This function will write the given string to the serial port where it can be read by equipment (e.g., component handler) connected to the other end of the serial port. No terminator characters are added to the data. The data will be written exactly as is.
<b>Also see</b>	<a href="#">serial.read</a>
<b>Example</b>	<p>Writes data string “1 2 3 4” to the serial port:</p> <pre>serial.write("1 2 3 4")</pre>

## setup functions and attribute

The functions and attribute in this group are used to save/recall setups and to set the power-on setup.

<b>setup.poweron</b>	
<b>Attribute</b>	The saved setup to recall when the unit is turned on.
<b>Usage</b>	<pre>n = setup.poweron</pre> <p>-- Reads the power-on setup.</p> <pre>setup.poweron = n</pre> <p>-- Writes the power-on setup.</p> <p>n Setup number to recall on power up (0 to 5).</p>
<b>Remarks</b>	For an n setting of 0, the unit powers up to the factory default (reset) setup. For an n setting of 1 to 5, the unit powers up to a user saved setup.
<b>Details</b>	See “ <a href="#">Remote operation setups</a> ” in <a href="#">Section 1</a> .
<b>Example</b>	<p>Sets unit to power on to the factory defaults:</p> <pre>setup.poweron = 0</pre>

<b>setup.recall</b>	
<b>Function</b>	Recalls settings from a saved setup.
<b>Usage</b>	<pre>setup.recall (n) n      Setup number to recall (0 to 5).</pre>
<b>Remarks</b>	For an $n$ setting of 0, the unit recalls the factory default (reset) setup. For an $n$ setting of 1 to 5, the unit recalls a user saved setup.
<b>Details</b>	See “ <a href="#">Remote operation setups</a> ” in <a href="#">Section 1</a> .
<b>Example</b>	Recalls the user-setup at location 2: <pre>setup.recall (2)</pre>

<b>setup.save</b>	
<b>Function</b>	Saves the present setup as a user-setup.
<b>Usage</b>	<pre>setup.save (n) n      Setup number to save (1 to 5).</pre>
<b>Remarks</b>	Numbers 1 through 5 are used to designate user-setup locations. When you save at one of these locations, the previous setup at that location is overwritten.
<b>Details</b>	See “ <a href="#">Remote operation setups</a> ” in <a href="#">Section 1</a> .
<b>Example</b>	Saves the present setup at location 5: <pre>setup.save (5)</pre>

## smuX functions and attributes

The functions and attributes in this group are used to control basic source-measure operations of the SMUs and perform calibration.

<b>smuX.cal.date</b>	
	X = SMU channel (a or b)
<b>Attribute</b>	Calibration date for the active calibration set.
<b>Usage</b>	<pre>caldate = smuX.cal.date      -- Reads calibration date. smuX.cal.date = caldate     -- Writes calibration date.  Set caldate to the following value: os.time({year=yr, month=mo, day=da})  where: yr  2005 to 2037          mo  1 to 12          da  1 to 31</pre>



<b>Remarks</b>	<ul style="list-style-type: none"> <li>This attribute stores the calibration date associated with the active calibration set. The calibration date can be read at any time but can only be assigned a new value when calibration has been enabled with the <code>smuX.cal.unlock</code> function.</li> <li>This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect the date stored with that set.</li> <li>Hour and minute can also be included in <code>caldate</code> as follows:  <pre>os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})</pre>           Seconds can be included, but will essentially be ignored due to the precision of the internal date storage format.</li> <li>The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.</li> </ul>
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.cal.due</a> , <a href="#">smuX.cal.restore</a> , <a href="#">smuX.cal.save</a> , <a href="#">smuX.makebuffer</a>
<b>Example</b>	Sets calibration date for SMU A (July 1, 2005): <pre>smua.cal.date = os.time({year=2005, month=7, day=1})</pre>

<b>smuX.cal.due</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Calibration due date for the next calibration.
<b>Usage</b>	<pre>caldue = smuX.cal.due      -- Reads calibration due date. smuX.cal.due = caldue     -- Writes calibration due date.</pre> <p>Set <code>caldue</code> to the following value:  <pre>os.time({year=yr, month=mo, day=da})</pre>           where: yr 2005 to 2037                  mo 1 to 12                  da 1 to 31</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>This attribute stores the calibration due date associated with the active calibration set. The calibration due date can be read at any time but can only be assigned a new value when calibration has been enabled with the <code>smuX.cal.unlock</code> function.</li> <li>This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect the due date stored with that set.</li> <li>Hour and minute can also be included in <code>caldate</code> as follows:  <pre>os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})</pre>           Seconds can be included, but will essentially be ignored due to the precision of the internal date storage format.</li> <li>The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.</li> </ul>
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.cal.date</a> , <a href="#">smuX.cal.restore</a> , <a href="#">smuX.cal.state</a> , <a href="#">smuX.makebuffer</a>
<b>Example</b>	Sets calibration due date for SMU A (July 1, 2006): <pre>smua.cal.due = os.time({year=2005, month=7, day=1})</pre>

<b>smuX.cal.lock</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Function</b>	Disables commands that change calibration settings.
<b>Usage</b>	<code>smuX.cal.lock()</code>
<b>Remarks</b>	This function will disable the calibration functions that can change the calibration settings. It is an error to call this function while the calibration state is <code>smuX.CALSTATE_CALIBRATING</code> . The calibration constants must be written to non-volatile memory, or a previous calibration set must be restored prior to locking calibration.
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.cal.restore</a> , <a href="#">smuX.cal.save</a> , <a href="#">smuX.cal.state</a> , <a href="#">smuX.makebuffer</a>
<b>Example</b>	Disable calibration functions for SMU A: <code>smua.cal.lock()</code>

<b>smuX.cal.password</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Password to enable calibration.
<b>Usage</b>	<code>smuX.cal.password = newpassword</code> <code>newpassword</code> The new password (string).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• A new password can only be assigned when calibration has been unlocked.</li> <li>• The calibration password is write-only and cannot be read.</li> </ul>
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.makebuffer</a>
<b>Example</b>	Assign a new calibration password for SMU A: <code>smua.cal.password = "LetMeIn"</code>

<b>smuX.cal.polarity</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Control which calibration constants are used for all subsequent measurements.
<b>Usage</b>	<pre>calpolarity = smuX.cal.polarity  -- Reads cal polarity. smuX.cal.polarity = calpolarity  -- Writes cal polarity.  Set calpolarity to one of the following values: 0 or smuX.CAL_AUTO           Automatic polarity detection. 1 or smuX.CAL_POSITIVE       Measure with positive polarity calibration constants. 2 or smuX.CAL_NEGATIVE       Measure with negative polarity calibration constants.</pre>

<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute controls which polarity calibration constants are used to make all subsequent measurements. This attribute does not affect the <code>smuX.measure.calibrateY</code> or the <code>smuX.source.calibrateY</code> function. The polarity for those commands are dictated by the range parameter given to the command.</li> <li>• The measurement calibration commands require the measurements provided to have been made using the polarity being calibrated. When the calibration points are sufficiently far away from zero the desired polarity constants are inherently used when making those measurements. When measuring near zero, it is possible for the measurement to be made using the calibration constants from either polarity without knowing which was used. Setting this attribute to positive or negative forces measurements to be made using the calibration constants for a given polarity rather than basing the choice on the raw measurement data.</li> <li>• This attribute can only be set to positive or negative when calibration is unlocked. This attribute will automatically be set back to <code>CAL_AUTO</code> when calibration is locked.</li> </ul>
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.makebuffer</a> , <a href="#">smuX.measure.calibrateY</a> , <a href="#">smuX.source.calibrateY</a>
<b>Example</b>	Selects positive calibration constants for all subsequent measurements: <code>smua.cal.polarity = smua.CAL_POSITIVE</code>

<b>smuX.cal.restore</b> <span style="float: right;">X = SMU channel (a or b)</span>									
<b>Function</b>	Loads a stored set of calibration constants.								
<b>Usage</b>	<p>There are two ways to use this function:</p> <pre>smuX.cal.restore() smuX.cal.restore(calset)</pre> <p><code>calset</code>            Calibration set to be loaded.</p> <p>Set <code>calset</code> to one of the following values:</p> <table> <tr> <td><code>smuX.CALSET_NOMINAL</code></td> <td>A set of calibration constants that are uncalibrated but set to nominal values to allow rudimentary functioning of the instrument.</td> </tr> <tr> <td><code>smuX.CALSET_FACTORY</code></td> <td>The calibration constants when the instrument left the factory.</td> </tr> <tr> <td><code>smuX.CALSET_DEFAULT</code></td> <td>The normal calibration set.</td> </tr> <tr> <td><code>smuX.CALSET_PREVIOUS</code></td> <td>The calibration set that was used before the last default set was overwritten.</td> </tr> </table> <p>If <code>calset</code> is not specified, <code>smuX.CALSET_DEFAULT</code> will be used.</p>	<code>smuX.CALSET_NOMINAL</code>	A set of calibration constants that are uncalibrated but set to nominal values to allow rudimentary functioning of the instrument.	<code>smuX.CALSET_FACTORY</code>	The calibration constants when the instrument left the factory.	<code>smuX.CALSET_DEFAULT</code>	The normal calibration set.	<code>smuX.CALSET_PREVIOUS</code>	The calibration set that was used before the last default set was overwritten.
<code>smuX.CALSET_NOMINAL</code>	A set of calibration constants that are uncalibrated but set to nominal values to allow rudimentary functioning of the instrument.								
<code>smuX.CALSET_FACTORY</code>	The calibration constants when the instrument left the factory.								
<code>smuX.CALSET_DEFAULT</code>	The normal calibration set.								
<code>smuX.CALSET_PREVIOUS</code>	The calibration set that was used before the last default set was overwritten.								
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will overwrite the current set of calibration constants from non-volatile memory.</li> <li>• This function will be disabled until a successful call to <code>smuX.cal.unlock</code> is made.</li> </ul>								
<b>Details</b>	See <a href="#">Section 17</a> (calibration).								

<b>Also see</b>	<a href="#">smuX.makebuffer</a>
<b>Example</b>	Restores factory calibration for SMU A: <code>smua.cal.restore(smua.CALSET_FACTORY)</code>

<b>smuX.cal.save</b>	
X = SMU channel (a or b)	
<b>Function</b>	Stores the calibration constants in non-volatile memory.
<b>Usage</b>	<code>smuX.cal.save()</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>This function will store the current set of calibration constants in non-volatile memory. The previous calibration constants (from the default calibration set) will be copied to the previous calibration set (<code>CALSET_PREVIOUS</code>) prior to overwriting the default calibration set.</li> <li>This function will be disabled until a successful call to <code>smuX.cal.unlock</code> is made. If any of the calibration constants have been changed, this function will be disabled unless both the calibration date and the calibration due date have been assigned new values.</li> </ul>
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.cal.date</a> , <a href="#">smuX.cal.due</a> , <a href="#">smuX.cal.restore</a> , <a href="#">smuX.makebuffer</a>
<b>Example</b>	Stores calibration constants for SMU A in non-volatile memory: <code>smua.cal.save()</code>

<b>smuX.cal.state</b>										
X = SMU channel (a or b)										
<b>Attribute</b>	Calibration state.									
<b>Usage</b>	<code>calstate = smuX.cal.state</code>									
<b>Remarks</b>	<p>When reading this read-only attribute, <code>calstate</code> returns one of the following values:</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 15%; vertical-align: top;">0</td> <td style="width: 55%; vertical-align: top;"><code>smuX.CALSTATE_LOCKED</code></td> <td style="width: 30%; vertical-align: top;">Calibration is locked.</td> </tr> <tr> <td style="vertical-align: top;">1</td> <td style="vertical-align: top;"><code>smuX.CALSTATE_CALIBRATING</code></td> <td style="vertical-align: top;">The calibration constants or dates have been changed but not yet saved to non-volatile memory.</td> </tr> <tr> <td style="vertical-align: top;">2</td> <td style="vertical-align: top;"><code>smuX.CALSTATE_UNLOCKED</code></td> <td style="vertical-align: top;">Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore.</td> </tr> </table>	0	<code>smuX.CALSTATE_LOCKED</code>	Calibration is locked.	1	<code>smuX.CALSTATE_CALIBRATING</code>	The calibration constants or dates have been changed but not yet saved to non-volatile memory.	2	<code>smuX.CALSTATE_UNLOCKED</code>	Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore.
0	<code>smuX.CALSTATE_LOCKED</code>	Calibration is locked.								
1	<code>smuX.CALSTATE_CALIBRATING</code>	The calibration constants or dates have been changed but not yet saved to non-volatile memory.								
2	<code>smuX.CALSTATE_UNLOCKED</code>	Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore.								
<b>Details</b>	See <a href="#">Section 17</a> (calibration).									
<b>Also see</b>	<a href="#">smuX.cal.due</a> , <a href="#">smuX.cal.restore</a> , <a href="#">smuX.cal.save</a> , <a href="#">smuX.makebuffer</a>									

<b>Example</b>	<p>Reads calibration state for SMU A:</p> <pre>calstate = smua.cal.state print(calstate)</pre> <p>Output: 0.000000e+00</p> <p>The above output indicates that calibration is locked.</p>
----------------	--

<b>smuX.cal.unlock</b>		X = SMU channel (a or b)
<b>Function</b>	Enables the commands that change calibration settings.	
<b>Usage</b>	<pre>smuX.cal.unlock(password) password</pre> <p>Calibration password.</p>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function enables the calibration functions to change the calibration settings.</li> <li>• The password when the unit is shipped from the factory is "KI002602".</li> </ul>	
<b>Details</b>	See <a href="#">Section 17</a> (calibration).	
<b>Also see</b>	<a href="#">smuX.cal.password</a>	
<b>Example</b>	<p>Unlocks calibration for SMU A:</p> <pre>smua.cal.unlock("KI002602")</pre>	

<b>smuX.makebuffer</b>		X = SMU channel (a or b)
<b>Function</b>	Creates a RAM buffer.	
<b>Usage</b>	<pre>mybuffer = smuX.makebuffer(bufferize) bufferize</pre> <p>Number of readings that can be stored.</p>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• RAM reading buffers can be allocated dynamically. These are created and allocated with the <code>smuX.makebuffer(buffer)</code> function, where <code>bufferize</code> is the number of readings the buffer can store.</li> <li>• Dynamically allocated reading buffers can be used interchangeably with the <code>smuX.nvbufferY</code> buffers.</li> <li>• A RAM buffer can be deleted using <code>nil</code>. The following command deletes <code>mybuffer</code>: <code>mybuffer = nil</code></li> </ul>	
<b>Details</b>	See <a href="#">"Reading buffers"</a> in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.nvbufferY</a>	
<b>Example</b>	<p>Creates a 200 reading RAM buffer named "mybuffer2" for SMUA:</p> <pre>mybuffer2 = smua.makebuffer(200)</pre>	

<b>smuX.measure.autorangeY</b>		X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
<b>Attribute</b>	Measure auto range setting.	

<b>Usage</b>	<pre>autorange = smuX.measure.autorangeY    -- Reads measure auto range. smuX.measure.autorangeY = autorange    -- Writes measure auto range.</pre> <p>Set <code>autorange</code> to one of the following values:</p> <pre>0 or smuX.AUTORANGE_OFF    Disables measure auto range. 1 or smuX.AUTORANGE_ON    Enables measure auto range.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute indicates the measurement auto range state. Its value will be <code>smuX.AUTORANGE_OFF</code> when the SMU measure circuit is on a fixed range and <code>smuX.AUTORANGE_ON</code> when it is in auto range mode.</li> <li>• Setting this attribute to <code>smuX.AUTORANGE_OFF</code> puts the SMU on a fixed range. The fixed range used will be the range the SMU measure circuit was currently using.</li> <li>• Setting this attribute to <code>smuX.AUTORANGE_ON</code> puts the SMU measure circuit into auto range mode. It will remain on its present measure range until the next measurement is requested.</li> </ul>
<b>Details</b>	See “Range” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.measure.rangeY</a>
<b>Example</b>	Enables voltage measure autoranging for SMU A: <code>smua.measure.autorangev = smua.AUTORANGE_ON</code>

<b>smuX.measure.autozero</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Behavior of the SMU's A/D internal reference measurements (autozero).
<b>Usage</b>	<pre>azmode = smuX.measure.autozero    -- Reads autozero. smuX.measure.autozero = azmode    -- Writes autozero.</pre> <p>Set <code>azmode</code> to be one of the following values:</p> <pre>0 or smuX.AUTOZERO_OFF    Autozero disabled. 1 or smuX.AUTOZERO_ONCE    Performs autozero once, then disables autozero. 2 or smuX.AUTOZERO_AUTO    Automatic checking of reference and zero measurements. An autozero is performed when needed.</pre>

<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The Model 260x uses a ratio metric A/D conversion technique. To ensure accuracy of readings, the instrument must periodically obtain fresh measurements of its internal ground and voltage reference. The time interval between needing to update these reference measurements is determined by the integration aperture being used for measurements. Separate reference and zero measurements are used for each aperture.</li> <li>• By default, the instrument automatically checks these reference measurements whenever a signal measurement is made. If the reference measurements have expired when a signal measurement is made, the instrument will automatically take two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes longer than normal.</li> </ul> <p>This extra time can cause problems in sweeps and other test sequences in which measurement timing is critical. To avoid the extra time for the reference measurements in these situations, the <code>smuX.measure.autozero</code> attribute can be used to disable the automatic reference measurements. Keep in mind that with automatic reference measurements disabled, the instrument may gradually drift out of specification.</p> <p>To minimize the drift, a reference and zero measurement should be made just prior to the critical test sequence. The <code>smuX.AUTOZERO_ONCE</code> setting can be used to force a refresh of the reference and zero measurements used for the current aperture setting.</p> <ul style="list-style-type: none"> <li>• Autozero reference measurements for the last 5 used NPLC settings are stored in a reference cache. If an NPLC setting is selected and an entry for it is not in the cache, the oldest (least recently used) entry will be discarded to make room for the new entry.</li> </ul>
<b>Example</b>	Perform autozero once for SMU A: <pre>smua.measure.autozero = smua.AUTOZERO_ONCE</pre>

<b>smuX.measure.calibrateY</b>	
X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current	
<b>Function</b>	Generates and activates new measurement calibration constants.





<b>smuX.measure.filter.count</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Number of measured readings to yield one filtered measurement.
<b>Usage</b>	<pre>count = smuX.measure.filter.count    -- Reads filter count. smuX.measure.filter.count = count    -- Writes filter count.  count    Set filter count from 1 to 100.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute is the number of measurements that will be performed to yield one filtered measurement.</li> <li>• The <code>reset</code> function sets the filter count to 1.</li> </ul>
<b>Details</b>	See “Filters” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.measure.filter.enable</a> , <a href="#">smuX.measure.filter.type</a>
<b>Example</b>	Sets filter count for SMU A: <pre>smua.measure.filter.count = 10</pre>

<b>smuX.measure.filter.enable</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Enables/disables filtered measurements.
<b>Usage</b>	<pre>filter = smuX.measure.filter.enable  -- Reads on/off state of the filter. smuX.measure.filter.enable = filter  -- Writes on/off state of the filter.  Set filter to one of the following values: 0 or smuX.FILTER_OFF    Disables the filter. 1 or smuX.FILTER_ON    Enables the filter.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute enables or disables the filter.</li> <li>• The <code>reset</code> function disables the filter.</li> </ul>
<b>Details</b>	See “Filters” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.measure.filter.count</a> , <a href="#">smuX.measure.filter.type</a>
<b>Example</b>	Enable the filter for SMU A: <pre>smua.measure.filter.enable = smua.FILTER_ON</pre>

<b>smuX.measure.filter.type</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Type of filter for measurements.
<b>Usage</b>	<pre>type = smuX.measure.filter.type      -- Reads filter type. smuX.measure.filter.type = type      -- Writes filter type.  Set type to one of the following values: 0 or smuX.FILTER_MOVING_AVG          Selects the moving average filter. 1 or smuX.FILTER_REPEAT_AVG         Selects the repeat filter.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• There are two averaging filter types to choose from: Repeating and moving. For the repeating filter (which is the power-on default), the stack (filter count) is filled, and the conversions are averaged to yield a reading. The stack is then cleared, and the process starts over. The moving average filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the measurement conversions are averaged, yielding a reading. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The stack is re-averaged, yielding a new reading.</li> <li>• The <code>reset</code> function selects the repeat filter.</li> </ul>
<b>Details</b>	See “Filters” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.measure.filter.count</a> , <a href="#">smuX.measure.filter.enable</a>
<b>Example</b>	Selects the moving average filter for SMU A: <pre>smua.measure.filter.type = smua.FILTER_MOVING_AVG</pre>

<b>smuX.measure.interval</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Interval between multiple measurements.
<b>Usage</b>	<pre>interval = smuX.measure.interval    -- Reads measure interval. smuX.measure.interval = interval    -- Writes measure interval.  interval      Set interval (in seconds) from 0 to 1.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute sets the time interval between groups of measurements when <code>smua.measure.count</code> is set to a value greater than 1. The SMU will do its best to start the measurement of each group when scheduled.</li> <li>• If filtered measurements are being made, this interval is from the start of the first measurement for the filtered reading to the first measurement for a subsequent filtered reading. Extra measurements made to satisfy a filtered reading are not paced by this interval.</li> <li>• If the SMU cannot keep up with the interval setting, measurements will be made as fast as possible.</li> <li>• The <code>reset</code> function sets the measure interval to 0.</li> </ul>
<b>Details</b>	See “Triggering” in <a href="#">Section 4</a> .
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a>
<b>Example</b>	Sets measure interval for SMU A: <pre>smua.measure.interval = 0.5</pre>

<b>smuX.measure.lowrangeY</b>	
X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current	
<b>Attribute</b>	Lowest measure range that will be used during autoranging.
<b>Usage</b>	<pre>rangeval = smuX.measure.lowrangeY    -- Reads low range. smuX.measure.lowrangeY = rangeval    -- Writes low range. rangeval    Set to the lowest voltage or current measure range.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute is used with auto-ranging to put a lower bound on the range used. Lower ranges generally require greater settling times. By setting a low range value, measurements might be able to be made with less settling time.</li> <li>• If the instrument is set to auto range and it is on a range lower than the one specified, the range will be changed to the range specified.</li> </ul>
<b>Details</b>	See “ <a href="#">Range</a> ” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.measure.autorangeY</a>
<b>Example</b>	Sets volts lowrange for SMU A to 1V: <code>smua.measure.lowrangev = 1</code>

<b>smuX.measure.nplc</b>	
X = SMU channel (a or b)	
<b>Attribute</b>	Integration aperture for measurements.
<b>Usage</b>	<pre>nplc = smuX.measure.nplc    -- Reads nplc. smuX.measure.nplc = nplc    -- Writes nplc. nplc    Set from 0.001 to 25.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The integration aperture is based on the number of power line cycles (NPLC), where 1PLC for 60Hz is 16.67msec (1/60) and 1 PLC for 50Hz is 20msec (1/50).</li> <li>• The reset function sets the aperture to 1.0.</li> </ul>
<b>Details</b>	See “ <a href="#">Speed</a> ” in <a href="#">Section 6</a> .
<b>Example</b>	Sets integration time for SMU A (0.5/60 seconds): <code>smua.measure.nplc = 0.5</code>

<b>smuX.measure.overlappedY</b> <b>smuX.measure.overlappediv</b>		X = SMU channel (a or b) Y = SMU measure function (v, i, r or p)    where: v = voltage, i = current, r = resistance, p = power
<b>Function</b>	Starts an asynchronous (background) measurement.	
<b>Usage</b>	There are two ways to use this function: <code>smuX.measure.overlappedY(rbuffer)</code> <code>smuX.measure.overlappediv(ibuffer, vbuffer)</code>  rbuffer            A reading buffer object where the reading(s) will be stored. ibuffer            A reading buffer object where current reading(s) will be stored. vbuffer            A reading buffer object where voltage reading(s) will be stored.	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will start a measurement and return immediately. The measurements, as they are performed, are stored in a reading buffer (along with any ancillary information also being acquired). If the instrument is configured to return multiple readings where one is requested, the readings will be available as they are made.</li> <li>• The <code>smuX.measure.overlappediv</code> function stores both current and voltage readings in respective buffers (current and then voltage are stored in separate buffers).</li> <li>• This function is an overlapped command. Script execution will continue while the measurement(s) is made in the background. Attempts to access result values that have not yet been generated will cause the script to block and wait for the data to become available. The <code>waitcomplete</code> function can also be used to wait for the measurement(s) to complete before continuing.</li> <li>• If a given reading buffer contains any data, it will be cleared prior to taking any measurements, unless the reading buffer has been configured to append data.</li> </ul>	
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.nvbufferY</a> , <a href="#">smuX.nvbufferY</a> , <a href="#">waitcomplete</a>	
<b>Example</b>	Starts background voltage measurements for SMU A: <code>smua.measure.overlappedv(smua.nvbuffer1)</code>	

<b>smuX.measure.rangeY</b>	
X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current	
<b>Attribute</b>	Fixed measure range for voltage or current.
<b>Usage</b>	<pre>rangeval = smuX.measure.rangeY    -- Reads measure range. smuX.measure.rangeY = rangeval    -- Writes measure range. rangeval    Set to the expected voltage or current to be measured.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Reading this attribute returns the positive full-scale value of the measure range the SMU is currently using.</li> <li>• Assigning to this attribute sets the SMU on a fixed range large enough to measure the given value. The instrument will select the best range for measuring a value of rangeval.</li> <li>• This attribute is primarily intended to eliminate the time required by the automatic range selection performed by a measuring instrument. Because selecting a fixed range will prevent auto-ranging, an over-range condition can occur, for example, measuring 10.0V on the 6V range will cause an over-range. The value 9.91000E+37 is returned when this occurs.</li> <li>• If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is locked to be the same as the source range. However, the setting for the voltage measure range is retained and used when the source function is changed to current, and the present voltage measurement range will be used. For example, assume the source function is voltage. The source range is 1V and you set the measure range for 6V. Since the source range is 1V, the SMU will perform voltage measurements on the 1V range. If you now change the function to current, measurements will be performed on the 6V range.</li> <li>• Explicitly setting either a source or measurement range for a function will disable auto ranging for that function. Auto ranging is controlled separately for each source and measurement function: source voltage, source current, measure voltage and measure current. Auto ranging is enabled for all four by default.</li> <li>• Changing the range while the output is off will not update the hardware settings, but querying will return the range setting that will be used once the output is turned on. Setting a range while the output is on will take effect immediately.</li> <li>• With source auto ranging enabled, the output level controls the range. Querying the range after the level is set will return the range the unit chose as appropriate for that source level.</li> <li>• With measure auto ranging enabled, the range will be changed only when a measurement is taken. Querying the range after a measurement will return the range selected for that measurement.</li> </ul>
<b>Details</b>	See “Range” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.measure.autorangeY</a>
<b>Example</b>	Selects 1V measure range for SMU A: <pre>smua.measure.rangev = 0.5</pre>

<b>smuX.measure.rel.enableY</b>		X = SMU channel (a or b) Y = SMU measure function (v, i, r or p) where: v = voltage, i = current, r = resistance, p = power
<b>Attribute</b>	Relative measurement control (on/off).	
<b>Usage</b>	<pre>rel = smuX.measure.rel.enableY      -- Reads relative state. smuX.measure.rel.enableY = rel      -- Writes relative state.  Set rel to one of the following values: 0 Or smuX.REL_OFF      Disables relative measurements. 1 Or smuX.REL_ON      Enables relative measurements.</pre>	
<b>Remarks</b>	<p>When relative measurements are enabled, all subsequent measured readings will be offset by the specified relative offset value (see <code>smuX.measure.rel.levelY</code>). Specifically, each returned measured relative reading will be the result of the following calculation:</p> $\text{Relative reading} = \text{Actual measured reading} - \text{Relative offset value}$	
<b>Details</b>	See “Rel” in <a href="#">Section 6</a> .	
<b>Also see</b>	<a href="#">smuX.measure.rel.levelY</a>	
<b>Example</b>	Enables relative voltage measurements for SMU A: <code>smua.measure.rel.enablev = smua.REL_ON</code>	

<b>smuX.measure.rel.levelY</b>		X = SMU channel (a or b) Y = SMU measure function (v, i, r or p) where: v = voltage, i = current, r = resistance, p = power
<b>Attribute</b>	Offset value for relative measurements.	
<b>Usage</b>	<pre>relval = smuX.measure.rel.levelY    -- Reads relative offset level. smuX.measure.rel.levelY = relval    -- Writes relative offset level. relval                               Relative offset value.</pre>	
<b>Remarks</b>	<p>When relative measurements are enabled (see <a href="#">smuX.measure.rel.enableY</a>), all subsequent measured readings will be offset by the specified relative offset value. Specifically, each returned measured relative reading will be the result of the following calculation:</p> $\text{Relative reading} = \text{Actual measured reading} - \text{Relative offset value}$	
<b>Details</b>	See “Rel” in <a href="#">Section 6</a> .	
<b>Also see</b>	<a href="#">smuX.measure.rel.enableY</a>	
<b>Example</b>	Performs a voltage measurement and uses it as the relative offset value: <code>smua.measure.rel.levelv = smua.measure.v()</code>	

<b>smuX.measure.Y</b> <b>smuX.measure.iv</b>		X = SMU channel (a or b) Y = SMU measure function (v, i, r or p)    where: v = voltage, i = current, r = resistance, p = power
<b>Function</b>	Performs one or more measurements.	
<b>Usage</b>	There are three ways to use this function: <pre>reading = smuX.measure.Y()</pre> <pre>reading = smuX.measure.Y(rbuffer)</pre> <pre>reading = smuX.measure.iv(ibuffer, vbuffer)</pre> <pre>reading</pre> Returns the last reading of the measurement process. <pre>rbuffer</pre> A reading buffer object where all the reading(s) will be stored. <pre>ibuffer</pre> A reading buffer object where current reading(s) will be stored. <pre>vbuffer</pre> A reading buffer object where voltage reading(s) will be stored.	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function returns only the last actual measurement as <code>reading</code>. To use the additional information acquired while making a measurement, a reading buffer must be used. If the instrument is configured to return multiple readings when a measurement is requested, all readings will be available in <code>rbuffer</code> if one is provided, but only the last measurement will be returned as <code>reading</code>.</li> <li>• The <code>smuX.measure.iv</code> function stores both current and voltage readings in respective buffers (current and then voltage are stored in separate buffers).</li> <li>• The <code>smuX.measure.count</code> attribute determines how many measurements are performed. When using a buffer, it also determines the number of readings to store in the buffer.</li> </ul>	
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.nvbufferY</a> , <a href="#">smuX.nvbufferY</a>	
<b>Example</b>	Performs ten voltage measurements using SMU A and stores them in a buffer: <pre>smua.measure.count = 10</pre> <pre>smua.measure.v(smua.nvbuffer1)</pre>	

<b>smuX.measureYandstep</b> <b>smuX.measureivandstep</b>		X = SMU channel (a or b) Y = SMU measure function (v, i, iv, r or p) where: v = voltage, i = current, r = resistance, p = power
<b>Function</b>	Performs one or two measurements and then steps the source.	
<b>Usage</b>	This function can be used in two ways: reading = smuX.measureYandstep(sourcevalue) readings = smuX.measureivandstep(sourcevalue)  reading               Returns the measured reading before stepping the source. readings             Returns the two measured readings before stepping the source. sourcevalue         Source value to be set after the measurement is made.	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The smuX.measureYandstep function performs a measurement and then sets the source to sourcevalue. The smuX.measureivandstep function is similar, but performs two measurements; one for current (i) and one for voltage (v).</li> <li>• The specified source value should be appropriate for the selected source function. For example, if the source voltage function is selected, then sourcevalue is expected to be a new voltage level.</li> <li>• Both source and measure auto range must be disabled before using this function.</li> <li>• This function is provided for very fast execution of source-measure loops. The measurement will be made prior to stepping the source. Prior to using this function, and before any loop this function may be used in, the source value should be set to its initial level.</li> </ul>	
<b>Also see</b>	<a href="#">smuX.measure.Y</a>	
<b>Example</b>	This measure and step function measures current starting at a source value of 0V. After each current measurement, the source is stepped 100mV for the next current measurement. The final source level is 1V where current is again measured. <pre> local ivalues = {}  smua.source.rangev = 1 smua.source.levelv = 0 smua.measure.rangei = 0.01 smua.source.output = smua.OUTPUT_ON for index = 1, 10 do     ivalues[index] = smua.measureiandstep(index / 10) end ivalues[11] = smua.measure.i() </pre>	



<b>smuX.nvbufferY</b>	
	X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Non-volatile reading buffers.
<b>Usage</b>	<code>smuX.nvbufferY</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• There are two reading buffers: <code>smuX.nvbuffer1</code> and <code>smuX.nvbuffer2</code>.</li> <li>• All routines that return measurements can return them in reading buffers. Overlapped measurements are always returned in a reading buffer. Synchronous measurements return either a single-point measurement or can be stored in a reading buffer if passed to the measurement command.</li> <li>• The non-volatile reading buffers will retain their data between power cycles.</li> </ul>
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .
<b>Also see</b>	<a href="#">smuX.makebuffer</a> , <a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a>
<b>Example</b>	Store current readings from SMU A into buffer 1: <code>smua.measure.overlappedi(smua.nvbuffer1)</code>

<b>smuX.nvbufferY.appendmode</b>	
	X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Append mode for the reading buffer.
<b>Usage</b>	<pre>state = smuX.nvbufferY.appendmode    -- Reads append mode. smuX.nvbufferY.appendmode = state    -- Writes append mode.</pre> <p>Set <code>state</code> to one of the following values:</p> <ul style="list-style-type: none"> <li>0 Append mode off – New measure data overwrites the previous buffer content.</li> <li>1 Append mode on – Appends new measure data to the present buffer content.</li> </ul>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Assigning to this attribute enables or disables the buffer append mode.</li> <li>• With append mode on, the first new measurement will be stored at <code>rb[n+1]</code>, where <code>n</code> is the number of readings stored in the buffer.</li> <li>• The default is buffer append mode off.</li> </ul>
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>
<b>Example</b>	Append new readings for SMU A to contents of buffer 1: <code>smua.nvbuffer1.appendmode = 1</code>

<b>smuX.nvbufferY.basetimestamp</b>	
	X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Timestamp of when the first reading was stored from time of power-up.
<b>Usage</b>	<code>basetime = smuX.nvbufferY.basetimestamp</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Reading this attribute returns the timestamp (in seconds) for the first reading (<code>rb[1]</code>) stored in a buffer. The timestamp is based on the number of seconds from power-up that the measurement was performed and stored.</li> <li>• This is a read-only attribute.</li> </ul>
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>
<b>Example</b>	<p>Read the timestamp for the first reading stored in buffer 1 of SMU A:</p> <pre>basetime = smua.nvbuffer1.basetimestamp print(basetime)</pre> <p>Output: 2.369900+03</p> <p>The above output indicates that the timestamp is 2369.9 seconds.</p>

<b>smuX.nvbufferY.capacity</b>	
	X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Capacity of the buffer.
<b>Usage</b>	<code>capacity = smuX.nvbufferY.capacity</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Reading this attribute returns the number of readings that can be stored in the buffer.</li> <li>• A buffer with only basic collection items turned on can store over 100,000 readings. Capacity does not change as readings fill the buffer. Turning on additional collection items, such as timestamps and source values, decreases the capacity of the buffer.</li> <li>• This is a read-only attribute.</li> </ul>
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>
<b>Example</b>	<p>Read the capacity of SMU A buffer 1:</p> <pre>capacity = smua.nvbuffer1.capacity print(capacity)</pre> <p>Output: 1.123410+05</p>

<b>smuX.nvbufferY.clear</b>		X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Function</b>	Clears the buffer.	
<b>Usage</b>	<code>smuX.nvbufferY.clear()</code>	
<b>Remarks</b>	This function clears all readings from the indicated buffer.	
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>	
<b>Example</b>	Clears SMU A buffer 1: <code>smua.nvbuffer1.clear()</code>	

<b>smuX.nvbufferY.collectsourcevalues</b>		X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Source value collection for the buffer.	
<b>Usage</b>	<pre>state = smuX.nvbufferY.collectsourcevalues -- Reads collection state. smuX.nvbufferY.collectsourcevalues = state -- Writes collection state.</pre> <p>Set <code>state</code> to one of the following values:</p> <ul style="list-style-type: none"> <li>0 Source value collection disabled (off).</li> <li>1 Source value collection enabled (on).</li> </ul>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Assigning a <code>state</code> value to this attribute enables or disables the storage of source values. Reading this attribute returns the <code>state</code> of source value collection.</li> <li>• When on, source values will be stored with readings in the buffer. This requires four extra bytes of storage per reading.</li> <li>• This value, off (default) or on, can only be changed when the buffer is empty. The buffer can be emptied using the <a href="#">smuX.nvbufferY.clear</a> function.</li> </ul>	
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>	
<b>Example</b>	Include source values with readings for SMU A buffer 1: <code>smua.nvbuffer1.collectsourcevalues = 1</code>	

<b>smuX.nvbufferY.collecttimestamps</b>		X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Timestamp collection for the buffer.	
<b>Usage</b>	<pre>state = smuX.nvbufferY.collecttimestamps -- Reads collection state. smuX.nvbufferY.collecttimestamps = state -- Writes collection state.</pre> <p>Set <code>state</code> to one of the following values:</p> <ul style="list-style-type: none"> <li>0 Timestamp collection disabled (off).</li> <li>1 Timestamp collection enabled (on).</li> </ul>	

<b>Remarks</b>	<ul style="list-style-type: none"> <li>Assigning a <code>state</code> value to this attribute enables or disables the storage of timestamps. Reading this attribute returns the <code>state</code> of timestamp collection.</li> <li>When on, timestamps will be stored with readings in the buffer. This requires four extra bytes of storage per reading. The first reading is timestamped at zero seconds. Subsequent readings are timestamped relative to the time storage was started.</li> <li>This value, off (default) or on, can only be changed when the buffer is empty. The buffer can be emptied using the <code>smuX.nvbufferY.clear</code> function.</li> </ul>
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on page 12-6 and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>
<b>Example</b>	Include timestamps with readings for SMU A buffer 1: <code>smua.nvbuffer1.collecttimestamps = 1</code>

<b>smuX.nvbufferY.n</b>		X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Number of readings in the buffer.	
<b>Usage</b>	<code>bufferreadings = smuX.nvbufferY.n</code>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>Reading this attribute returns the number of readings that are stored in the buffer.</li> <li>This is a read-only attribute.</li> </ul>	
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on page 12-6 and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>	
<b>Example</b>	Read the number of readings stored in SMU A buffer 1: <code>bufferreadings = smua.nvbuffer1.n</code> <code>print(bufferreadings)</code> Output: 1.250000+02 The above output indicates that there are 125 readings stored in the buffer.	

<b>smuX.nvbufferY.timestampresolution</b>		X = SMU channel (a or b) Y = NV buffer (1 or 2)
<b>Attribute</b>	Timestamp resolution.	
<b>Usage</b>	<pre>tsres = smuX.nvbufferY.timestampresolution -- Reads collection state. smuX.nvbufferY.timestampresolution = tsres -- Writes collection state. tsres      Timestamp resolution in seconds.</pre>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>Assigning to this attribute sets the resolution for the timestamps. Reading this attribute returns the timestamp resolution value.</li> <li>The default timestamp resolution is 0.000001seconds (1µs). At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. This value can be increased for very long tests.</li> <li>The minimum value that can be set is 1µs.</li> <li>When setting this value it will be rounded to an even power of 2µs.</li> </ul>	
<b>Details</b>	See “ <a href="#">Reading buffers</a> ” on <a href="#">page 12-6</a> and “ <a href="#">Reading buffers</a> ” in <a href="#">Section 7</a> .	
<b>Also see</b>	<a href="#">smuX.measure.overlappedY</a> , <a href="#">smuX.measure.Y</a> , <a href="#">smuX.nvbufferY</a>	
<b>Example</b>	Set the timestamp resolution for SMU A buffer 1 to 10µs: <pre>smua.nvbuffer1.timestampresolution = 0.00001</pre>	

<b>smuX.reset</b>		X = SMU channel (a or b)
<b>Function</b>	Turns off the output and resets the SMU to the default settings.	
<b>Usage</b>	<pre>smuX.reset()</pre>	
<b>Remarks</b>	Returns the SMU to the default settings listed in <a href="#">Table 1-3</a> .	
<b>Details</b>	See “ <a href="#">Default settings</a> ” in <a href="#">Section 1</a> .	
<b>Also see</b>	<a href="#">reset</a>	

<b>smuX.sense</b>		X = SMU channel (a or b)
<b>Attribute</b>	Remote/local sense mode.	
<b>Usage</b>	<pre>sense = smuX.sense      -- Reads sense mode. smuX.sense = sense      -- Writes sense mode.</pre> <p>Set <i>sense</i> to one of the following values:</p> <pre>0 or smuX.SENSE_LOCAL   Selects local sense (2-wire). 1 or smuX.SENSE_REMOTE  Selects remote sense (4-wire). 3 or smuX.SENSE_CALA    Selects calibration sense mode.</pre>	

<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Source-measure operations are performed using either 2-wire local sense connections or 4-wire remote sense connections. Writing to this attribute selects the sense mode.</li> <li>• The <code>smuX.SENSE_CALA</code> mode is only used for calibration and may only be selected when calibration is enabled.</li> <li>• The sense mode can be changed between local and remote while the output is on.</li> <li>• The calibration sense mode cannot be selected while the output is on.</li> <li>• The <code>reset</code> function selects the local sense mode.</li> </ul>
<b>Details</b>	See “ <a href="#">Sensing methods</a> ” in <a href="#">Section 3</a> .
<b>Example</b>	Selects remote sensing for SMU A: <code>smua.sense = smua.SENSE_REMOTE</code>

<b>smuX.source.autorangeY</b>					
X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current					
<b>Attribute</b>	Source auto range control (on/off).				
<b>Usage</b>	<pre>sautorange = smuX.source.autorangeY    -- Reads source auto range. smuX.source.autorangeY = sautorange     -- Writes source auto range.</pre> <p>Set <code>autorange</code> to one of the following values:</p> <table> <tr> <td>0 or <code>smuX.AUTORANGE_OFF</code></td> <td>Disables source auto range.</td> </tr> <tr> <td>1 or <code>smuX.AUTORANGE_ON</code></td> <td>Enables source auto range.</td> </tr> </table>	0 or <code>smuX.AUTORANGE_OFF</code>	Disables source auto range.	1 or <code>smuX.AUTORANGE_ON</code>	Enables source auto range.
0 or <code>smuX.AUTORANGE_OFF</code>	Disables source auto range.				
1 or <code>smuX.AUTORANGE_ON</code>	Enables source auto range.				
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute indicates the source auto range state. Its value will be <code>smuX.AUTORANGE_OFF</code> when the SMU source circuit is on a fixed range and <code>smuX.AUTORANGE_ON</code> when it is in auto range mode.</li> <li>• Setting this attribute to <code>smuX.AUTORANGE_OFF</code> puts the SMU on a fixed source range. The fixed range used will be the range the SMU source circuit was currently using.</li> <li>• Setting this attribute to <code>smuX.AUTORANGE_ON</code> puts the SMU source circuit into auto range mode. If the source output is on, the SMU will immediately change range to the range most appropriate for the value being sourced if that range is different from the SMU range.</li> <li>• Auto range will disable if the source level is edited from the front panel.</li> </ul>				
<b>Details</b>	See “ <a href="#">Range</a> ” in <a href="#">Section 6</a> .				
<b>Also see</b>	<a href="#">smuX.measure.autorangeY</a> , <a href="#">smuX.source.rangeY</a>				
<b>Example</b>	Enables volts source auto range for SMU A: <code>smua.source.autorangev = smua.AUTORANGE_ON</code>				

<b>smuX.source.calibrateY</b>	
X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current	
<b>Function</b>	Generates and activates new source calibration constants.
<b>Usage</b>	<pre>smuX.source.calibrateY(range, cp1expected, cp1reference, cp2expected, cp2reference)</pre> <p>range                    The measurement range to calibrate.</p> <p>cp1expected            The source value programmed for calibration point 1.</p> <p>cp1reference            The reference measurement for calibration point 1 as measured externally.</p> <p>cp2expected            The source value programmed for calibration point 2.</p> <p>cp2reference            The reference measurement for calibration point 2 as measured externally.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function generates and activates new calibration constants for the given range. The positive and negative polarities of the source must be calibrated separately. Use a positive value for <code>range</code> to calibrate the positive polarity and a negative value for <code>range</code> to calibrate the negative polarity.</li> <li>• Typically the two calibration points used will be near zero for calibration point 1 and 90% of full scale for calibration point 2. Full scale for calibration point 2 should be avoided if the SMU's source is substantially out of calibration.</li> <li>• Do not use 0.0 for a negative calibration point as 0.0 is considered a positive number.</li> <li>• The two reference measurements must be made with the source using the active calibration set. For example, source a value, measure it, and do not change the active calibration set before issuing this command.</li> <li>• The new calibration constants will be activated immediately but they will not be written to non-volatile storage. Use <code>smuX.cal.save</code> to commit the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from non-volatile storage with the <code>smuX.cal.restore</code> function.</li> <li>• This function will be disabled until a successful call to <code>smuX.cal.unlock</code> is made.</li> </ul>
<b>Details</b>	See <a href="#">Section 17</a> (calibration).
<b>Also see</b>	<a href="#">smuX.cal.restore</a> , <a href="#">smuX.cal.save</a> , <a href="#">smuX.makebuffer</a> , <a href="#">smuX.measure.calibrateY</a>

<b>smuX.source.compliance</b> <span style="float: right;">X = SMU channel (a or b)</span>	
<b>Attribute</b>	Source compliance state.
<b>Usage</b>	<code>compliance = smuX.source.compliance</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Use this attribute to read the state of source compliance. <code>true</code> indicates that the limit function is in control of the source (source in compliance). <code>false</code> indicates that the source function is in control of the output (source not in compliance).</li> <li>• This is a read-only attribute. Writing to this attribute will generate an error.</li> <li>• Reading this attribute also updates the status model and the front panel with generated compliance information.</li> </ul>
<b>Details</b>	See <a href="#">Section 4</a> (basic operation) and <a href="#">Appendix D</a> (status model).
<b>Also see</b>	<a href="#">smuX.source.limitY</a>
<b>Example</b>	<p>Reads the source compliance state for SMU A:</p> <pre>compliance = smua.source.compliance print(compliance) Output: true</pre> <p>The above output indicates that the voltage limit has been reached (if configured as a current source), or that the current limit has been reached (if configured as a voltage source).</p>

<b>smuX.source.func</b> <span style="float: right;">X = SMU channel (a or b)</span>					
<b>Attribute</b>	Source function.				
<b>Usage</b>	<pre>iv = smuX.source.func          -- Reads source function. smuX.source.func = iv         -- Writes source function.</pre> <p>Set <code>iv</code> to one of the following values:</p> <table style="width: 100%; border: none;"> <tr> <td style="width: 50%;"><code>0</code> or <code>smuX.OUTPUT_DCAMPS</code></td> <td>Selects current source function.</td> </tr> <tr> <td style="width: 50%;"><code>1</code> or <code>smuX.OUTPUT_DCVOLTS</code></td> <td>Selects voltage source function.</td> </tr> </table>	<code>0</code> or <code>smuX.OUTPUT_DCAMPS</code>	Selects current source function.	<code>1</code> or <code>smuX.OUTPUT_DCVOLTS</code>	Selects voltage source function.
<code>0</code> or <code>smuX.OUTPUT_DCAMPS</code>	Selects current source function.				
<code>1</code> or <code>smuX.OUTPUT_DCVOLTS</code>	Selects voltage source function.				
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Reading this attribute gives the output function of the source. Setting this attribute configures the SMU as either a voltage source or a current source.</li> <li>• The <code>reset</code> function selects the voltage function.</li> </ul>				
<b>Details</b>	See <a href="#">Section 4</a> (basic operation).				
<b>Also see</b>	<a href="#">smuX.source.output</a> , <a href="#">smuX.source.levelY</a>				
<b>Example</b>	<p>Selects the source amps function for SMU A:</p> <pre>smua.source.func = smua.OUTPUT_DCAMPS</pre>				



<b>smuX.source.levelY</b>		X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
<b>Attribute</b>	Source levels.	
<b>Usage</b>	<pre>sourceval = smuX.source.levelY      -- Reads source value. smuX.source.levelY = sourceval      -- Writes source value.  sourceval      Set voltage from 0 to ±40 (volts).                 Set current from 0 to ±3 (amps).</pre>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute configures the source level of the voltage or current source.</li> <li>• If the source is configured as a voltage source and the output is on, the new <code>smuX.source.levelv</code> setting will be sourced immediately. If the output is off or if the source is configured as a current source, the voltage level will be sourced when the source is configured as a voltage source and the output is turned on.</li> <li>• If the source is configured as a current source and the output is on, the new <code>smuX.source.leveli</code> setting will be sourced immediately. If the output is off or if the source is configured as a voltage source, the current level will be sourced when the source is configured as a current source and the output is turned on.</li> <li>• The sign of <code>sourceval</code> 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.</li> <li>• The <code>reset</code> function sets the source levels to 0V and 0A.</li> </ul>	
<b>Details</b>	See <a href="#">Section 4</a> (basic operation)	
<b>Also see</b>	<a href="#">smuX.source.func</a> , <a href="#">smuX.source.output</a>	
<b>Example</b>	Sets V-source to 1V for SMU A: <code>smua.source.levelv = 1</code>	

<b>smuX.source.limitY</b>		X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
<b>Attribute</b>	Compliance limits.	
<b>Usage</b>	<pre>limit = smuX.source.limitY      -- Reads compliance limit. smuX.source.limitY = limit      -- Writes compliance limit.  limit      Voltage compliance from 0 to ±40 (volts).             Current compliance from 0 to ±3 (amps).</pre>	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Use the <code>smuX.source.limiti</code> attribute to limit the current output of the voltage source. Use <code>smuX.source.limitv</code> to limit the voltage output of the current source. The SMU will always choose (auto range) the source range for the limit setting.</li> <li>• This attribute should be set in the test sequence before the turning the source on.</li> <li>• Reading this attribute indicates the presently set compliance value. Use <code>smuX.source.compliance</code> to read the state of source compliance.</li> </ul>	
<b>Details</b>	See <a href="#">Section 4</a> (basic operation).	

<b>Also see</b>	<a href="#">smuX.source.compliance</a> , <a href="#">smuX.source.func</a> , <a href="#">smuX.source.output</a>
<b>Example</b>	Sets V-compliance to 30V for SMU A: <code>smua.source.limitv = 30</code>

<b>smuX.source.lowrangeY</b>		X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
<b>Attribute</b>	Lowest source range that will be used during autoranging.	
<b>Usage</b>	<code>rangeval = smuX.source.lowrangeY</code> -- Reads low range. <code>smuX.source.lowrangeY = rangeval</code> -- Writes low range. <code>rangeval</code> Set to the lowest voltage or current to be sourced.	
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute is used with source autoranging to put a lower bound on the range used. Lower ranges generally require greater settling times. By setting a low range value, sourcing small values might be able to be made with less settling time.</li> <li>• If the instrument is set to auto range and it is on a range lower than the one specified by <code>rangeval</code>, the source range will be changed to the range specified by <code>rangeval</code>.</li> </ul>	
<b>Details</b>	See “Range” in <a href="#">Section 6</a> .	
<b>Also see</b>	<a href="#">smuX.source.autorangeY</a> , <a href="#">smuX.source.rangeY</a>	
<b>Example</b>	Sets volts lowrange for SMU A to 1V. This prevents the source from using the 100mV range when sourcing voltage: <code>smua.source.lowrangev = 1</code>	

<b>smuX.source.offmode</b>		X = SMU channel (a or b)
<b>Attribute</b>	Source output-off mode.	
<b>Usage</b>	<code>offmode = smuX.source.offmode</code> -- Reads output-off mode. <code>smuX.source.offmode = offmode</code> -- Writes output-off mode. Set <code>offmode</code> to one of the following values: 0 or <code>smuX.OUTPUT_NORMAL</code> Outputs 0V when the output is turned off. 1 or <code>smuX.OUTPUT_ZERO</code> Zero the output (in either volts or current) when off. 2 or <code>smuX.OUTPUT_HIGH_Z</code> Opens the output relay when the output is turned off.	

<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Reading this attribute gives the output-off mode of the source. Setting this attribute configures the SMU output-off mode.</li> <li>• The default <code>offmode</code> is <code>smuX.OUTPUT_NORMAL</code>. In this mode, the SMU will source 0 volts and set the compliance to 10% of the current source range or 100uA, whichever is smaller. If the source function is voltage, the 10% compliance will inherently be a reduction in compliance current. If the source function is current, the 10% compliance value may be more or less than the current that was being sourced.</li> <li>• When <code>offmode</code> is set to <code>smuX.OUTPUT_HIGH_Z</code>, the SMU will open the output relay when the output is turned off.</li> <li>• When the <code>offmode</code> is set to <code>smuX.OUTPUT_ZERO</code>, the SMU will source 0 volts just as <code>OUTPUT_NORMAL</code> mode does. If the source function is voltage, the current limit will not be changed. If the source function was current, the current limit will be set to the current source level or 10% of the current source range, whichever is greater.</li> </ul>
<b>Details</b>	See “Output-off states” in <a href="#">Section 3</a> .
<b>Also see</b>	<a href="#">smuX.source.output</a>
<b>Example</b>	Sets output-off mode for SMU A: <code>smua.source.offmode = smua.OUTPUT_HIGH_Z</code>

<b>smuX.source.output</b>		X = SMU channel (a or b)
<b>Attribute</b>	Source output control (on/off).	
<b>Usage</b>	<pre>state = smuX.source.output      -- Reads output state. smuX.source.output = state      -- Writes output state.  Set state to one of the following values: 0 or smuX.OUTPUT_OFF           Turns the source output off. 1 or smuX.OUTPUT_ON            Turns the source output on.</pre>	
<b>Remarks</b>	Reading this attribute gives the output state of the source. Setting this attribute will turn the output of the source on or off. The default for the source is off. When the output is turned on, the SMU will source either voltage or current as dictated by the <code>smuX.source.func</code> setting.	
<b>Details</b>	See <a href="#">Section 4</a> (basic operation).	
<b>Also see</b>	<a href="#">smuX.source.func</a> , <a href="#">smuX.source.offmode</a>	
<b>Example</b>	Turns SMU A source output on: <code>smua.source.output = smua.OUTPUT_ON</code>	

<b>smuX.source.outputenableaction</b> X = SMU channel (a or b)	
<b>Attribute</b>	Output enable action for the source.
<b>Usage</b>	<pre>action = smuX.source.outputenableaction -- Reads enable action. smuX.source.outputenableaction = action -- Writes enable action.</pre> <p>Set <code>action</code> to one of the following values:</p> <pre>0 or smuX.OE_NONE           No action. 1 or smuX.OE_OUTPUT_OFF    Turns the source output off.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute controls the SMU action taken when the output enable line is asserted/de-asserted. The default setting is <code>smuX.OE_NONE</code>.</li> <li>• When set to <code>smuX.OE_NONE</code>, the SMU will take no action when the output enable line goes low (de-asserted).</li> <li>• When set to <code>smuX.OE_OUTPUT_OFF</code> and the output enable line is de-asserted, the SMU will turn its output off as if the <code>smuX.source.output = smuX.OUTPUT_OFF</code> command had been received.</li> <li>• The SMU will not automatically turn its output on when the output enable line returns to the high state.</li> <li>• If the output enable line is not asserted when this attribute is set to <code>smuX.OE_OUTPUT_OFF</code> and the output is on, the output will turn off immediately.</li> <li>• Detection of the output enable line going low will not abort any running scripts. This may cause execution errors.</li> </ul>
<b>Details</b>	See “ <a href="#">Output enable</a> ” in <a href="#">Section 10</a> .
<b>Also see</b>	<a href="#">smuX.source.offmode</a> , <a href="#">smuX.source.output</a>
<b>Example</b>	Reconfigures the SMU to turn the output off if the output enable line goes low (de-asserted): <pre>smua.source.outputenableaction = smua.OE_OUTPUT_OFF</pre>

<b>smuX.source.rangeY</b> X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current	
<b>Attribute</b>	Source range.
<b>Usage</b>	<pre>rangeval = smuX.source.rangeY -- Reads source range. smuX.source.rangeY = rangeval -- Writes source range.</pre> <p><code>rangeval</code> The expected voltage or current to be sourced.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Reading this attribute returns the positive full-scale value of the source range the SMU is currently using. Assigning to this attribute sets the SMU on a fixed range large enough to source the given value. The instrument will select the best range for sourcing a value of <code>rangeval</code>.</li> <li>• <code>smuX.source.rangeX</code> is primarily intended to eliminate the time required by the automatic range selection performed by a sourcing instrument. Because selecting a fixed range will prevent auto-ranging, an over-range condition can occur, for example, sourcing 10.0V on the 6.0V range.</li> </ul>

<b>Details</b>	See “Range” in <a href="#">Section 6</a> .
<b>Also see</b>	<a href="#">smuX.source.autorangeY</a>
<b>Example</b>	Selects 1V source range for SMU A: <code>smua.source.rangev = 1</code>

## timer functions

The functions in this group are used for the timer. The timer can be used to measure the time it takes to perform various operations. Use the [timer.reset](#) function at the beginning of an operation to reset the timer to zero, and then use the [timer.measure.t](#) at the end of the operation to measure the elapsed time.

<b>timer.measure.t</b>	
<b>Function</b>	Measures the elapsed time since the timer was last reset.
<b>Usage</b>	<code>time = timer.measure.t()</code> <code>time</code> Returns the elapsed time in seconds. (1 $\mu$ s resolution).
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This function will return the elapsed time in seconds since the timer was reset.</li> <li>• The returned resolution for <code>time</code> depends on how long it has been since the timer was reset. It starts with 1<math>\mu</math>s resolution and starts to lose resolution after about 2.8 minutes.</li> </ul>
<b>Also see</b>	<a href="#">timer.reset</a>
<b>Example</b>	Resets the timer and then measures the time since the reset: <pre>timer.reset() ... time = timer.measure.t() print(time)</pre> Output: 1.469077e+01 The above output indicates that <code>timer.measure.t</code> was executed 14.69077 seconds after <code>timer.reset</code> .

<b>timer.reset</b>	
<b>Function</b>	Resets the timer to 0 seconds.
<b>Usage</b>	<code>timer.reset()</code>
<b>Remarks</b>	This function will restart the timer at zero.
<b>Also see</b>	<a href="#">timer.measure.t</a>

<b>Example</b>	<p>Resets the timer and then measures the time since the reset:</p> <pre>timer.reset() ... time = timer.measure.t() print(time)</pre> <p>Output: 1.469077e+01</p> <p>The above output indicates that <code>timer.measure.t</code> was executed 14.69077 seconds after <code>timer.reset</code>.</p>
----------------	---

## trigger functions

The functions in this group are used to control triggering.

<b>trigger.clear</b>	
<b>Function</b>	Clears the command interface trigger event detector.
<b>Usage</b>	<code>trigger.clear()</code>
<b>Remarks</b>	The trigger event detector remembers if an event has been detected since the last <code>trigger.wait</code> call. This function clears the trigger's event detector and discards the previous history of command interface trigger events.
<b>Details</b>	see "Triggering" in <a href="#">Section 4</a> .
<b>Also see</b>	<a href="#">trigger.wait</a>

<b>trigger.wait</b>	
<b>Function</b>	Wait for a trigger event.
<b>Usage</b>	<pre>triggered = trigger.wait(timeout)</pre> <p><code>timeout</code>           Maximum amount of time in seconds to wait for the trigger.</p> <p><code>triggered</code>         Returns <code>true</code> if a trigger was detected. Returns <code>false</code> if no triggers were detected during the timeout period.</p>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>This function will wait up to <code>timeout</code> seconds for a GPIB GET command or a *TRG message on the GPIB interface if that is the active command interface or a *TRG message on the command interface for all other interfaces. If one or more of these trigger events were previously detected, this function will return immediately.</li> <li>After waiting for a trigger with this function, the event detector will be automatically reset and rearmed. This is true regardless of the number of events detected.</li> </ul>
<b>Details</b>	see "Triggering" in <a href="#">Section 4</a>
<b>Also see</b>	<a href="#">trigger.clear</a>

<b>Example</b>	<p>Waits up to 10 seconds for a trigger:</p> <pre>triggered = trigger.wait(10) print(triggered)</pre> <p>Output: false</p> <p>The above output indicates that no trigger was detected during the 10 second timeout.</p>
----------------	---

## tsplink function and attributes

The function and attributes in this group are used to assign node numbers to Model 260x instruments and initialize the TSP-Link system.

<b>tsplink.node</b>	
<b>Attribute</b>	TSP-Link node number.
<b>Usage</b>	<pre>mynode = tsplink.node          -- Reads the node number. tsplink.node = mynode         -- Writes the node number. mynode          Set node number from 1 to 64.</pre>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute sets the TSP-Link node number and saves the value in non-volatile memory.</li> <li>• After changing the node number, it will not take effect until the next time <code>tsplink.reset</code> is executed on any node in the system.</li> <li>• Each node connected to the TSP-Link must be assigned a different node number.</li> </ul>
<b>Details</b>	See <a href="#">Section 9</a> (system expansion).
<b>Also see</b>	<a href="#">tsplink.reset</a> , <a href="#">tsplink.state</a>
<b>Example</b>	<p>Sets the TSP-Link node to number 2:</p> <pre>tsplink.node = 2</pre>

<b>tsplink.reset</b>	
<b>Function</b>	Initializes (resets) all nodes (instruments) in the TSP-Link system.
<b>Usage</b>	<code>tsplink.reset()</code>
<b>Remarks</b>	This function will erase all knowledge of other nodes connected on the TSP-Link and will regenerate the system configuration. This function must be called at least once before any remote nodes can be accessed. If the node number for any instrument is changed, the TSP-Link must again be initialized.
<b>Details</b>	See <a href="#">Section 9</a> (system expansion).
<b>Also see</b>	<a href="#">tsplink.node</a> , <a href="#">tsplink.state</a>

<b>tsplink.state</b>	
<b>Attribute</b>	TSP-Link on-line state.
<b>Usage</b>	<code>state = tsplink.state</code>
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• This attribute stores the TSP-Link status, either <code>online</code> or <code>offline</code>. The state will be “offline” after the unit is powered on. After <code>tsplink.reset</code> is successful, the state will be “online”.</li> <li>• This attribute is read-only.</li> </ul>
<b>Details</b>	See <a href="#">Section 9</a> (system expansion).
<b>Also see</b>	<a href="#">tsplink.node</a> , <a href="#">tsplink.reset</a>
<b>Example</b>	Reads the on-line state of the TSP-Link: <pre>state = tsplink.state print(state)</pre> Output: <code>online</code>

## userstring functions

The functions in this group are used to store/retrieve user-defined strings in non-volatile memory.

<b>userstring.add</b>	
<b>Function</b>	Adds a user-defined string to non-volatile memory.
<b>Usage</b>	<pre>userstring.add(name, value)</pre> <p><code>name</code>                    The name for the string.  <code>value</code>                    The string to associate with the name.</p>
<b>Remarks</b>	This function will associate the string <code>value</code> with the string <code>name</code> and store the pair in non-volatile memory. The value associated with the given name can be retrieved with the <code>userstring.get</code> function.
<b>Also see</b>	<a href="#">userstring.catalog</a> , <a href="#">userstring.delete</a> , <a href="#">userstring.get</a>
<b>Example</b>	Stores user-defined strings in non-volatile memory: <pre>userstring.add("assetnumber", "236") userstring.add("department", "Widgets") userstring.add("contact", "John Doe")</pre>



<b>userstring.catalog</b>	
<b>Function</b>	Creates an iterator for the user string catalog.
<b>Usage</b>	<code>for name in userstring.catalog() do ... end</code>
<b>Remarks</b>	Accessing the catalog for user string names allows the user to print or delete all string name values in non-volatile memory. The entries will be enumerated in no particular order.
<b>Also see</b>	<a href="#">userstring.add</a> , <a href="#">userstring.delete</a> , <a href="#">userstring.get</a>
<b>Example</b>	<p>Deletes all user strings in non-volatile memory:</p> <pre>for name in userstring.catalog() do   userstring.delete(name) end</pre> <p>Prints all user string name value pairs in non-volatile memory:</p> <pre>for name in userstring.catalog() do   print(name .. " = " .. userstring.get(name)) end</pre> <p>Output: department = Widgets  assetnumber = 236  contact = John Doe</p> <p>The above output lists the user strings added in the “Example” for the <a href="#">userstring.add</a> function. Notice that they are not listed in the order that they were added.</p>

<b>userstring.delete</b>	
<b>Function</b>	Deletes a user-defined string from non-volatile memory.
<b>Usage</b>	<code>userstring.delete(name)</code> name                   Name of the user string.
<b>Remarks</b>	This function will delete from non-volatile memory the string that is associated with the string name.
<b>Also see</b>	<a href="#">userstring.add</a> , <a href="#">userstring.catalog</a> , <a href="#">userstring.get</a>
<b>Example</b>	Deletes user-defined strings from non-volatile memory: <pre>userstring.delete("assetnumber") userstring.delete("department") userstring.delete("contact")</pre>

<b>userstring.get</b>	
<b>Function</b>	Retrieves a user-defined string from non-volatile memory.
<b>Usage</b>	<pre>value = userstring.get (name) name      Name of the user string. value     Returns the string value associated with name.</pre>
<b>Remarks</b>	This function will retrieve from non-volatile memory the string that is associated with the string name.
<b>Also see</b>	<a href="#">userstring.add</a> , <a href="#">userstring.catalog</a> , <a href="#">userstring.delete</a>
<b>Example</b>	Retrieves the value for a user string from non-volatile memory: <pre>value = userstring.get ("assetnumber") print (value) Output: 236</pre>

## waitcomplete function

This function waits for all overlapped commands to complete.

<b>waitcomplete</b>	
<b>Function</b>	Waits for all overlapped commands to complete.
<b>Usage</b>	<code>waitcomplete()</code>
<b>Remarks</b>	This function will wait for all previously started overlapped commands to complete.

# 13

# Factory Scripts

---

## **Section 13 topics**

**Introduction**, page 13-2

**Factory script**, page 13-2  
KIGeneral, page 13-2

**Flash firmware upgrade**, page 13-13

## Introduction

The Model 260x is shipped with one or more Factory Scripts saved in its flash firmware memory. A factory script is made up of a number of functions that can be called from the front panel LOAD TEST menu, or called using remote programming.

As Keithley develops additional factory scripts, they will be made available on the Keithley web site ([www.keithley.com](http://www.keithley.com)) as a flash firmware upgrade for the Model 260x. See “[Flash firmware upgrade](#)” for instructions on upgrading the flash firmware of your Model 260x.

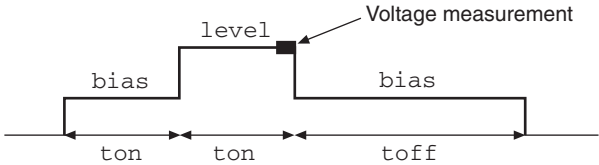
## Factory script

### KIGeneral

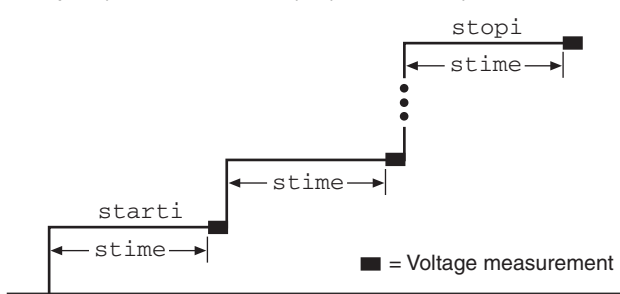
The KIGeneral factory script is made up of the following functions:

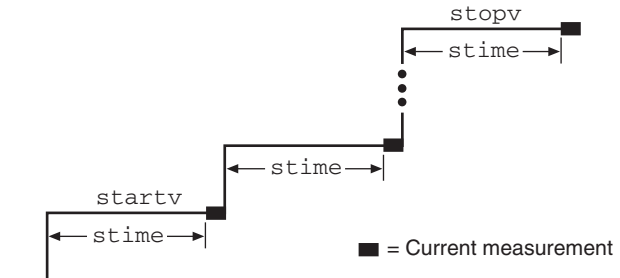
```
PulseIMeasureV(smu, bias, level, ton, toff, points)
PulseVMeasureI(smu, bias, level, ton, toff, points)
SweepILinMeasureV(smu, starti, stopi, stime, points)
SweepVLinMeasureI(smu, startv, stopv, stime, points)
SweepILogMeasureV(smu, starti, stopi, stime, points)
SweepVLogMeasureI(smu, startv, stopv, stime, points)
SweepIListMeasureV(smu, ilist, stime, points)
SweepVListMeasureI(smu, vlist, stime, points)
```

Details on the above functions are provided in the following tables.

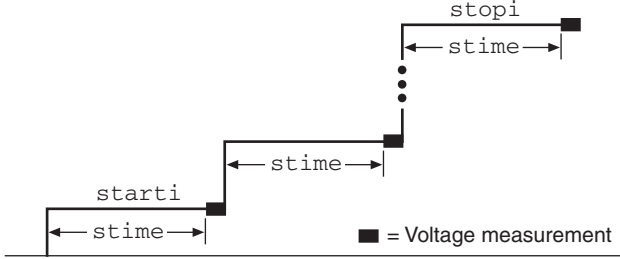
TSP Project Name:           KIFactoryGeneral TSP Test Script Name:       KIGeneral Version: 1.0.3RC1	
Function: PulseIMeasureV(smu, bias, level, ton, toff, points)	
Description:	<p>Performs a specified number of pulse I, measure V cycles:</p> <ul style="list-style-type: none"> <li>• Sets the smu to output bias amps and dwell for ton seconds.</li> <li>• Sets the smu to output level amps and dwell for ton seconds.</li> <li>• Performs voltage measurement with source at level amps.</li> <li>• Sets the smu to output bias amps for toff seconds.</li> <li>• Repeats the above sequence for points pulse-measure cycles.</li> </ul> 
Parameters:	smu, bias, level, ton, toff, points smu:       SourceMeter Channel (A or B). Defaults to smua if all parameters are omitted when function is called. bias:       Bias level in amps. level:      Pulse level in amps. ton:        Pulse on-time in seconds. toff:       Pulse off-time in seconds. points:     Number of pulse-measure cycles.
Data:	Pulsed voltage measurements, current levels and timestamps are stored in smuX.nvbuffer1.
Example:	PulseIMeasureV(smua, 0.001, 1.0, 20E-3, 40E-3, 10) SMU A will output 1mA and dwell for 20ms, output 1A and dwell for 20ms, and then perform a voltage measurement. After the measurement, the output will return to 1mA and dwell for 40ms. This pulse-measure process will repeat 9 more times.

TSP Project Name:        KIFactoryGeneral TSP Test Script Name:    KIGeneral Version: 1.0.3RC1	
Function: PulseVMeasureI(smu, bias, level, ton, toff, points)	
Description:	<p>Performs a specified number of pulse V, measure I cycles:</p> <ul style="list-style-type: none"> <li>• Sets the smu to output <code>bias</code> volts and dwell for <code>ton</code> seconds.</li> <li>• Sets the smu to output <code>level</code> volts and dwell for <code>ton</code> seconds.</li> <li>• Performs current measurement with source at <code>level</code> amps.</li> <li>• Sets the smu to output <code>bias</code> volts for <code>toff</code> seconds.</li> <li>• Repeats the above sequence for <code>points</code> pulse-measure cycles.</li> </ul>
Parameters:	<p>smu, bias, level, ton, toff, points</p> <p>smu:        SourceMeter Channel (A or B). Defaults to smua if all parameters are omitted when function is called.</p> <p>bias:        Bias level in volts.</p> <p>level:        Pulse level in volts.</p> <p>ton:        Pulse on-time in seconds.</p> <p>toff:        Pulse off-time in seconds.</p> <p>points:      Number of pulse-measure cycles.</p>
Data:	Pulsed current measurements, voltage levels and timestamps are stored in <code>smuX.nvbuffer1</code> .
Example:	<pre>PulseVMeasureI(smub, -1, 1, 1E-3, 2E-3, 20)</pre> <p>SMU B will output -1V and dwell for 1ms, output 1V and dwell for 2ms, and then perform a current measurement. After the measurement, the output will return to -1V and dwell for 2ms. This pulse-measure process will repeat 19 more times.</p>

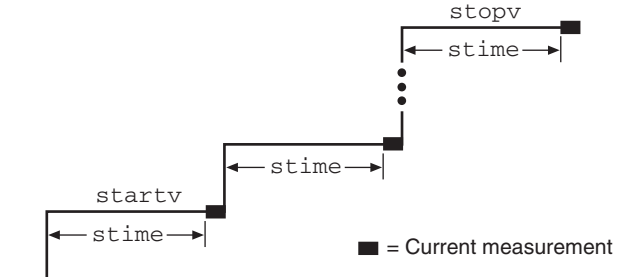
TSP Project Name:        KIFactoryGeneral TSP Test Script Name:    KIGeneral Version: 1.0.3RC1	
Function: <code>SweepILinMeasureV(smu, starti, stopi, stime, points)</code>	
Description:	<p>Performs a linear current sweep with voltage measured at every step (point):</p> <ul style="list-style-type: none"> <li>• Sets the <code>smu</code> to output <code>starti</code> amps, allows the source to settle for <code>stime</code> seconds and then performs a voltage measurement.</li> <li>• Sets the <code>smu</code> to output the next amps step, allows the source to settle for <code>stime</code> seconds and then performs a voltage measurement.</li> <li>• Repeats the above sequence until the voltage is measured on the <code>stopi</code> amps step.</li> </ul> <p>The linear step size is automatically calculated as follows:</p> $\text{step} = (\text{stopi} - \text{starti}) / (\text{points} - 1)$ 
Parameters:	<code>smu, starti, stopi, stime, points</code> <code>smu</code> :        SourceMeter Channel (A or B). Defaults to <code>smua</code> if all parameters are omitted when function is called. <code>starti</code> :     Sweep start current in amps. <code>stopi</code> :      Sweep stop current in amps. <code>stime</code> :      Settling time in seconds. Occurs after stepping the source and before performing a measurement. <code>points</code> :     Number of sweep points (must be $\geq 2$ ).
Data:	Voltage measurements, current source values and timestamps are stored in <code>smuX.nvbuffer1</code> .
Example:	<code>SweepILinMeasureV(smua, -1E-3, 1E-3, 0, 100)</code> This function performs a 100-point linear current sweep starting at -1mA and stopping at +1mA. Voltage is measured at every step (point) in the sweep. Since <code>stime</code> is set for 0s, voltage will be measured as fast as possible after each current step.

TSP Project Name:           KIFactoryGeneral TSP Test Script Name:      KIGeneral Version: 1.0.3RC1	
Function: SweepVLinMeasureI(smu, startv, stopv, stime, points)	
Description:	Performs a linear voltage sweep with current measured at every step (point): <ul style="list-style-type: none"> <li>• Sets the <code>smu</code> to output <code>startv</code> volts, allows the source to settle for <code>stime</code> seconds and then performs a current measurement.</li> <li>• Sets the <code>smu</code> to output the next volts step, allows the source to settle for <code>stime</code> seconds and then performs a current measurement.</li> <li>• Repeats the above sequence until the current is measured on the <code>stopv</code> volts step.</li> </ul> <p>The linear step size is automatically calculated as follows:</p> $\text{step} = (\text{stopv} - \text{startv}) / (\text{points} - 1)$ 
Parameters:	<code>smu, startv, stopv, stime, points</code> <code>smu</code> :       SourceMeter Channel (A or B). Defaults to <code>smua</code> if all parameters are omitted when function is called. <code>startv</code> :    Sweep start voltage in volts. <code>stopv</code> :     Sweep stop voltage in volts. <code>stime</code> :     Settling time in seconds. Occurs after stepping the source and before performing a measurement. <code>points</code> :    Number of sweep points (must be $\geq 2$ ).
Data:	Current measurements, voltage source values and timestamps are stored in <code>smuX.nvbuffer1</code> .
Example:	SweepVLinMeasureI(smua, -1, 1, 1E-3, 1000) This function performs a 1000-point linear voltage sweep starting at -1V and stopping at +1V. Current is measured at every step (point) in the sweep after a 1ms source settling period.



TSP Project Name:           KIFactoryGeneral TSP Test Script Name:       KIGeneral Version: 1.0.3RC1	
Function: SweepILogMeasureV(smu, starti, stopi, stime, points)	
Description:	<p>Performs a logarithmic current sweep with voltage measured at every step (point):</p> <ul style="list-style-type: none"> <li>• Sets the <code>smu</code> to output <code>starti</code> amps, allows the source to settle for <code>stime</code> seconds and then performs a voltage measurement.</li> <li>• Sets the <code>smu</code> to output the next amps step, allows the source to settle for <code>stime</code> seconds and then performs a voltage measurement.</li> <li>• Repeats the above sequence until the voltage is measured on the <code>stopi</code> amps step.</li> </ul> <p>The source level at each step (SourceStepLevel) is automatically calculated as follows:</p> <p><b>MeasurePoint</b> = The step point number for a measurement. For example, for a 5-point sweep (<code>points = 5</code>), a measurement will be performed at MeasurePoint 1, 2, 3, 4 and 5.</p> <p><b>LogStepSize</b> = <math>(\log_{10}(\text{stopi}) - \log_{10}(\text{starti})) / (\text{points} - 1)</math></p> <p><b>LogStep</b> = <math>(\text{MeasurePoint} - 1) \times (\text{LogStepSize})</math></p> <p><b>SourceStepLevel</b> = <math>\text{antilog}(\text{LogStep}) \times \text{starti}</math></p> 
Parameters:	<p><code>smu, starti, stopi, stime, points</code></p> <p><code>smu</code>:       SourceMeter Channel (A or B). Defaults to <code>smua</code> if all parameters are omitted when function is called.</p> <p><code>starti</code>:    Sweep start current in amps.</p> <p><code>stopi</code>:     Sweep stop current in amps.</p> <p><code>stime</code>:     Settling time in seconds. Occurs after stepping the source and before performing a measurement.</p> <p><code>points</code>:    Number of sweep measure points (must be <math>\geq 2</math>).</p>
Data:	<p>Voltage measurements, current source values and timestamps are stored in <code>smuX.nvbuffer1</code>.</p>

<b>Example:</b>	<code>SweepILogMeasureV(smua, 0.01, 0.1, 0.001, 5)</code>																								
	<p>This function performs a 5-point logarithmic current sweep starting at 10mA and stopping at 100mA. Voltage is measured at every step (point) in the sweep. The source will be allowed to settle on each step for 1ms before a measurement is performed.</p> <p>The following log values and corresponding source levels for the 5-point log sweep are listed as follows:</p>																								
	<table><thead><tr><th><b>MeasurePoint</b></th><th><b>LogStepSize</b></th><th><b>LogStep</b></th><th><b>SourceStepLevel</b></th></tr></thead><tbody><tr><td>1</td><td>0.25</td><td>0.0</td><td>0.01A</td></tr><tr><td>2</td><td>0.25</td><td>0.25</td><td>0.017783A</td></tr><tr><td>3</td><td>0.25</td><td>0.5</td><td>0.031623A</td></tr><tr><td>4</td><td>0.25</td><td>0.75</td><td>0.056234A</td></tr><tr><td>5</td><td>0.25</td><td>1.0</td><td>0.1A</td></tr></tbody></table>	<b>MeasurePoint</b>	<b>LogStepSize</b>	<b>LogStep</b>	<b>SourceStepLevel</b>	1	0.25	0.0	0.01A	2	0.25	0.25	0.017783A	3	0.25	0.5	0.031623A	4	0.25	0.75	0.056234A	5	0.25	1.0	0.1A
<b>MeasurePoint</b>	<b>LogStepSize</b>	<b>LogStep</b>	<b>SourceStepLevel</b>																						
1	0.25	0.0	0.01A																						
2	0.25	0.25	0.017783A																						
3	0.25	0.5	0.031623A																						
4	0.25	0.75	0.056234A																						
5	0.25	1.0	0.1A																						

TSP Project Name:           KIFactoryGeneral TSP Test Script Name:      KIGeneral Version: 1.0.3RC1	
Function: SweepVLogMeasureI(smu, startv, stopv, stime, points)	
Description:	<p>Performs a logarithmic voltage sweep with current measured at every step (point):</p> <ul style="list-style-type: none"> <li>• Sets the <i>smu</i> to output <i>startv</i> volts, allows the source to settle for <i>stime</i> seconds and then performs a current measurement.</li> <li>• Sets the <i>smu</i> to output the next volts step, allows the source to settle for <i>stime</i> seconds and then performs a current measurement.</li> <li>• Repeats the above sequence until the current is measured on the <i>stopi</i> amps step.</li> </ul> <p>The source level at each step (SourceStepLevel) is automatically calculated as follows:</p> <p><b>MeasurePoint</b> = The step point number for a measurement. For example, for a 5-point sweep (<i>points</i> = 5), a measurement will be performed at MeasurePoint 1, 2, 3, 4 and 5.</p> <p><b>LogStepSize</b> = <math>(\log_{10}(\text{stopi}) - \log_{10}(\text{starti})) / (\text{points} - 1)</math></p> <p><b>LogStep</b> = <math>(\text{MeasurePoint} - 1) \times (\text{LogStepSize})</math></p> <p><b>SourceStepLevel</b> = <math>\text{antilog}(\text{LogStep}) \times \text{starti}</math></p> 
Parameters:	<p><i>smu</i>, <i>startv</i>, <i>stopv</i>, <i>stime</i>, <i>points</i></p> <p><i>smu</i>:       SourceMeter Channel (A or B). Defaults to <i>smua</i> if all parameters are omitted when function is called.</p> <p><i>starti</i>:    Sweep start voltage in amps.</p> <p><i>stopi</i>:     Sweep stop voltage in amps.</p> <p><i>stime</i>:     Settling time in seconds. Occurs after stepping the source and before performing a measurement.</p> <p><i>points</i>:    Number of sweep measure points (must be <math>\geq 2</math>).</p>
Data:	Current measurements, voltage source values and timestamps are stored in <i>smuX.nvbuffer1</i> .

**Example:** `SweepVLogMeasureI(smua, 1, 10, 0.001, 5)`

This function performs a 5-point logarithmic voltage sweep starting at 1V and stopping at 10V. Current is measured at every step (point) in the sweep. The source will be allowed to settle on each step for 1ms before a measurement is performed.

The following log values and corresponding source levels for the 5-point log sweep are listed as follows:

<b>MeasurePoint</b>	<b>LogStepSize</b>	<b>LogStep</b>	<b>SourceStepLevel</b>
1	0.25	0.0	1.0000V
2	0.25	0.25	1.7783V
3	0.25	0.5	3.1623V
4	0.25	0.75	5.6234V
5	0.25	1.0	10.000V

TSP Project Name:            KIFactoryGeneral	
TSP Test Script Name:        KIGeneral	
Version: 1.0.3RC1	
Function: SweepIListMeasureV(smu, ildist, stime, points)	
Description:	<p>Performs a current list sweep with voltage measured at every step (point):</p> <ul style="list-style-type: none"> <li>• Sets the <code>smu</code> to output the first <code>ildist</code> amps value, allows the source to settle for <code>stime</code> seconds and then performs a voltage measurement.</li> <li>• Sets the <code>smu</code> to output the next <code>ildist</code> amps value, allows the source to settle for <code>stime</code> seconds and then performs a voltage measurement.</li> <li>• Repeats the above sequence until the voltage is measured for the last amps value. The last point in the list to be measured is <code>points</code>.</li> </ul>
Parameters:	<p><code>smu, ildist, stime, points</code></p> <p><code>smu</code>:        SourceMeter Channel (A or B). Defaults to <code>smua</code> if all parameters are omitted when function is called.</p> <p><code>ildist</code>:     Arbitrary list of current source values:               <code>ildist = {value1, value2, ...valueN}</code></p> <p><code>stime</code>:     Settling time in seconds. Occurs after sourcing a value and before performing a measurement.</p> <p><code>points</code>:    Number of sweep measure points (must be <math>\geq 2</math>).</p>
Data:	Voltage measurements, current source values and timestamps are stored in <code>smuX.nvbuffer1</code> .
Example:	<pre>myildist = {-100E-9, 100E-9, -1E-6, 1E-6, -1E-3, 1E-3} SweepIListMeasureV(smua, myildist, 500E-6, 6)</pre> <p>This function performs a 6-point current list sweep starting at the first point in <code>myildist</code>. Voltage is measured at every step (point) in the sweep. The source will be allowed to settle on each value for 500<math>\mu</math>s before a measurement is performed.</p>

TSP Project Name:            KIFactoryGeneral	
TSP Test Script Name:        KIGeneral	
Version: 1.0.3RC1	
Function: SweepVListMeasureI(smu, vlist, stime, points)	
Description:	<p>Performs a voltage list sweep with current measured at every step (point):</p> <ul style="list-style-type: none"> <li>• Sets the <code>smu</code> to output the first <code>vlist</code> volts value, allows the source to settle for <code>stime</code> seconds and then performs a current measurement.</li> <li>• Sets the <code>smu</code> to output the next <code>vlist</code> volts value, allows the source to settle for <code>stime</code> seconds and then performs a current measurement.</li> <li>• Repeats the above sequence until the current is measured for the last volts value. The last point in the list to be measured is <code>points</code>.</li> </ul>
Parameters:	<p><code>smu, vlist, stime, points</code></p> <p><code>smu</code>:        SourceMeter Channel (A or B). Defaults to <code>smua</code> if all parameters are omitted when function is called.</p> <p><code>vlist</code>:      Arbitrary list of voltage source values:               <code>vlist = {value1, value2, ...valueN}</code></p> <p><code>stime</code>:     Settling time in seconds. Occurs after sourcing a value and before performing a measurement.</p> <p><code>points</code>:    Number of sweep measure points (must be <math>\geq 2</math>).</p>
Data:	Current measurements, voltage source values and timestamps are stored in <code>smuX.nvbuffer1</code> .
Example:	<pre>myvlist = {-0.1, 0.1, -1, 1, -6, 6, -40, 40, 0, 0} SweepVListMeasureI(smua, myvlist, 500E-6, 10)</pre> <p>This function performs a 10-point voltage list sweep starting at the first point in <code>myvlist</code>. Current is measured at every step (point) in the sweep. The source will be allowed to settle on each value for 500<math>\mu</math>s before a measurement is performed.</p>

# Flash firmware upgrade

As Keithley develops additional factory scripts, they will be made available on the Keithley web site ([www.keithley.com](http://www.keithley.com)) as a flash firmware upgrade for the Model 260x.

**CAUTION** External circuitry connected to input/output terminals while attempting a flash upgrade may cause instrument and/or DUT damage. Disconnect input/output terminals before performing a flash upgrade.

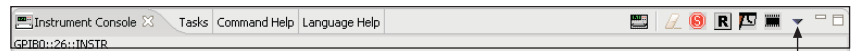
After downloading the new flash file from the Keithley website, use the Test Script Builder (TSB) to upgrade the firmware of your Model 260x:

1. On the PC desktop, double-click the icon for the Test Script Builder.
2. On the Instrument Console toolbar, click the Open Instrument icon and then select your communication interface from the Select Instrument Resource dialog box. Details on opening communications are provided in [Figure 2-6](#).



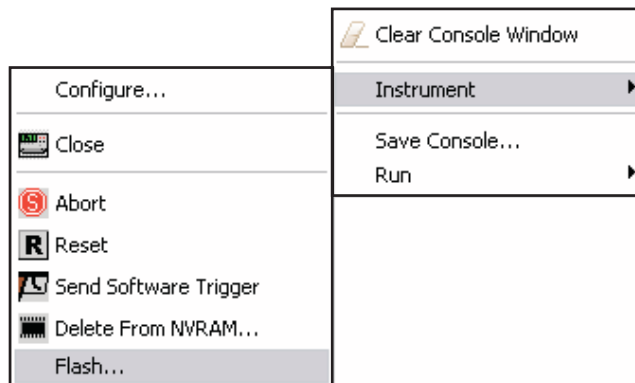
Open Instrument icon

3. On the Instrument Console toolbar, click the Menu icon to display the menu.



Menu icon

4. From the drop-down menu, select Instrument and then click Flash.



5. From the Select A Firmware Data File dialog box, use the browser to select the File name of the new firmware and click Open to upgrade the firmware of the Model 260x.

# 14

## Display Operations

(Display functions & attributes)

---

### Section 14 topics

**Display functions and attributes**, page 14-2

**Display features**, page 14-3

Display screen, page 14-3

Measurement functions, page 14-3

Display resolution, page 14-4

**Display messages**, page 14-4

Clearing the display, page 14-5

Cursor position, page 14-5

Displaying text messages, page 14-6

**Input prompting**, page 14-8

Menu, page 14-8

Parameter value prompting, page 14-10

**Annunciators**, page 14-11

**LOCAL lockout**, page 14-12

**Load test menu**, page 14-13

Saving a user script, page 14-13

Adding USER TESTS menu entries, page 14-13

Deleting USER TESTS menu entries, page 14-14

Running a test from the front panel, page 14-14

**Display triggering**, page 14-15

**Key-press codes**, page 14-16

Sending keycodes, page 14-16

Capturing key-press codes, page 14-17



## Display functions and attributes

The display functions and attributes are used to perform the display operations covered in this section. [Table14-1](#) lists each display function/attribute (in alphabetical order) and cross references it to the section topic where the function/attribute is explained.

[Section 12](#) provides additional information on the display functions and attributes.

Table14-1

### Cross referencing functions/attributes to section topics

Function/Attribute	Section Topic
<code>display.clear</code>	<a href="#">Clearing the display, page 14-5</a>
<code>display.getannunciators</code>	<a href="#">Annunciators, page 14-11</a>
<code>display.getcursor</code>	<a href="#">Cursor position, page 14-5</a>
<code>display.gettext</code>	<a href="#">Displaying text messages, page 14-6</a>
<code>display.input</code>	<a href="#">Capturing key-press codes, page 14-17</a>
<code>display.inputvalue</code>	<a href="#">Parameter value prompting, page 14-10</a>
<code>display.loadmenu.add</code> <code>display.loadmenu.delete</code>	<a href="#">Load test menu, page 14-13</a>
<code>display.locallockout</code>	<a href="#">LOCAL lockout, page 14-12</a>
<code>display.menu</code>	<a href="#">Menu, page 14-8</a>
<code>display.prompt</code>	<a href="#">Parameter value prompting, page 14-10</a>
<code>display.screen</code>	<a href="#">Display screen, page 14-3</a>
<code>display.sendkey</code>	<a href="#">Sending keycodes, page 14-16</a>
<code>display.setcursor</code>	<a href="#">Cursor position, page 14-5</a>
<code>display.settext</code>	<a href="#">Displaying text messages, page 14-6</a>
<code>display.smuX.digits</code>	<a href="#">Display resolution, page 14-4</a>
<code>display.smuX.measure.func</code>	<a href="#">Measurement functions, page 14-3</a>
<code>display.trigger.clear</code> <code>display.trigger.wait</code>	<a href="#">Display triggering, page 14-15</a>

# Display features

## Display screen

The Model 260x can display source-measure values/readings or user defined messages. The display screen options include the following:

- Source-measure, compliance screens:
  - Display source and compliance values, and measure readings for SMU A.
  - Display source and compliance values, and measure readings for SMU B.
- Source-measure screen – Display source values and measure readings for SMU A and SMU B.
- User screen – Display user-defined messages and prompts.

The `display.screen` attribute is used to select the display screen:

```
display.screen = displayId
```

where: `displayId` is set to one of the following values or names:

- 0 or `display.SMUA`
- 1 or `display.SMUB`
- 2 or `display.SMUA_SMUB`
- 3 or `display.USER`

### Display screen example:

The following command displays source-measure and compliance for SMU A:

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

## Measurement functions

With a source-measure screen selected, the measured reading can be displayed as volts, amps, ohms or watts.

The `display.smuX.measure.func` attribute is used to select the displayed measure function:

```
display.smuX.measure.func = function
```

where: `smuX` = `sma` or `smub`

`function` is set to one of the following values:

- 0 or `display.MEASURE_DCAMPS`
- 1 or `display.MEASURE_DCVOLTS`
- 2 or `display.MEASURE_OHMS`
- 3 or `display.MEASURE_WATTS`

### Measurement function example:

The following command sets SMU A to display ohms measurements:

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

## Display resolution

Display resolution for measured readings can be set to 4-1/2, 5-1/2 or 6-1/2 digit resolution.

The `display.smuXdigits` attribute is used to set display resolution for measured readings:

```
display.smuX.digits = digits
```

Where: `smuX` = `smua` or `smub`

`digits` is set to one of the following values:

4 or `display.DIGITS_4_5`

5 or `display.DIGITS_5_5`

6 or `display.DIGITS_6_5`

### Display resolution example:

The following command sets SMU A for 5-1/2 digit resolution for measured readings:

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

## Display messages

<b>NOTE</b>	<p>Most of the display functions and attributes that are associated with display messaging will automatically select the user screen. The attribute for the display screen is explained in “<a href="#">Display screen</a>” on <a href="#">page 14-3</a>.</p> <p>The reset functions (<code>reset</code> or <code>smuX.reset</code>) have no effect on the defined display message or its configuration, but will set the display mode back to the previous source-measure display mode.</p>
-------------	--

The display of the Model 260x can be used to display user-defined messages. For example, while a test is running, the following message can be displayed on the Model 260x.

### Test in Process

#### Do Not Disturb

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

**NOTE** The `display.clear`, `display.setcursor`, and `display.settext` functions (which are explained in the following paragraphs) are overlapped, non-blocking commands. The script will NOT wait for one of these commands to complete.

These non-blocking functions do not immediately update the display. For performance considerations, they write to a shadow and will update the display as soon as processing time becomes available.

## Clearing the display

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

After displaying an input prompt, the message will remain displayed even after the operator performs the prescribed action. The `clear` function must be sent to clear the display.

The following command clears both lines of the display, but does not affect any of the annunciators:

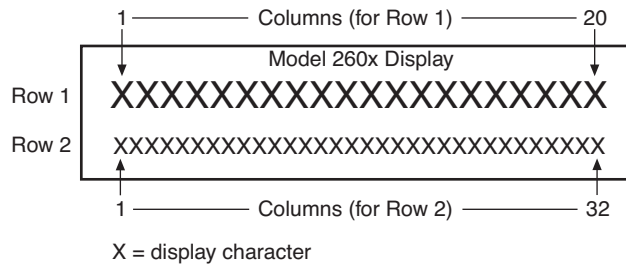
```
display.clear()
```

## Cursor position

When displaying a message, the cursor position determines where the message will start. On power-up, the cursor is positioned at Row 1, Column 1 (see [Figure 14-1](#)). At this cursor position, a user-defined message will be displayed on the top row (Row 1).

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

Figure 14-1

**Row/column format for display messaging**

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

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

where: row = 1 or 2  
 column = 1 to 20 (Row 1)  
           = 1 to 32 (Row 2)  
 style = 0 (invisible)  
        = 1 (blink)

When set to 0, the cursor will not be seen. When set to 1, a display character will blink to indicate its position.

The `display.getcursor` function returns the present cursor position, and can be used three ways:

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

**Example:** The following code positions the cursor on Row 2, Column 1, and then reads the cursor position:

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

Output: 2.000000e+00 1.000000e+00

## Displaying text messages

The `display.settext` function is used to define and display a message. The message will start at the present cursor position.

```
display.settext(text)
```

where: `text` is the text string to be displayed.

**Example:** The following code will display “Test in Process” on the top line, and “Do Not Disturb” on the bottom line:

```
display.clear()
```

```
display.setCursor(1, 1, 0)
display.setText("Test in Process")
display.setCursor(2, 6, 0)
display.setText("Do Not Disturb")
```

## Character codes

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

<code>\$N</code>	Newline – Starts text on the next line. If the cursor is already on line 2, text will be ignored after the ‘\$N’ is received.
<code>\$R</code>	Sets text to Normal.
<code>\$B</code>	Sets text to Blink.
<code>\$D</code>	Sets text to Dim intensity.
<code>\$F</code>	Set text to background blink.
<code>\$\$</code>	Escape sequence to display a single “\$”.

### Examples:

The following code uses the `$N` and `#B` character codes to display the message “Test in Process” on the top line and the blinking message “Do Not Disturb” on the bottom line:

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

The following code uses the `$$` character code to display the message “You owe me \$8” on the top line:

```
display.clear()
display.setCursor(1, 1)
display.setText("You owe me $$8")
```

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

#### NOTE

Care must be taken when imbedding character codes in the `text` string. It is easy to forget that the character following the `$` is part of the code. For example, assume you want to display “Hello” on the top line and “Nate” on the bottom line, and so you send the following command:

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

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

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

## Returning a text message

The `display.gettext` function returns the displayed message and can be used in five ways:

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, column start)
text = display.gettext(embellished, row, column start, column end)
```

`embellished` Set to `false` to return text as a simple character string. Set to `true` to include character codes.

`row` Set to 1 or 2 to select which row to read text from. If not included, text from both rows are read.

`column start` Set to starting column for reading text.

`column end` Set to ending column for reading text.

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

With `embellished` set to `true`, all other character codes that were used in the creation of each message line will be returned along with the message. With `embellished` set to `false`, only the message will be returned.

Sending the command without the `column start` parameter defaults to Column 1. Sending the command without the `column end` argument defaults to the last column (Column 20 for Row 1, Column 32 for Row 2).

## Input prompting

Display messaging can also be used along with front panel controls to make a user script interactive. For an interactive script, input prompts are displayed so that the operator can perform a prescribed action using the front panel controls. While displaying an input prompt, the test will pause and wait for the operator to perform the prescribed action from the front panel.

### Menu

A user-defined menu can be presented on the display. The menu consists of the menu name on the top line, and a selectable list of menu items on the bottom line. The following function is used to define a menu:

```
display.menu(menu, items)
```

where: `menu` is the name of the menu (string up to 20 characters, including spaces). The `items` string is made up of one or more menu items, where each item must be separated by whitespace.

When the `display.menu` function is executed, script execution will wait for the operator to select one of the menu items. Rotate the Wheel to place the blinking cursor on the desired menu item. Items that don't fit in the display area will be displayed by rotating the wheel to the right. With the cursor on the desired menu item, press the Rotary Wheel (or the Enter key) to select it.

Pressing the EXIT key will not abort the script while the menu is displayed, but it will return `nil`. The script can be aborted by calling the `exit` function when `nil` is returned.

Example: The menu for the following code will present the operator with the choice of two menu items; Test1 or Test2. If Test1 is selected, the message "Running Test1" will be displayed. If Test2 is selected, the message "Running Test2" will be displayed.

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



## Parameter value prompting

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

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

Each of these two functions can be used in four ways:

```
display.inputvalue(format)
display.inputvalue(format, default)
display.inputvalue(format, default, min)
display.inputvalue(format, default, min, max)

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

`format` – The `format` string creates an editable input field on the user screen at the present cursor position. Examples of the `format` for an input field:

```
+0.00 00 +00.0000E +00 0.00000E+0
```

Value field:

- + Include a “+” sign for positive/negative value entry. Not including the “+” sign prevents negative value entry.
- 0’s’ Defines the digit positions for the value. Up to six ‘0’s can be used for the value (as shown above in the third and fourth examples).
- .
- If used, include the decimal point (.) where needed for the value.

Exponent field (optional):

- E Include the “E” for exponent entry.
- + Include a “+” sign for positive/negative exponent entry. Not including the “+” sign prevents negative exponent entry.
- 0’s’ Defines the digit positions for the exponent.

`default` – Use this option to set a default value for the parameter. The `default` value will be displayed when the command is sent.

`min` and `max` – There are options to specify minimum and maximum limits for the input field. When NOT using the “+” sign for the value field, the minimum limit cannot be set to less than zero. When using the “+” sign, the minimum limit can be set to less than zero (e.g., -2).

`units` and `help` – `units` is a text string to identify the units for the value (8 characters maximum). Example units text is “V” for volts and “A” or amps. `help` is an information text string to display on the bottom line (32 characters maximum).

The two functions are similar in that they both display the editable input field, but the `display.inputvalue` function does not include the text strings for `units` and `help`.

After one of the above functions is executed, script execution will pause and wait for the operator to input the source level. The program will continue after the operator enters the value by pressing the Rotary Wheel or the Enter key.

### Examples:

The following code will prompt the operator to enter a source value:

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

The script will pause after displaying the prompt message and wait for the operator to enter the voltage level. The display will then toggle to the source-measure display for SMU A and set the source level to `value`.

<b>NOTE</b> <b>If the operator presses EXIT instead of entering a source value, <code>value</code> will be set to <code>nil</code>.</b>
---

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

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

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

## Annunciators

Send the following code to determine which display annunciators are turned on:

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

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

[Table14-2](#) identifies the bit position for each annunciator. The table also includes the weighted value of each bit. The returned value is the sum of all the weighted values for the bits that are set.

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

1000010100001101

For the above binary number, the following bits are set to “1”: 16, 11, 9, 4, 3 and 1. Using [Table14-2](#) , the following annunciators are on: REL, REM, EDIT, AUTO, 4W and EDIT.

Table14-2

**Bit identification for annunciators**

Bit	16	15	14	13	12	11	10	9
Annunciator	REL	REAR	SRQ	LSTN	TALK	REM	ERR	EDIT
Weighted Value*	32768	16384	8192	4096	2048	1024	512	256
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Bit	8	7	6	5	4	3	2	1
Annunciator	SMPL	STAR	TRIG	ARM	AUTO	4W	MATH	FILT
Weighted Value*	128	64	32	16	8	4	2	1
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

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

Note that not all of the above annunciators shown in [Table14-2](#) are used by the Model 260x.

## LOCAL lockout

The front panel LOCAL key is used to cancel remote operation and return control to the front panel. However, the LOCAL key can be locked out to prevent a test from being interrupted. When locked, the LOCAL key becomes a NO-OP (no operation). Use the following attribute to lock or unlock the LOCAL key:

```
display.locallockout = lockout
```

where lockout is set to one of the following values:

0 or display.UNLOCK

1 or display.LOCK

**LOCAL lockout example:**

The following command locks out the LOCAL key:

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

## Load test menu

The LOAD TEST menu lists script tests (USER and FACTORY) that can be run from the front panel. Factory script tests (functions) are pre-loaded and saved in non-volatile memory at the factory. They are available in the FACTORY TESTS submenu.

After a user script is loaded into the Model 260x, it is not automatically added to the front panel USER TESTS submenu. A menu name and a chunk is added by the user.

### Saving a user script

After a user script is loaded into the Model 260x it can be saved in non-volatile memory. If it is not stored in non-volatile memory, the script will be lost when the Model 260x is turned off.

When loading a script from the Test Script Builder, the launch can be configured to save the script in non-volatile memory (see [“Using Test Script Builder”](#) on [page 2-12](#)).

When loading a user script from another program, `myscript.save()` is used to save the script in non-volatile memory (see [“Saving a user script”](#) on [page 2-43](#)).

### Adding USER TESTS menu entries

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

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

<code>displayname</code>	Name string to add to the menu.
<code>chunk</code>	Chunk is the code to be executed.
<code>memory</code>	Save or don't save <code>chunk</code> and <code>displayname</code> in non-volatile memory.

Set `memory` to one of the following values:

0 or `display.DONT_SAVE`

1 or `display.SAVE`

The default `memory` setting is `display.SAVE`.

The `chunk` can be made up of scripts, functions, variables and commands. With `memory` set to `display.SAVE`, commands are saved with the `chunk` in non-volatile memory. Scripts, functions and variables used in the `chunk` are not saved by `display.SAVE`. Functions and variables need to be saved along with the script (see [“Saving a user script”](#)). If the script is not saved in non-volatile memory, it will be lost when the Model 260x is turned off. See **Example 1** below.

**Example 1:**

Assume a script with a function named “DUT1” has already been loaded into the Model 260x, and the script has NOT been saved in non-volatile memory.

Now assume you want to add a test named “Test” to the USER TESTS menu. You want the test to run the function named “DUT1” and sound the beeper. The following command will add “Test” to the menu, define the chunk, and then save `displayname` and chunk in non-volatile memory:

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

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

Now assume you cycle power on the Model 260x. Since the script was not saved in non-volatile memory, the function named “DUT1” is lost. When “Test” is again run from the front panel, the beeper will beep, but “DUT1” will not execute because it no longer exists in the chunk.

**Example 2:**

The following command adds an entry called “Part1” to the front panel “USER TESTS” submenu for the chunk “testpart([[Part1]], 5.0)”, and saves it in non-volatile memory:

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

## Deleting USER TESTS menu entries

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

```
display.loadmenu.delete(displayname)  
displayname Name to delete from the menu.
```

**Example:**

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

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

## Running a test from the front panel

Front panel user tests and factory tests can be run as follows:

1. Press the LOAD key to display the LOAD TEST menu.
2. Select the USER or FACTORY menu item.
3. Position the blinking cursor on the test to be run.
4. Press the RUN key to run the test.

# Display triggering

Script execution can be held up for a specified period of time and wait for the operator to press the **TRIG** key to continue. The script will continue when the **TRIG** key is pressed, or the specified wait period has expired.

The following two functions are used for display triggering:

```
display.trigger.wait(timeout)
display.trigger.clear()
```

where: `timeout` = timeout period in seconds

Pressing the **TRIG** key sets the front panel trigger detector. The detector has to be cleared before you can again effectively use the `display.trigger.wait` function. If you send the trigger wait call while the detector is set, the trigger will be detected immediately and the script will continue. Send the `display.trigger.clear` function to clear the trigger detector.

The trigger wait function can be read to determine if trigger detection was caused by the **TRIG** key or the timeout:

```
triggered = display.trigger.wait(timeout)
print(triggered)
```

Output: 1.000000e+00 (if **TRIG** key was pressed)  
0.000000e+00 (if operation timed out)

## Display triggering example:

The following code will pause a script and prompt the operator to press the **TRIG** key when ready to continue. If the **TRIG** key is not pressed, the test will continue after waiting 10 minutes (600s).

```
display.clear()
display.setcursor(1, 1, 0)
display.settext("Take a Break")
display.setcursor(2, 1, 0)
display.settext("Press TRIG to continue")
display.trigger.wait(600)
display.trigger.clear()
display.clear()
```

After trigger detection is satisfied (**TRIG** key pressed or timeout occurred), the trigger detector and the display will clear.

# Key-press codes

## Sending keycodes

Keycodes are provided to remotely “press” a front key or the Rotary Knob. There are also keycodes to “rotate” the Knob to the left or right (one click at a time). Use the `display.sendkey` function to perform these actions:

```
display.sendkey(keycode)
```

where: `keycode` is the value of the front panel control. The keycode for each control is listed alphabetically in [Table14-3](#).

### Key press example:

Either of the following commands will press the **MENU** key:

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

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

Table14-3

### Keycodes to send for `display.sendkey`

<code>display.KEY_AUTO</code>	or	73	<code>display.KEY_OUTPUTA</code>	or	88
<code>display.KEY_CONFIG</code>	or	80	<code>display.KEY_OUTPUTB</code>	or	96
<code>display.KEY_DIGITSA</code>	or	87	<code>display.KEY_RANGEDOWN</code>	or	81
<code>display.KEY_DIGITSB</code>	or	84	<code>display.KEY_RANGEUP</code>	or	65
<code>display.KEY_DISPLAY</code>	or	72	<code>display.KEY_RECALL</code>	or	85
<code>display.KEY_ENTER</code>	or	82	<code>display.KEY_RELA</code>	or	70
<code>display.KEY_EXIT</code>	or	75	<code>display.KEY_RELB</code>	or	67
<code>display.KEY_FILTERA</code>	or	77	<code>display.KEY_RIGHT</code>	or	103
<code>display.KEY_FILTERB</code>	or	74	<code>display.KEY_RUN</code>	or	71
<code>display.KEY_LEFT</code>	or	104	<code>display.KEY_SPEEDA</code>	or	94
<code>display.KEY_LIMITA</code>	or	93	<code>display.KEY_SPEEDB</code>	or	91
<code>display.KEY_LIMITB</code>	or	90	<code>display.KEY_SRCA</code>	or	79
<code>display.KEY_LOAD</code>	or	95	<code>display.KEY_SRCB</code>	or	76
<code>display.KEY_MEASA</code>	or	86	<code>display.KEY_STORE</code>	or	78
<code>display.KEY_MEASB</code>	or	83	<code>display.KEY_TRIG</code>	or	92
<code>display.KEY_MENU</code>	or	68	<code>display.WHEEL_ENTER</code>	or	97
<code>display.KEY_MODEA</code>	or	69	<code>display.WHEEL_LEFT</code>	or	107
<code>display.KEY_MODEB</code>	or	66	<code>display.WHEEL_RIGHT</code>	or	114

## Capturing key-press codes

A history of the keycode for the last pressed front panel key is maintained by the Model 260x. When the instrument is powered-on, (or when transitioning from local to remote), the keycode is set to 0 (`display.KEY_NONE`).

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

### `display.getlastkey`

The `display.getlastkey` function is used to immediately return the keycode for the last pressed key:

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

The above code will return the keycode value (see [Table14-4](#)). Keep in mind that a value of 0 (`display.KEY_NONE`) indicates that the keycode history had been cleared.

Table14-4

#### Keycode values returned for `display.getlastkey`

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

**NOTE** The OUTPUT ON/OFF keys for SMU A and SMU B cannot be tracked by this function.



### **display.waitkey**

The `display.waitkey` function captures the keycode value for the next key press:

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

After sending the `display.waitkey` function, the script will pause and wait for the operator to press a front panel key. For example, if the **MEAS** key is pressed, the function will return the value 86, which is the keycode for that key. The keycode values are listed in [Table14-3](#) .

**Example:** The following code will prompt the user to press the **EXIT** key to abort the script, or any other key to continue it:

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

The above code captures the key that is pressed by the operator. The keycode value for the **EXIT** key is 75. If **EXIT** is pressed, the script aborts. If any other key is pressed, the script will continue.

# 15

# Performance Verification

---

## **Section 15 topics**

**Introduction**, page 15-2

**Verification test requirements**, page 15-2

Environmental conditions, page 15-2

Warm-up period, page 15-3

Line power, page 15-3

Recommended test equipment, page 15-3

Verification limits, page 15-4

**Restoring factory defaults**, page 15-5

**Performing the verification test procedures**,  
page 15-5

Test summary, page 15-5

Test considerations, page 15-5

Setting the source range and output value,  
page 15-6

Setting the measurement range, page 15-6

**Output voltage accuracy**, page 15-7

**Voltage measurement accuracy**, page 15-8

**Output current accuracy**, page 15-9

**Current measurement accuracy**, page 15-11

# Introduction

Use the procedures in this section to verify that the Model 260x accuracy is within the limits stated in the instrument's one-year accuracy specifications. Perform the verification procedures:

- When you first receive the instrument to make sure that it was not damaged during shipment.
- To verify that the unit meets factory specifications.
- To determine if calibration is required.
- Following calibration to make sure it was performed properly.

**WARNING** The information in this section is intended for qualified service personnel only. Do not attempt these procedures unless you are qualified to do so. Some of these procedures may expose you to hazardous voltages, which could cause personal injury or death if contacted. Use appropriate safety precautions when working with hazardous voltages.

**NOTE** If the instrument is still under warranty and its performance is outside specified limits, contact your Keithley representative or the factory to determine the correct course of action.

## Verification test requirements

Be sure that you perform the verification tests:

- Under the proper environmental conditions.
- After the specified warm-up period.
- Using the correct line voltage.
- Using the proper test equipment.
- Using the specified output signal and reading limits.

### Environmental conditions

Conduct your performance verification procedures in a test environment with:

- An ambient temperature of 18-28°C (65-82°F).
- A relative humidity of less than 70% unless otherwise noted.

## Warm-up period

Allow the Model 260x to warm up for at least two hours before conducting the verification procedures. If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the instrument's internal temperature to stabilize. Typically, allow one extra hour to stabilize a unit that is 10°C (18°F) outside the specified temperature range.

Also, allow the test equipment to warm up for the minimum time specified by the manufacturer.

## Line power

The Model 260x requires a line voltage of 100V to 240V and a line frequency of 50Hz or 60Hz. Verification tests should be performed within this range.

## Recommended test equipment

[Table 15-1](#) summarizes recommended verification equipment. You can use alternate equipment as long as that equipment has specifications equal or greater than those listed in [Table 15-1](#). Keep in mind, however, that test equipment uncertainty will add to the uncertainty of each measurement. Generally, test equipment uncertainty should be at least four times better than corresponding Model 260x specifications. [Table 15-1](#) lists the uncertainties of the recommended test equipment.

Table 15-1  
**Recommended verification equipment**

Description	Manufacturer/ Model	Accuracy
Digital Multimeter	Agilent 3458A	DC Voltage <sup>1</sup> 90mV: ±8ppm 0.9V: ±5ppm 5.4V: ±8ppm 36V: ±11ppm  DC current <sup>1</sup> 90nA: ±430ppm 0.9µA: ±55ppm 9µA: ±25ppm 90µA: ±23ppm 0.9mA: ±20ppm 9mA: ±20ppm 90mA: ±35ppm 0.9A: ±110ppm
0.5Ω, 250W, 0.1% Precision Resistor	Isotek RUG-Z-R500-0.1-TK3	Resistance <sup>2</sup> 0.5Ω: ±125ppm

1. 90-day specifications show accuracy of recommended model at specified measurement point.

2. Resistor used to test 3A range only should be characterized to uncertainty shown using resistance function of digital multimeter before use.

## Verification limits

The verification limits stated in this section have been calculated using only the Model 260x one-year accuracy specifications, and they do not include test equipment uncertainty. If a particular measurement falls outside the allowable range, recalculate new limits based both on the Model 260x specifications and corresponding test equipment specifications.

### Example limits calculation

As an example of how verification limits are calculated, assume you are testing the 6V DC output range using a 5.4V output value. Using the Model 260x one-year accuracy specification for 5.4V DC output of ± (0.02% of output + 1.8mV offset), the calculated output limits are:

$$\text{Output limits} = 5.4V \pm [(5.4V \times 0.02\%) + 1.8mV]$$

$$\text{Output limits} = 5.4V \pm (0.00108 + 0.0018)$$

$$\text{Output limits} = 5.4V \pm 0.00288V$$

$$\text{Output limits} = 5.39712V \text{ to } 5.40288V$$

## Restoring factory defaults

Before performing the verification procedures, restore the instrument to its factory front panel (bench) defaults as follows:

1. Press the MENU key. The instrument will display the following prompt:  
MAIN MENU  
SAVE-SETUP COMMUNICATION TEST
2. Select SAVE-SETUP, and then press ENTER. The unit then displays:  
SAVE SETUP MENU  
SAVE RECALL POWERON
3. Select RECALL, and then press ENTER. The unit displays:  
RECALL SETUP  
FACTORY USER-1 USER-2 USER-3 USER-4 USER-5
4. Select FACTORY, then press ENTER to restore defaults.

## Performing the verification test procedures

### Test summary

- DC voltage output accuracy
- DC voltage measurement accuracy
- DC current output accuracy
- DC current measurement accuracy

If the Model 260x is not within specifications and not under warranty, see the calibration procedures in [Section 16](#) for information on calibrating the unit.

### Test considerations

When performing the verification procedures:

- Be sure to restore factory front panel defaults as outlined above.
- Make sure that the test equipment is properly warmed up and connected to the Model 260x output terminals (use 4-wire sensing for voltage).
- Make sure the Model 260x is set to the correct source range.
- Be sure the Model 260x output is turned on before making measurements.
- Be sure the test equipment is set up for the proper function and range.
- Allow the Model 260x output signal to settle before making a measurement.
- Do not connect test equipment to the Model 260x through a scanner, multiplexer, or other switching equipment.

**WARNING** The maximum common-mode voltage (voltage between LO and chassis ground) is 250VDC. Exceeding this value may cause a breakdown in insulation, creating a shock hazard.

The terminals of the Model 260x are rated for connection to circuits rated Installation Category I only. Do not connect the Model 260x terminals to CAT II, CAT III, or CAT IV circuits. Connection of the SourceMeter terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, **NEVER** make or break connections to the Model 260x while the unit is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 260x before handling cables connected to the outputs. Putting the equipment into standby mode does not guarantee the outputs are not powered if a hardware or software fault occurs.

## Setting the source range and output value

Before testing each verification point, you must properly set the source range and output value as outlined below.

1. Press the SRC key to select the appropriate source function.
2. Press the Rotary Knob or ENTER to enable the edit mode (EDIT annunciator on).
3. When the cursor in the source display field is flashing, set the source range to the lowest possible range for the value being sourced. Use the up or down RANGE key to set the value. For example, you should use the 6V source range to output a 5.4V source value. With a 5.4V source value and the 6V range selected, the Model 2602 Channel A source field display will appear as follows:  
SrcA:+5.40000 V
4. Use the Rotary Knob and CURSOR keys to set the source value to the required value, then press ENTER or the Rotary Knob to complete editing.

## Setting the measurement range

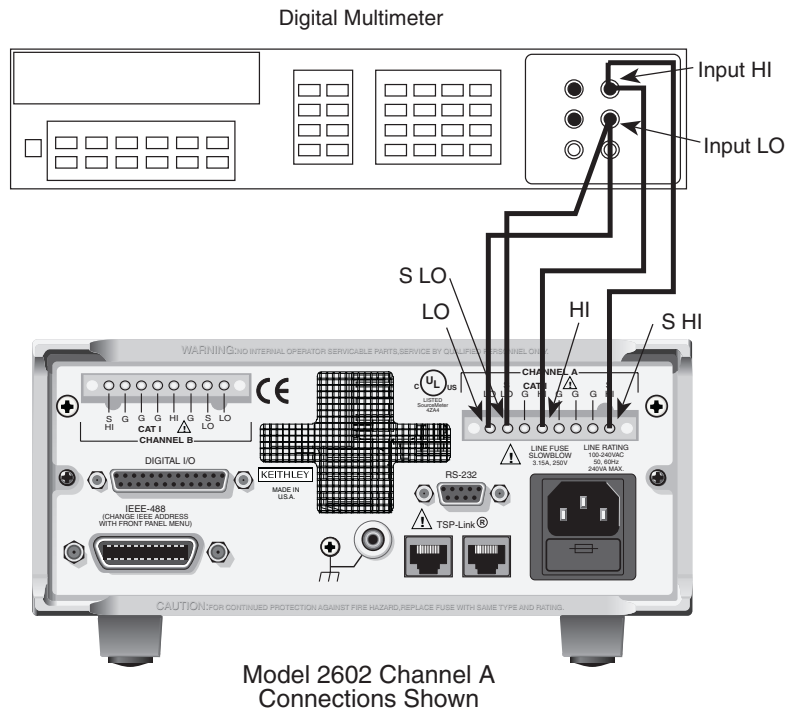
When simultaneously sourcing and measuring either voltage or current, the measure range is coupled to the source range, and you cannot independently control the measure range. Thus, it is not necessary for you to set the range when testing voltage or current measurement accuracy.

# Output voltage accuracy

Follow the steps below to verify that the Model 260x output voltage accuracy is within specified limits. This test involves setting the output voltage to each full-range value and measuring the voltages with a precision digital multimeter.

1. With the power off, connect the digital multimeter to the Model 260x output terminals using 4-wire connections, as shown in [Figure 15-1](#).
2. Select the multimeter DC volts measuring function.
3. Select the Model 2602 single-channel display mode. Press the Model 260x SRC key to source voltage, and make sure the source output is turned on.
4. Enable the Model 260x 4-wire (remote sense) mode by pressing CONFIG then SRC, then select V-SOURCE > SENSE-MODE > 4-WIRE.
5. Verify output voltage accuracy for each of the voltages listed in [Table 15-2](#). For each test point:
  - Select the correct source range.
  - Set the Model 260x output voltage to the indicated value.
  - Verify that the multimeter reading is within the limits given in the table.

Figure 15-1  
Connections for voltage verification





6. Repeat the procedure for negative output voltages with the same magnitudes as those listed in [Table 15-2](#).
7. For the Model 2602, repeat the above procedure for the other channel.

Table 15-2

**Output voltage accuracy limits**

Model 260x source range	Model 260x output voltage setting	Output voltage limits (1 year, 18°C–28°C)
100mV	90.000mV	89.732 to 90.268mV
1V	0.90000V	0.89942 to 0.90058V
6V	5.4000V	5.39712 to 5.40288V
40V	36.000V	35.9808 to 36.0192V

## Voltage measurement accuracy

Follow the steps below to verify that the Model 260x voltage measurement accuracy is within specified limits. The test involves setting the source voltage, as measured by a precision digital multimeter, and then verifying that the Model 260x voltage readings are within required limits.

1. With the power off, connect the digital multimeter to the Model 260x output terminals using 4-wire connections, as shown in [Figure 15-1](#).
2. Select the multimeter DC volts function.
3. Select the Model 2602 single-channel display mode.
4. Enable the Model 260x 4-wire (remote sense) mode by pressing CONFIG then MEAS, then select V-MEAS > SENSE-MODE > 4-WIRE.
5. Set the Model 260x to both source and measure voltage by pressing the SRC and MEAS keys, and make sure the source output is turned on.
6. Verify voltage measurement accuracy for each of the voltages listed in [Table 15-3](#). For each test point:
  - Select the correct source range.
  - Set the Model 260x output voltage to the indicated value **as measured by the digital multimeter**.
  - If necessary, press the TRIG key to display readings.
  - Verify that the Model 260x voltage reading is within the limits given in the table. It may not be possible to set the voltage source to the required value. Use the closest possible setting, and modify reading limits accordingly.
7. Repeat the procedure for negative source voltages with the same magnitudes as those listed in [Table 15-3](#).
8. For the Model 2602, repeat the above procedure for the other channel.

Table 15-3  
Voltage measurement accuracy limits

Model 260x source and measure range <sup>1</sup>	Source voltage <sup>2</sup>	Model 260x voltage reading limits (1 year, 18°C–28°C)
100mV	90.000mV	89.8365 to 90.1635mV
1V	0.90000V	0.899665 to 0.900335V
6V	5.4000V	5.39819 to 5.40181V
40V	36.000V	35.9866 to 36.0134V

1. Measure range coupled to source range when simultaneously sourcing and measuring voltage.
2. As measured by precision digital multimeter. Use closest possible value, and modify reading limits accordingly if necessary.

## Output current accuracy

Follow the steps below to verify that the Model 260x output current accuracy is within specified limits.

1. With the power off, connect the digital multimeter to the Model 260x output terminals, as shown in [Figure 15-2](#).
2. Select the multimeter DC current measuring function.
3. Select the Model 2602 single-channel display mode.
4. Press the Model 260x SRC key to source current, and make sure the source output is turned on.
5. Verify output current accuracy for each of the currents for the 100nA to 1A ranges listed in [Table 15-4](#). For each test point:
  - Select the correct source range.
  - Set the Model 260x output current to the correct value.
  - Verify that the multimeter reading is within the limits given in the table.
6. Repeat the procedure for negative output currents with the same magnitudes as those listed in [Table 15-4](#).
7. Turn the output off, and change connections as shown in [Figure 15-3](#) (use 4-wire connections to the 0.5Ω resistor as shown).
8. Select the DMM DC volts function.
9. Repeat steps 4 through 7 for the 3A range. Calculate the current from the DMM voltage reading and characterized 0.5Ω resistance value:  $I=V/R$ .
10. For the Model 2602, repeat the above procedure for the other channel.

Figure 15-2  
**Current verification connections (100nA to 1A ranges)**

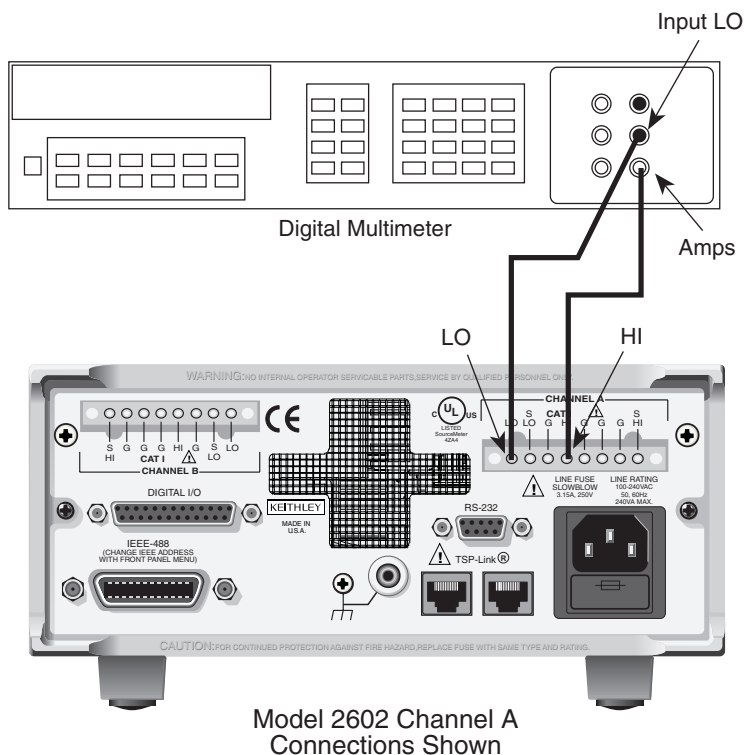


Table 15-4  
**Output current accuracy limits**

Model 260x source range	Model 260x output current setting	Output current limits (1 year, 18°C–28°C)
100nA	90.000nA	89.846 to 90.154nA
1µA	0.90000µA	0.89913 to 0.90087µA
10µA	9.0000µA	8.9953 to 9.0047µA
100µA	90.000µA	89.943 to 90.057µA
1mA	0.90000mA	0.89953 to 0.90047mA
10mA	9.0000mA	8.9943 to 9.0057mA
100mA	90.000mA	89.953 to 90.047mA
1A	0.90000A	0.89885 to 0.90115A
3A	2.40000A	2.39706 to 2.40294A



5. Verify measure current accuracy for each of the currents listed in [Table 15-5](#). For each measurement:
  - Select the correct source range.
  - Set the Model 260x source output to the correct value as measured by the digital multimeter.
  - If necessary, press the TRIG key to display readings.
  - Verify that the Model 260x current reading is within the limits given in the table. It may not be possible to set the current source to the required value. Use the closest possible setting, and modify reading limits accordingly.
6. Repeat the procedure for negative calibrator currents with the same magnitudes as those listed in [Table 15-5](#).
7. Turn the output off, change connections as shown in [Figure 15-3](#), then select the DMM volts function.
8. Repeat steps 4 through 6 for the 3A range. Calculate the current from the DMM voltage reading and characterized  $0.5\Omega$  resistance value.
9. For the Model 2602, repeat the above procedure for the other channel.

Table 15-5

**Current measurement accuracy limits**

Model 260x source and measure range <sup>1</sup>	Source current <sup>2</sup>	Model 260x current reading limits (1 year, 18°C–28°C)
100nA	90.000nA	89.855 to 90.145nA
1μA	0.9000μA	0.899475 to 0.900525μA
10μA	9.0000μA	8.99715 to 9.00285μA
100μA	90.000μA	89.973 to 90.027μA
1mA	0.9000mA	0.89976 to 0.90024mA
10mA	9.0000mA	8.9973 to 9.0027mA
100mA	90.000mA	89.976 to 90.024mA
1A	0.90000A	0.89923 to 0.90077A
3A	2.4000A	2.3981 to 2.4019A

1. Measure range coupled to source range when simultaneously sourcing and measuring current.
2. As measured by precision digital multimeter. Use closest possible value, and modify reading limits accordingly if necessary.

# 16

# Calibration

---

## **Section 16 topics**

**Introduction**, page 16-2

**Environmental conditions**, page 16-2

Temperature and relative humidity, page 16-2

Warm-up period, page 16-2

Line power, page 16-2

**Calibration considerations**, page 16-3

Calibration cycle, page 16-3

Recommended calibration equipment, page 16-4

Calibration errors, page 16-5

**Calibration**, page 16-5

Calibration steps, page 16-5

Calibration commands, page 16-6

Calibration procedure, page 16-8

# Introduction

Use the procedures in this section to calibrate the Model 260x. The procedures require accurate test equipment to measure precise DC voltages and currents.

**WARNING** The information in this section is intended for qualified service personnel only. Do not attempt these procedures unless you are qualified to do so. Some of these procedures may expose you to hazardous voltages.

## Environmental conditions

### Temperature and relative humidity

Conduct the calibration procedures at an ambient temperature of 18-28°C (65-82°F) with relative humidity of less than 70% unless otherwise noted.

### Warm-up period

Allow the Model 260x to warm up for at least two hours before performing calibration.

If the instrument has been subjected to temperature extremes (those outside the ranges stated above), allow additional time for the instrument's internal temperature to stabilize. Typically, allow one extra hour to stabilize a unit that is 10°C (18°F) outside the specified temperature range.

Also, allow the test equipment to warm up for the minimum time specified by the manufacturer.

### Line power

The Model 260x requires a line voltage of 100V to 240V at line frequency of 50Hz or 60Hz. The instrument must be calibrated within this range.

# Calibration considerations

When performing the calibration procedures:

- Make sure that the test equipment is properly warmed up and connected to the correct Model 260x terminals.
- Always allow the source signal to settle before calibrating each point.
- Do not connect test equipment to the Model 260x through a scanner or other switching equipment.
- If an error occurs during calibration, the Model 260x will generate an appropriate error message. See [“Error summary” on page B-2](#) for more information.

**WARNING** The maximum common-mode voltage (voltage between LO and chassis ground) is 250VDC. Exceeding this value may cause a breakdown in insulation, creating a shock hazard.

The terminals of the Model 260x are rated for connection to circuits rated Installation Category I only. Do not connect the Model 260x terminals to CAT II, CAT III, or CAT IV circuits. Connection of the SourceMeter terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltage.

Hazardous voltages may be present on the output and guard terminals. To prevent electrical shock that could cause injury or death, NEVER make or break connections to the Model 260x while the unit is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Model 260x before handling cables connected to the outputs. Putting the equipment into standby mode does not guarantee the outputs are not powered if a hardware or software fault occurs.

## Calibration cycle

Perform calibration at least once a year to ensure the unit meets or exceeds its specifications.



## Recommended calibration equipment

Table 16-1 lists the recommended equipment for the calibration procedures. You can use alternate equipment as long as that equipment has specifications equal to or greater than those listed in the table. When possible, test equipment specifications should be at least four times better than corresponding Model 260x specifications.

Table 16-1

**Recommended calibration equipment**

Description	Manufacturer/ Model	Accuracy
Digital Multimeter	Agilent 3458A	DC Voltage <sup>1</sup> 90mV: ±8ppm 0.9V: ±5ppm 5.4V: ±8ppm 36V: ±11ppm  DC current <sup>1</sup> 90nA: ±430ppm 0.9µA: ±55ppm 9µA: ±25ppm 90µA: ±23ppm 0.9mA: ±20ppm 9mA: ±20ppm 90mA: ±35ppm 0.9A: ±110ppm
0.5Ω, 250W, 0.1% Precision Resistor	Isotek RUG-Z-R500-0.1-TK3	Resistance <sup>2</sup> 0.5Ω: ±125ppm

1. 90-day specifications show accuracy of recommended model at specified calibration point.

2. Resistor used to calibrate 3A and 10A ranges should be characterized to uncertainty shown using resistance function of a digital multimeter before use.

## Calibration errors

The Model 260x checks for errors after each calibration step, minimizing the possibility that improper calibration may occur due to operator error.

You can detect errors while in remote by testing the state of EAV (Error Available) bit (bit 2) in the status byte. (Use the `*STB?` query to request the status byte.)

Query the instrument for the type of error by using the `errorqueue.next` command. The Model 260x will respond with the nature of the error. See [Appendix B](#) for error messages and [Appendix D](#) for status byte information.

## Calibration

Use the following procedure to perform remote calibration by sending commands over the IEEE-488 bus or RS-232 port. The remote commands and appropriate parameters are separately summarized for each step.

### Calibration steps

#### Step sequence

Calibration steps must be performed in the order shown in [Table 16-2](#). Note that all steps are performed using 2-wire (local sensing) except as noted. Calibration of each range is performed as a four-point calibration:

- + ZERO
- + FULL SCALE
- - ZERO
- - FULL SCALE

#### Parameter values

The full-scale parameters are actually 90% of full scale as indicated in [Table 16-2](#). Note that you cannot send a value of exactly 0 for the two zero parameters, but must instead send a very small value, for example 1e-60 or -1e-60.

#### Sense modes

[Table 16-2](#) lists sense modes for the calibration steps. Note that all source and measure ranges are calibrated using the LOCAL sense mode. In addition, the 100mV source and measure ranges are also calibrated using the REMOTE sense mode, and the 1mA and 1V source ranges are also calibrated using the CALA sense mode.

Table 16-2  
**Calibration steps**

Function	Calibration steps <sup>2</sup>	Calibration points <sup>4</sup>	Sense mode <sup>5</sup>
Voltage Source and Measure <sup>1</sup>	100mV 100mV 1V 1V 6V 40V	$\pm 1e-60$ , $\pm 90mV$ $\pm 1e-60$ , $\pm 90mV$ $\pm 1e-60$ , $\pm 0.9V$ $\pm 1e-60$ , $\pm 0.9V$ $\pm 1e-60$ , $\pm 5.4V$ $\pm 1e-60$ , $\pm 36V$	smuX.SENSE_LOCAL smuX.SENSE_REMOTE smuX.SENSE_LOCAL smuX.SENSE_CALA smuX.SENSE_LOCAL smuX.SENSE_LOCAL
Current Source and Measure <sup>1</sup>	100nA 1 $\mu$ A 10 $\mu$ A 100 $\mu$ A 1mA 1mA 10mA 100mA 1A 3A 10A <sup>3</sup>	$\pm 1e-60$ , $\pm 90nA$ $\pm 1e-60$ , $\pm 0.9\mu A$ $\pm 1e-60$ , $\pm 9\mu A$ $\pm 1e-60$ , $\pm 90\mu A$ $\pm 1e-60$ , $\pm 0.9mA$ $\pm 1e-60$ , $\pm 0.9mA$ $\pm 1e-60$ , $\pm 9mA$ $\pm 1e-60$ , $\pm 90mA$ $\pm 1e-60$ , $\pm 0.9A$ $\pm 1e-60$ , $\pm 2.4A$ $\pm 1e-60$ , $\pm 2.4A$	smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_CALA smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_LOCAL smuX.SENSE_LOCAL

1. Calibrate only the source for the SENSE\_CALA sense steps.
2. Steps must be performed in the order shown.
3. 10A range for changing calibration of range only and is not available for normal use.
4. Do not use actual 0 values for zero calibration points. Send very small values such as  $\pm 1e-60$ . Calibration polarities must also be set as shown in the procedures.
5. Output must be off before changing to the CALA sense mode.

## Calibration commands

Table 16-3 summarizes remote calibration commands. For a more complete description of these commands, refer to Section 12.

Table 16-3

**Calibration commands**

Command <sup>1</sup>	Description
smuX.cal.date = caldate smuX.cal.due = caldue smuX.cal.lock() smuX.cal.password = "newpassword" smuX.cal.polarity = polarity	Set calibration date. Set calibration due date. Lock out calibration. Change password to "newpassword". Set polarity:
smuX.cal.restore([calset])	smuX.CAL_AUTO (auto polarity). smuX.CAL_NEGATIVE (negative polarity). smuX.CAL_POSITIVE (positive polarity). Load calibration set of constants:
smuX.cal.save()	smuX.CALSET_NOMINAL (nominal constants). smuX.CALSET_FACTORY (factory constants). smuX.CALSET_DEFAULT (normal constants). smuX.CALSET_PREVIOUS (previous constants). Store constants in non-volatile memory as DEFAULT calibration set.
calstate = smuX.cal.state	Request calibration state: smuX.CALSTATE_CALIBRATING smuX.CALSTATE_LOCKED smuX.CALSTATE_UNLOCKED
smuX.cal.unlock("password") smuX.measure.calibratei(range, cp1measured, cp1reference, cp2measured, cp2reference)	Unlock calibration (Default password: KI002601 or KI2602.) Calibrate current measure range: <sup>2</sup> ±range (measurement range to calibrate). cp1measured (Model 260x measured value for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured Model 260x measured value for cal. point 2). cp2reference (reference measurement for cal. point 2).
smuX.measure.calibratev(range, cp1measured, cp1reference, cp2measured, cp2reference)	Calibrate voltage measure range: <sup>2</sup> ±range (measurement range to calibrate). cp1measured (Model 260x measured value for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (Model 260x measured value for cal. point 2). cp2reference (reference measurement for cal. point 2).

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

2. Calibration point 1 should be performed at approximately 0% of range; calibration point 2 should be performed at approximately 90% of range. See [Table 16-2](#) for calibration points.

Table 16-3 (continued)  
**Calibration commands**

Command <sup>1</sup>	Description
smuX.source.calibratei(range, cp1expected, cp1reference, cp2expected, cp2reference)	Calibrate current source range: <sup>2</sup> ±range (range to calibrate). cp1expected (source value programmed for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2expected (source value programmed for cal. point 2). cp2reference (reference measurement for cal. point 2).
smuX.source.calibratev(range, cp1expected, cp1reference, cp2expected, cp2reference)	Calibrate voltage source range: <sup>2</sup> ±range (range to calibrate). cp1expected (source value programmed for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2expected (source value programmed for cal. point 2). cp2reference (reference measurement for cal. point 2).

1. smuX = smua for the Model 2601; smuX = smua (Channel A) or smub (Channel B) for the Model 2602.

2. Calibration point 1 should be performed at approximately 0% of range; calibration point 2 should be performed at approximately 90% of range. See [Table 16-2](#) for calibration points.

## Calibration procedure

### Step 1. Prepare the Model 260x for calibration

1. Connect the Model 260x to the controller IEEE-488 interface or RS-232 port using a shielded interface cable.
2. Turn on the Model 260x and the test equipment, and allow them to warm up for at least two hours before performing calibration.
3. Make sure the IEEE-488 or RS-232 interface parameters are set up properly. (Use the MENU key and the COMMUNICATION menu to configure the interface.)

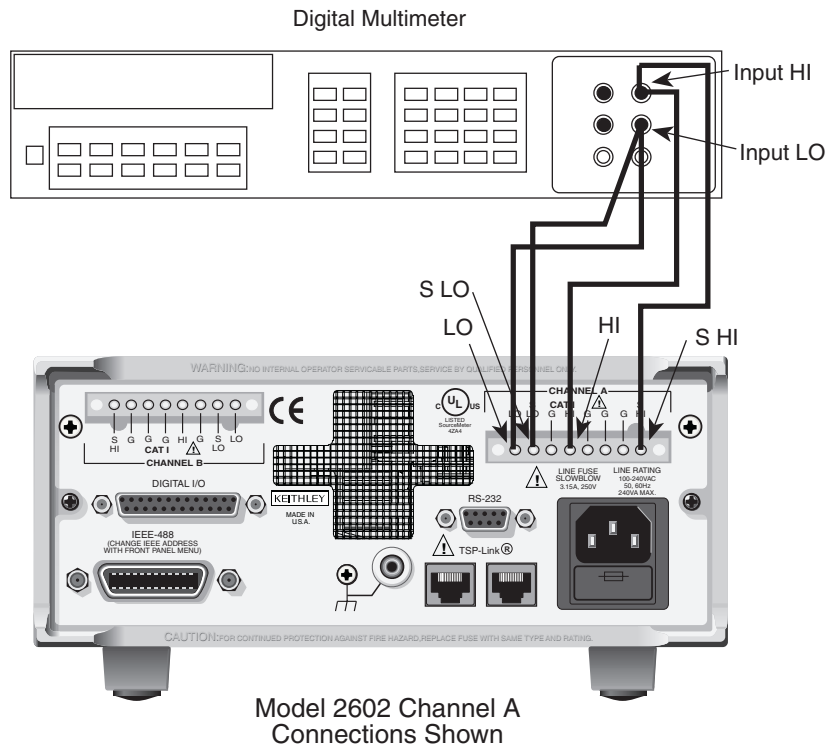
### Step 2. Voltage Calibration

1. Connect the Model 260x to the digital multimeter using the 4-wire connections shown in [Figure 16-1](#), and select the multimeter DC volts function.
2. Send the following commands in order to initialize voltage calibration:
 

```
smua.cal.unlock("KI002602")
smua.reset()
smua.source.func = smua.OUTPUT_DCVOLTS
```

(Substitute the actual calibration password in the `smua.cal.unlock` command. The default Model 2601 password is "KI002601"; the default Model 2602 password is "KI002602".)

Figure 16-1  
Connections for voltage calibration



3. Perform each calibration step listed in [Table 16-2](#) on [page 16-6](#) as follows:
  - a. Select the range being calibrated with this command:  
`smua.source.rangev = range`  
 (Note that it is not necessary to set the measure range for calibration.)  
 For example, for the 1V range, the following command would be sent:  
`smua.source.rangev = 1`
  - b. Select the correct sense mode based on the calibration step from [Table 16-2](#), for example:  
`smua.sense = smua.SENSE_LOCAL`
  - c. Select positive polarity, then set the source output to the positive zero value:  
`smua.cal.polarity = smua.CAL_POSITIVE`  
`smua.source.levelv = 1e-60`
  - d. Turn on the output:  
`smua.source.output = smua.OUTPUT_ON`

- e. Allow the readings to settle, then get both the multimeter and Model 260x voltage readings at the positive zero value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:  
`Z_rdg = smua.measure.v()`
- f. Turn off the output:  
`smua.source.output = smua.OUTPUT_OFF`
- g. Set the source output to the positive full scale value for the present range, for example:  
`smua.source.levelv = 0.9`
- h. Turn on the output:  
`smua.source.output = smua.OUTPUT_ON`
- i. Allow the readings to settle, then get both the multimeter and Model 260x voltage readings at the positive full-scale output value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:  
`FS_rdg = smua.measure.v()`
- j. Turn off the output:  
`smua.source.output = smua.OUTPUT_OFF`
- k. Send the source calibration command using the range, +zero and +FS multimeter readings, and +zero and +FS source values for the parameters:  
`smua.source.calibratev(range, src_Z, DMM_Z_rdg, src_FS, DMM_FS_rdg)`  
Where: `range` = present calibration range  
`src_Z` = +zero 260x source output value  
`DMM_Z_rdg` = +zero DMM measurement  
`src_FS` = +FS 260x source output value  
`DMM_FS_rdg` = +FS DMM measurement  
Typical values for the 1V range:  
`smua.source.calibratev(1, 1e-60, 1e-5, 0.9, 0.903)`

- i. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Model 260x readings, and range setting for the parameters:

```
smua.measure.calibratev(range, Z_rdg, DMM_Z_rdg,  
FS_rdg, DMM_FS_rdg)
```

Where: range = present calibration range

Z\_rdg = +zero 260x measurement

DMM\_Z\_rdg = +zero DMM measurement

FS\_rdg = +FS 260x measurement

DMM\_FS\_rdg = +FS DMM measurement

Typical 1V range values:

```
smua.measure.calibratev(1, 1e-4, 1e-5, 0.92, 0.903)
```

- m. Select negative polarity, then set the source output to the negative zero value, for example:

```
smua.cal.polarity = smua.CAL_NEGATIVE
```

```
smua.source.levelv = -1e-60
```

- n. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

- o. Allow the readings to settle, then get both the multimeter and Model 260x voltage readings at the negative zero value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:

```
Z_rdg = smua.measure.v()
```

- p. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

- q. Set the source output to the negative full scale value, for example:

```
smua.source.levelv = -0.9
```

- r. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

- s. Allow the readings to settle, then get both the multimeter and Model 260x voltage readings at the negative full-scale output value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:

```
FS_rdg = smua.measure.v()
```

- t. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```



- u. Send the source calibration command using the range, -zero and -FS multimeter readings, and -zero and -FS source values for the parameters:

```
smua.source.calibratev(-range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```

Where: -range = negative of the present calibration range

src\_Z = -zero 260x source output value

DMM\_Z\_rdg = -zero DMM measurement

src\_FS = -FS 260x source output value

DMM\_FS\_rdg = -FS DMM measurement

Typical values for the 1V range:

```
smua.source.calibratev(-1,-1e-60,-1e-4,-0.9,-0.896)
```

- v. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Model 260x readings, and range setting for the parameters:

```
smua.measure.calibratev(-range,Z_rdg,DMM_Z_rdg,
FS_rdg,DMM_FS_rdg)
```

Where: -range = negative of the present calibration range

Z\_rdg = -zero 260x measurement

DMM\_Z\_rdg = -zero DMM measurement

FS\_rdg = -FS 260x measurement

DMM\_FS\_rdg = -FS DMM measurement

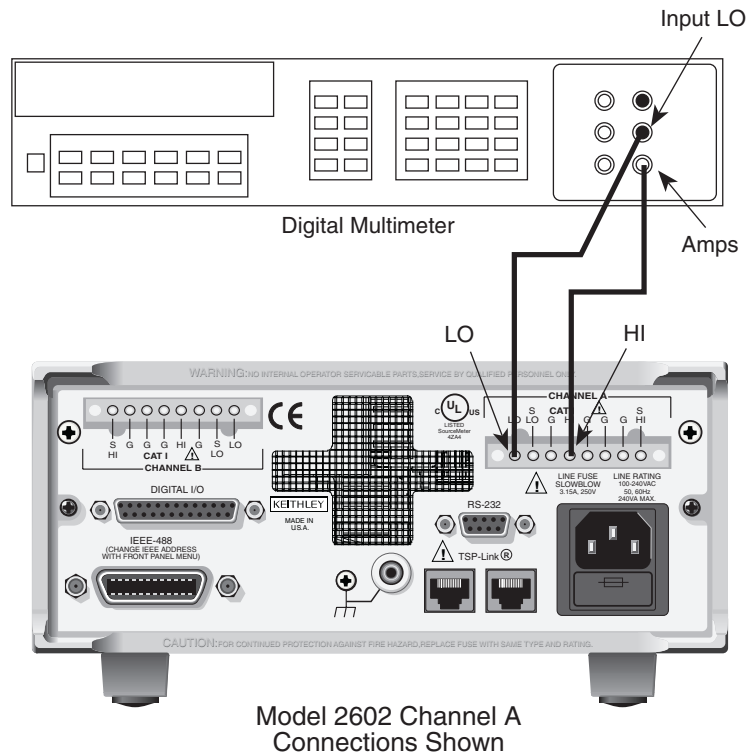
Typical 1V range values:

```
smua.measure.calibratev(-1,-1e-4,-1e-6,-0.89,-0.896)
```

4. Be sure to complete steps a through v for all six voltage steps in [Table 16-2](#) before continuing with current calibration.
5. Select auto polarity mode:  

```
smua.cal.polarity = smua.CAL_AUTO
```

Figure 16-2  
Connections for current calibration (100nA to 1A ranges)



### Step 3. Current Calibration

1. Connect the Model 260x to the digital multimeter (see [Figure 16-2](#)), and select the multimeter DC current function.
2. Send this command to initialize current calibration:  
`smua.source.func = smua.OUTPUT_DCAMPS`
3. Perform each calibration step listed in [Table 16-2 on page 16-6](#):
  - a. Select the range being calibrated with this command:  
`smua.source.rangei = range`  
(Note that it is not necessary to set the measure range for calibration.)  
For example, for the 1A range, the following command would be sent:  
`smua.source.rangei = 1`
  - b. Select the correct sense mode based on the calibration step from [Table 16-2](#), for example:  
`smua.sense = smua.SENSE_LOCAL`

- c. Select positive polarity, then set the source output to the positive zero value:
 

```
smua.cal.polarity = smua.CAL_POSITIVE
smua.source.level1 = 1e-60
```
- d. Turn on the output:
 

```
smua.source.output = smua.OUTPUT_ON
```
- e. Allow the readings to settle, then get both the multimeter and Model 260x current readings at the positive zero value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:
 

```
Z_rdg = smua.measure.i()
```
- f. Turn off the output:
 

```
smua.source.output = smua.OUTPUT_OFF
```
- g. Set the source output to the positive full scale value for the present range, for example:
 

```
smua.source.level1 = 0.9
```
- h. Turn on the output:
 

```
smua.source.output = smua.OUTPUT_ON
```
- i. Allow the readings to settle, then get both the multimeter and Model 260x voltage readings at the positive full-scale output value. (The Model 260x measurement is not necessary if calibration is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:
 

```
FS_rdg = smua.measure.i()
```
- j. Turn off the output:
 

```
smua.source.output = smua.OUTPUT_OFF
```
- k. Send the source calibration command using the range, zero and +FS multimeter readings, and zero and +FS source values for the parameters:
 

```
smua.source.calibratei(range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```

Where: range = present calibration range  
 src\_Z = +zero 260x source output value  
 DMM\_Z\_rdg = +zero DMM measurement  
 src\_FS = +FS 260x source output value  
 DMM\_FS\_rdg = +FS DMM measurement

Typical values for the 1A range:  

```
smua.source.calibratei(1,1e-60,1e-4,0.9,0.88)
```

- l. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Model 260x readings, and range setting for the parameters:

```
smua.measure.calibratei(range, Z_rdg, DMM_Z_rdg,
FS_rdg, DMM_FS_rdg)
```

Where: range = present calibration range

Z\_rdg = +zero 260x measurement

DMM\_Z\_rdg = +zero DMM measurement

FS\_rdg = +FS 260x measurement

DMM\_FS\_rdg = +FS DMM measurement

Typical 1A range values:

```
smua.measure.calibratei(1, 1e-5, 1e-4, 0.89, 0.88)
```

- m. Select negative polarity, then set the source output to the negative zero value, for example:

```
smua.cal.polarity = smua.CAL_NEGATIVE
```

```
smua.source.level1 = -1e-60
```

- n. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

- o. Allow the readings to settle, then get both the multimeter and Model 260x current readings at the negative zero full-scale value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:

```
Z_rdg = smua.measure.i()
```

- p. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

- q. Set the source output to the negative full scale value, for example:

```
smua.source.level1 = -0.9
```

- r. Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

- s. Allow the readings to settle, then get both the multimeter and Model 260x current readings at the negative full-scale output value. (The Model 260x measurement is not necessary if this calibration step is being done on the CALA sense mode.) The two measurements should be made as close as possible in time. Use this command for the Model 260x:

```
FS_rdg = smua.measure.v()
```

- t. Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

- u. Send the source calibration command using the `-range`, `-zero` and `-FS` multimeter readings, and `-zero` and `-FS` source values for the parameters:

```
smua.source.calibratei(-range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```

Where: `-range` = negative of the present calibration range

`src_Z` = `-zero` 260x source output value

`DMM_Z_rdg` = `-zero` DMM measurement

`src_FS` = `-FS` 260x source output value

`DMM_FS_rdg` = `-FS` DMM measurement

Typical values for the 1A range:

```
smua.source.calibratei(-1,-1e-60,-1e-5,-0.9,-0.892)
```

- v. If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Model 260x readings, and range setting for the parameters:

```
smua.measure.calibratei(-range,Z_rdg,DMM_Z_rdg,
FS_rdg,DMM_FS_rdg)
```

Where: `-range` = negative of the present calibration range

`Z_rdg` = `-zero` 260x measurement

`DMM_Z_rdg` = `-zero` DMM measurement

`FS_rdg` = `-FS` 260x measurement

`DMM_FS_rdg` = `-FS` DMM measurement

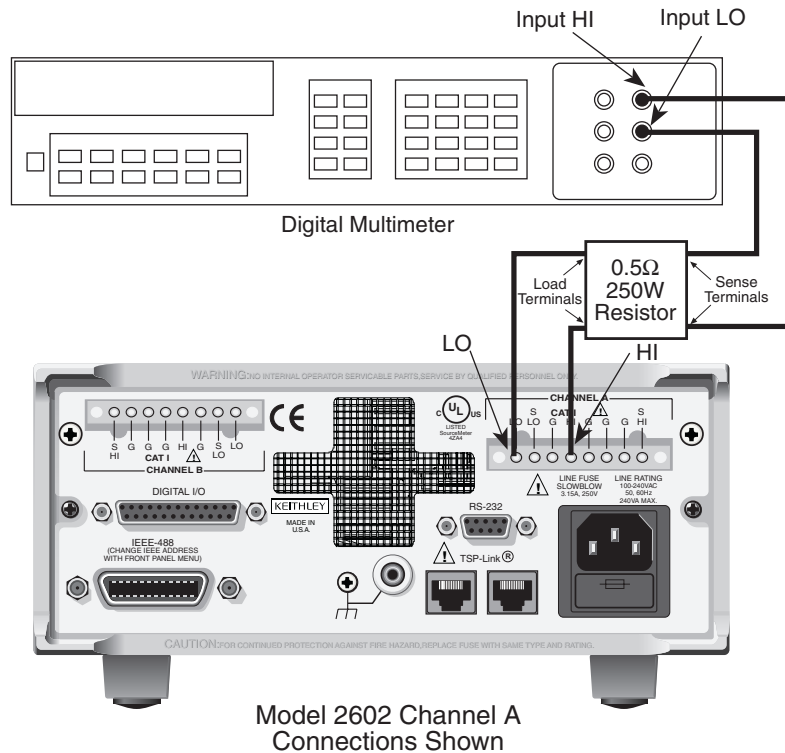
Typical 1A range values:

```
smua.measure.calibratei(-1,-1e-4,-1e-5,-0.91,-0.892)
```

4. Be sure to complete steps a through v for the 100nA to 1A ranges before continuing with 3A and 10A range calibration.
5. Change connections as shown in [Figure 16-3](#) (use 4-wire connections to the 0.5Ω resistor as shown).
6. Select the DMM DC volts function.
7. Repeat steps a through v for the 3A and 10A ranges. Compute the current reading from the DMM voltage reading and characterized 0.5Ω resistance value:  $I = V/R$ .
8. Select auto polarity mode:  

```
smua.cal.polarity = smua.CAL_AUTO
```

Figure 16-3  
Connections for current calibration (3A and 10A ranges)



#### Step 4. Program calibration dates

Use the following commands to set the calibration date and calibration due date:

```
smua.cal.date = os.time({year=2005, month=1, day=1})
smua.cal.due = os.time({year=2006, month=1, day=1})
```

The actual year, month, day, and optionally hour, and minute should be used. (Seconds can be given but will essentially be ignored due to the precision of the internal date storage format.) The allowable range for the `year` is from 2005 to 2037, the `month` is from 1 to 12, and the `day` is from 1 to 31.

#### Step 5. Save calibration constants

Calibration is now complete, so you can store the calibration constants in non-volatile memory by sending the following command:

```
smua.cal.save ()
```

Calibration just performed will be temporary unless you send the `save` command.

**Step 6. Lock out calibration**

To lock out further calibration, send the following command after completing the calibration procedure:

```
smua.cal.lock()
```

**Step 7. Repeat calibration procedure for Model 2602 Channel B**

For the Model 2602 only, repeat the entire procedure above for Channel B. Be sure to:

- Make test connections to Channel B terminals.
- Substitute “smub” for “smua” in all commands.

# 17

# Routine Maintenance

---

## **Section 17 topics**

**Introduction**, page 17-2

**Line fuse replacement**, page 17-2

**Front panel tests**, page 17-2

KEYS test, page 17-3

DISPLAY PATTERNS test, page 17-4



# Introduction

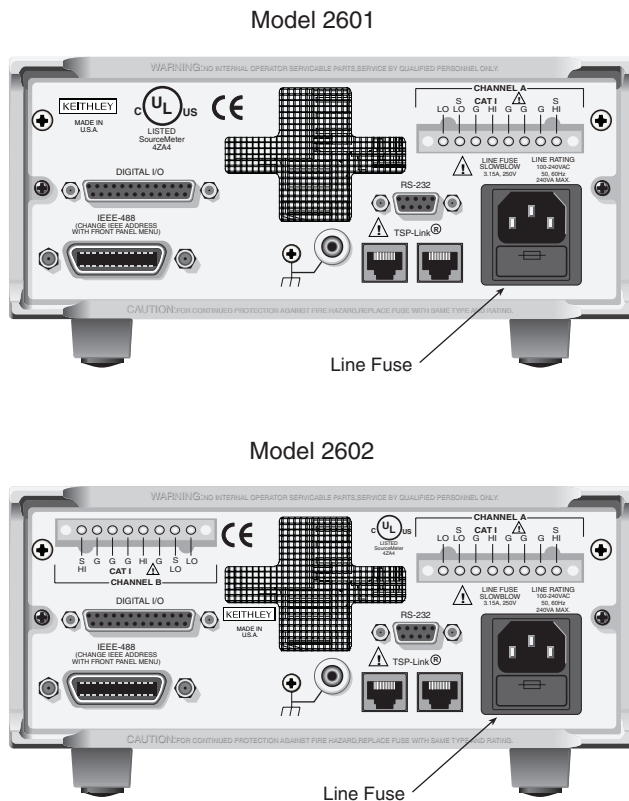
The information in this section deals with routine maintenance that can be performed by the operator.

## Line fuse replacement

**WARNING** Disconnect the line cord at the rear panel, and remove all test leads connected to the instrument before replacing the line fuse.

The power line fuse is accessible from the rear panel, just below the AC power receptacle (Figure 17-1).

Figure 17-1  
Line fuse replacement



Return to [Section 17 topics](#)

Perform the following steps to replace the line fuse:

1. Carefully grasp and squeeze together the locking tabs that secure the fuse carrier to the fuse holder.
2. Pull out the fuse carrier, and replace the fuse with the type specified in [Table 17-1](#).

**CAUTION** To prevent instrument damage, use only the fuse type specified in [Table 17-1](#).

3. Reinstall the fuse carrier.

If the power line fuse continues to blow, a circuit malfunction exists and must be corrected. Return the unit to Keithley Instruments for repair.

Table 17-1

**Line fuse**

Line voltage	Rating	Keithley part no.
100-240V	250V, 3.15A, Slow Blow 5 × 20mm	FU-106-3.15

## Front panel tests

There are two front panel tests: one to test the functionality of the front panel keys and one to test the display.

### KEYS test

The KEYS test lets you check the functionality of each front panel key. Perform the following steps to run the KEYS test.

1. Display the MAIN MENU by pressing the MENU key.
2. Select TEST, and press ENTER or the Rotary Knob to display the SELF-TEST MENU.
3. Select DISPLAY-TESTS, and press ENTER or the Rotary Knob to display the following menu:  
FRONT PANEL TESTS  
KEYS DISPLAY\_PATTERNS
4. Select KEYS, and press ENTER or the Rotary Knob to start the test. When a key is pressed, the label name for that key will be displayed to indicate that it is functioning properly. When the key is released, the message “No keys pressed” is displayed.

5. Pressing EXIT tests the EXIT key. However, the second consecutive press of EXIT aborts the test and returns the instrument to the SELF-TEST MENU. Continue pressing EXIT to back out of the menu structure.

## DISPLAY PATTERNS test

The Display Patterns test lets you verify that each pixel and annunciator in the vacuum fluorescent display is working properly. Perform the following steps to run the display test:

1. Display the MAIN MENU by pressing the MENU key.
2. Select TEST, and press ENTER or the Rotary Knob to display the SELF-TEST MENU.
3. Select DISPLAY-TESTS, and press ENTER or the Rotary Knob to display the following menu:  
FRONT PANEL TESTS  
KEYS DISPLAY\_PATTERNS
4. Select DISPLAY\_PATTERNS, and press ENTER or the Rotary Knob to start the display test. There are three parts to the display test. Each time ENTER or Rotary Knob is pressed, the next part of the test sequence is selected. The three parts of the test sequence are as follows:
  - Checkerboard pattern and the annunciators that are on during normal operation.
  - Checkerboard pattern (alternate pixels on) and all annunciators.
  - Each digit (and adjacent annunciator) is sequenced. All of the pixels of the selected digit are on.
5. When finished, abort the display test by pressing EXIT. The instrument returns to the FRONT PANEL TESTS MENU. Continue pressing EXIT to back out of the menu structure.

# A Specifications

---

**Appendix A topics**

# 2601 and 2602 System SourceMeter® Specifications

## SPECIFICATION CONDITIONS

This document contains specifications and supplemental information for the 2601 & 2602. Specifications are the standards against which the 2601 & 2602 are tested. Upon leaving the factory the 2601 & 2602 meet these specifications. Supplemental and typical values are non-warranted, apply at 23°C and are provided solely as useful information.

The source and measurement accuracies are specified at the SourceMeter CHANNEL A (2601 & 2602) or SourceMeter CHANNEL B (2602) terminals under the following conditions:

1. 23°C + 5°C <70% relative humidity.
2. After 2 hour warm up.
3. Speed normal (1 NPLC).
4. A/D auto-zero enabled.
5. Remote sense operation or properly zeroed local sense operation.
6. Calibration period = 1 year.

## SOURCE SPECIFICATIONS

### Voltage Programming Accuracy<sup>1</sup>

Range	Programming Resolution	Accuracy (1 Year) 23°C ±5°C ±(% rdg. + volts)	Noise (peak-peak) 0.1Hz-10Hz
100.000mV	1µV	0.02% + 250µV	20µV
1.00000V	10µV	0.02% + 400µV	50µV
6.00000V	10µV	0.02% + 1.8mV	100µV
40.0000V	100µV	0.02% + 12mV	500µV

**TEMPERATURE COEFFICIENT (0°-18°C & 28°-50°C):** ±(0.15 x accuracy specification)/°C.

**MAXIMUM OUTPUT POWER AND SOURCE/SINK LIMITS<sup>2</sup>:** 40.4W per channel maximum, ±40.4V @ ±1.0A, ±6.06V @ ±3.0A, four quadrant source or sink operation.

**VOLTAGE REGULATION:** Line: 0.01% of range. Load: 0.01% of range + 100µV.

**NOISE 10Hz-20MHz (peak-peak):** 25mV typical into a resistive load.

**CURRENT LIMIT/COMPLIANCE<sup>3</sup>:** Bipolar current limit (compliance) set with single value. Minimum value is 10nA. Accuracy same as current source.

**OVERSHOOT:** <0.1% + 10mV typical (step size = 10% to 90% of range, resistive load, maximum current limit/compliance).

**GUARD OFFSET VOLTAGE:** <10mV typical (Iout <= 100mA).

### Current Programming Accuracy

Range	Programming Resolution	Accuracy (1 Year) 23°C ±5°C ±(% rdg. + amps)	Noise (peak-peak) 0.1Hz-10Hz
100.000nA	1pA	0.06 + 100pA	5pA
1.00000µA	10pA	0.03% + 600pA	15pA
10.0000µA	100pA	0.03% + 2nA	50pA
100.000µA	1nA	0.03% + 30nA	2nA
1.00000mA	10nA	0.03% + 200nA	5nA
10.0000mA	100nA	0.03% + 3µA	200nA
100.000mA	1µA	0.03% + 20µA	500nA
1.00000A	10µA	0.05% + 900µA	50µA
3.00000A <sup>2</sup>	10µA	0.06% + 1.5mA	150µA

**TEMPERATURE COEFFICIENT (0°-18°C & 28°-50°C):** ±(0.15 x accuracy specification)/°C.

**MAXIMUM OUTPUT POWER AND SOURCE/SINK LIMITS<sup>2</sup>:** 40.4W per channel maximum, ±1.01A @ ±40.0V, ±3.03A @ ±6.0V, four quadrant source or sink operation.

**CURRENT REGULATION:** Line: 0.01% of range. Load: 0.01% of range + 100pA.

**VOLTAGE LIMIT/COMPLIANCE<sup>4</sup>:** Bipolar voltage limit (compliance) set with a single value. Minimum value is 10mV. Accuracy same as voltage source.

**OVERSHOOT:** <0.1% typical (step size = 10% to 90% of range, resistive load, see CURRENT SOURCE OUTPUT SETTLING TIME for additional test conditions).

## ADDITIONAL SOURCE SPECIFICATIONS

**TRANSIENT RESPONSE TIME:** <70µs for the output to recover to 0.1% for a 10% to 90% step change in load.

**VOLTAGE SOURCE OUTPUT SETTLING TIME:** Time required to reach 0.1% of final value after source level command is processed on a fixed range.  
100mV, 1V Ranges: <50µs typical.  
6V Range: <100µs typical.  
40V Range: <150µs typical.

**CURRENT SOURCE OUTPUT SETTLING TIME:** Time required to reach 0.1% of final value after source level command is processed on a fixed range. Values below for Iout Rload = 2V unless noted.

3A-10mA Ranges: <80µs typical (current less than 2.5A, Rload greater than 1.5Ω).

1mA Ranges: <100µs typical.  
100µA Range: <150µs typical.  
10µA Range: <500µs typical.  
1µA Range: <2ms typical.  
100nA Range: <20ms typical.

**DC FLOATING VOLTAGE:** Output can be floated up to ±250VDC from chassis ground.

**REMOTE SENSE OPERATING RANGE<sup>1</sup>:**

Maximum voltage between HI and SENSE HI = 3V.  
Maximum voltage between LO and SENSE LO = 3V.

**VOLTAGE OUTPUT HEADROOM:**

40V Range: Max. output voltage = 42V — total voltage drop across source leads (Maximum 1Ω per source lead).  
6V Range: Max. output voltage = 8V — total voltage drop across source leads.

**OVER TEMPERATURE PROTECTION:** Internally sensed temperature overload puts unit in standby mode.

**VOLTAGE SOURCE RANGE CHANGE OVERSHOOT:** Overshoot into a 100kΩ load, 20MHz BW, 300mV typical.

**CURRENT SOURCE RANGE CHANGE OVERSHOOT:** <5% + 300mV/Rload of larger range typical. (See CURRENT SOURCE OUTPUT SETTLING TIME for additional test conditions.)

## NOTES

1. Add 50µV to source accuracy specifications per volt of HI lead drop.
2. Full power source operation regardless of load to 30°C ambient. Above 30°C and/or power sink operation, refer to Section 8 - Operating boundaries in the Series 2600 Reference Manual for additional power derating information.
3. For sink mode operation (quadrant II and IV), add 12% of limit range and ±0.02% of limit setting to corresponding current limit accuracy specifications. For 1A range add an additional 40mA of uncertainty.
4. For sink mode operation (quadrant II and IV), add 10% of compliance range and ±0.02% of limit setting to corresponding voltage source specification. For 100mV range add an additional 60mV of uncertainty.

# 2601 and 2602 System SourceMeter® Specifications

## METER SPECIFICATIONS

### VOLTAGE MEASUREMENT ACCURACY<sup>1</sup>

Range	Display Resolution <sup>3</sup>	Input Resistance	Accuracy (1 Year) 23°C ±5°C ±(% rdg. + volts)
100.000mV	1μV	>10GΩ	0.015% + 150μV
1.00000V	10μV	>10GΩ	0.015% + 200μV
6.00000V	10μV	>10GΩ	0.015% + 1mV
40.0000V	100μV	>10GΩ	0.015% + 8mV

**TEMPERATURE COEFFICIENT (0°–18°C & 28°–50°C):** ±(0.15 x accuracy specification)/°C.

### CURRENT MEASUREMENT ACCURACY

Range	Display Resolution <sup>3</sup>	Voltage Burden <sup>2</sup>	Accuracy (1 Year) 23°C ±5°C ±(% rdg. + amps)
100.000nA	1pA	<1mV	0.05% + 100pA
1.00000μA	10pA	<1mV	0.025% + 300pA
10.0000μA	100pA	<1mV	0.025% + 600pA
100.000μA	1nA	<1mV	0.02% + 12nA
1.00000mA	10nA	<1mV	0.02% + 60nA
10.0000mA	100nA	<1mV	0.02% + 1.2μA
100.000mA	1μA	<1mV	0.02% + 6μA
1.00000A	10μA	<1mV	0.03% + 700μA
3.00000A	10μA	<1mV	0.05% + 1mA

**TEMPERATURE COEFFICIENT (0°–18°C & 28°–50°C):** ±(0.15 x accuracy specification)/°C.

### ADDITIONAL METER SPECIFICATIONS

**LOAD IMPEDANCE:** Stable into 10,000pF typical.

**COMMON MODE VOLTAGE:** 250VDC.

**COMMON MODE ISOLATION:** >1GΩ, <4500pF.

**OVERRANGE:** 101% of source range, 102% of measure range.

**MAXIMUM SENSE LEAD RESISTANCE:** 1kΩ for rated accuracy.

**SENSE INPUT IMPEDANCE:** >10GΩ.

### NOTES

1. Add 50μV to source accuracy specifications per volt of HI lead drop.
2. Four-wire remote sense only.
3. Applies when in single channel display mode.

## General

**HOST INTERFACES:** Computer control interfaces.

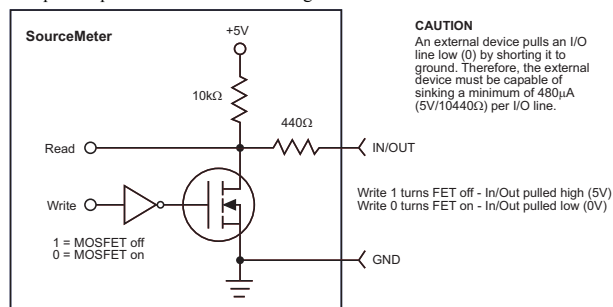
**IEEE-488:** IEEE-488.1 compliant. Supports IEEE-488.2 common commands and status model topology.

**RS-232:** Baud rates from 300 bps to 115200 bps. Programmable number of data bits, parity type, and flow control (RTS/CTS hardware or none). When not programmed as the active host interface, the SourceMeter can use the RS-232 interface to control other instrumentation.

**EXPANSION INTERFACE:** The TSP-Link expansion interface allows TSP enabled instruments to trigger and communicate with each other. Cable Type: Category 5e or higher LAN crossover cable. Length: 3 meters maximum between each TSP enabled instrument.

### DIGITAL I/O INTERFACE:

Connector: 25-pin female D  
Input/Output Pins: 14 I/O bits. See figure below.



**Output Enable Pin:** Active high input. When the output enable input has been activated, each SourceMeter output will be disabled when Output Enable is <1.5V.

**5V Power Supply Pin:** Limited to 600mA, solid state fuse protected.

**POWER SUPPLY:** 100V to 240VAC, 50–60Hz (manual setting), 240VA max.

**COOLING:** Forced air. Side intake and rear exhaust. One side must be unobstructed when rack mounted.

**WARRANTY:** 1 year.

**EMC:** Conforms to European Union Directive 89/336/EEC, EN 61326-1.

**SAFETY:** Conforms to European Union Directive 73/23/EEC, EN 61010-1, and UL 61010-1.

**DIMENSIONS:** 89mm high x 213mm wide x 460mm deep (3 1/2 in x 8 3/8 in x 17 1/2 in). Bench Configuration (with handle & feet): 104mm high x 238mm wide x 460mm deep (4 1/8 in x 9 3/8 in x 17 1/2 in).

**WEIGHT:** 2601: 4.75kg (10.40 lbs). 2602: 5.50kg (12.00 lbs).

**ENVIRONMENT:** For indoor use only.

**Altitude:** Maximum 2000 meters above sea level.

**Operating:** 0°–50°C, 70%R.H. up to 35°C. Derate 3% R.H./°C, 35°–50°C.

**Storage:** –25°C to 65°C.

### ACCESSORIES SUPPLIED:

**Cables & Connectors:** SourceMeter DUT interface connector kit for each SourceMeter channel. Kit includes one hooded screw terminal connector that mates with the SourceMeter measurement terminals. TSP-Link cable, power cable.

**Printed Documentation:** User's Manual

**Electronic Media:** CD-ROMs containing

- User's and Reference manual .PDF files
- Test Script Builder script development software
- IVI/VISA drivers for VB, VC/C++, LabVIEW, and LabWindows/CVI

# 2601 and 2602 System SourceMeter® Specifications

## SPEED SPECIFICATIONS<sup>1</sup>

### MAXIMUM SWEEP OPERATION RATES (operations per second) FOR 60Hz (50Hz):

A/D CONVERTER SPEED	TRIGGER ORIGIN	MEASURE TO MEMORY	MEASURE TO GPIB	SOURCE MEASURE TO MEMORY	SOURCE MEASURE TO GPIB	SOURCE MEASURE PASS/FAIL TO MEMORY	SOURCE MEASURE PASS/FAIL TO GPIB
0.001 NPLC	Internal	10000 (10000)	8000 (8000)	5500 (5500)	3600 (3600)	4900 (4900)	3100 (3100)
0.001 NPLC	Digital I/O	2700 (2650)	2100 (2100)	2300 (2300)	1900 (1875)	2200 (2150)	1800 (1775)
0.01 NPLC	Internal	4000 (3500)	3600 (3200)	2750 (2700)	2300 (2100)	2800 (2500)	2100 (1975)
0.01 NPLC	Digital I/O	1900 (1775)	1600 (1500)	1700 (1600)	1450 (1400)	1600 (1500)	1400 (1325)
0.1 NPLC	Internal	565 (475)	555 (470)	540 (450)	510 (440)	535 (455)	505 (430)
0.1 NPLC	Digital I/O	490 (420)	470 (405)	470 (410)	450 (390)	470 (400)	450 (390)
1.0 NPLC	Internal	59 (49)	59 (49)	58 (49)	58 (48)	58 (49)	58 (48)
1.0 NPLC	Digital I/O	58 (48)	58 (48)	58 (48)	57 (48)	57 (48)	57 (48)

### Maximum SINGLE MEASUREMENT RATES (operations per second) FOR 60Hz (50Hz):

A/D CONVERTER SPEED	TRIGGER ORIGIN	MEASURE TO GPIB	SOURCE MEASURE TO GPIB	SOURCE MEASURE PASS/FAIL TO GPIB
0.001 NPLC	Internal	1100 (1000)	880 (880)	840 (840)
0.01 NPLC	Internal	950 (900)	780 (760)	730 (710)
0.1 NPLC	Internal	390 (345)	355 (310)	340 (305)
1.0 NPLC	Internal	57 (48)	56 (47)	56 (47)

**MAXIMUM MEASUREMENT RANGE CHANGE RATE:** >4500/second typical.

**MAXIMUM SOURCE RANGE CHANGE RATE:** >1000/second typical.

**MAXIMUM SOURCE FUNCTION CHANGE RATE:** >500/second typical.

**EXTERNAL TRIGGER INPUT:** The Digital I/O interface signals can be configured to behave as trigger inputs.

**Input Latency (time from trigger input to start of measurement or source change):** <150µs, typical.

**Input Jitter:** <100µs, typical.

**COMMAND PROCESSING TIME:** Maximum time required for the output to begin to change following the receipt of the smux.source.levelv or smux.source.leveli command. <1ms typical.

## NOTES

1. See the Speed Specifications Test Conditions Appendix in the Series 2600 Reference Manual for more information regarding test conditions.

# 2601 and 2602 System SourceMeter® Specifications

## SUPPLEMENTAL INFORMATION

**FRONT PANEL INTERFACE:** 2-line Vacuum Florescent Display (VFD) with keypad and rotary knob.

**Display:**

- Show error messages and user defined messages.
- Display source and limit settings.
- Show current and voltage measurements.
- View measurements stored in non-volatile reading buffers.

**Keypad Operations:**

- Change host interface settings.
- Save and restore instrument setups.
- Load and run factory, and user defined, test scripts (i.e. sequences) that prompt for input and send results to the display.
- Store measurements into non-volatile reading buffers.

**PROGRAMMING:** Embedded Test Script Processor (TSP) accessible from any host interface. Responds to individual instrument control commands. Responds to high-speed test scripts comprised of instrument control commands and Test Script Language (TSL) statements (e.g. branching, looping, math, etc...). Able to execute high-speed test scripts stored in memory without host intervention.

**Minimum Memory Available:** 3 Mbytes (approximately 50,000 lines of TSL code).

**Test Script Builder:** Integrated Development Environment for building, running, and managing TSP scripts. Includes an Instrument Console for communicating with any TSP enabled instrument in an interactive manner. Requires:

- VISA (NI-VISA included on CD).
- Microsoft .NET Framework (included on CD).
- Keithley I/O Layer (included on CD).
- Pentium III 800MHz or faster personal computer.
- Microsoft Windows 98, NT, 2000, or XP.

**Drivers:** IVI/VISA drivers for VB, VC/C++, LabVIEW, TestPoint, and LabWindows/CVI.

**READING BUFFERS:** Non-volatile storage area(s) reserved for measurement data. Reading buffers are arrays of measurement elements. Each element can hold the following items:

- Measurement
- Measurement status
- Timestamp
- Source setting (at the time the measurement was taken)
- Range information

Two reading buffers are reserved for each SourceMeter channel. Reading buffers can be filled using the front panel STORE key and retrieved using the RECALL key or host interface.

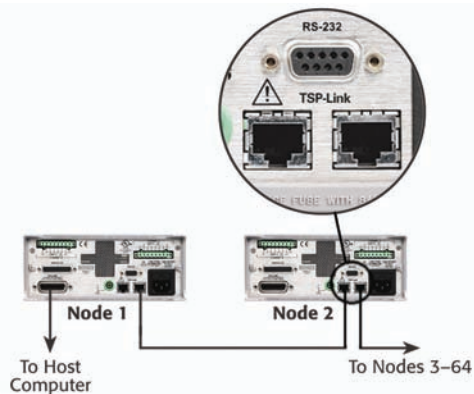
**Buffer Size, with timestamp and source setting:** >50,000 samples.

**Buffer Size, without timestamp and source setting:** >100,000 samples.

**Battery Backup:** Lithium-ion battery backup. 30 days of non-volatile storage @ 23°C, and >4 hours of charge time. 3 year battery life @ 23°C. 1.5 year battery life @ 50°C.

**FACTORY TSP SCRIPTS:** See [www.keithley.com](http://www.keithley.com) for Keithley-supported application-specific scripts.

**SYSTEM EXPANSION:** The TSP-Link expansion interface allows TSP enabled instruments to trigger and communicate with each other. See figure below:



- Each SourceMeter has two TSP-Link connectors to facilitate chaining instruments together.
  - Once SourceMeters are interconnected via TSP-Link, a computer can access all of the resources of each SourceMeter via the host interface of any SourceMeter.
  - A maximum of 64 TSP-Link nodes can be interconnected. Each SourceMeter consumes one TSP-Link node.

**TIMER:** Free running 47 bit counter with 1MHz clock input. Reset each time instrument powers up. Rolls over every 4 years.

**Timestamp:** TIMER value automatically saved when each measurement is triggered.

**Resolution:** 1 $\mu$ s.

**Accuracy:** 50ppm.



# B Error and Status Messages

---

## **Appendix B topics**

**Introduction**, page B-2

**Error summary**, page B-2

**Reading errors**, page B-2

# Introduction

This appendix includes information on error levels, how to read errors, and a complete listing of error messages.

## Error summary

Errors are listed in [Table B-2](#) starting on the next page. Error levels are listed below:

- NO\_SEVERITY: not severe
- INFORMATIONAL: informational status message only
- RECOVERABLE: error not serious, can be recovered
- SERIOUS: error serious, but unit still operational by correcting error
- FATAL: unit non-operational

## Reading errors

When errors occur, the error messages will be placed in the error queue (see “Queues” on page D-29). [Table B-1](#) lists commands associated with the error queue (see [Section 12](#) for more information). For example, the following commands request the next complete error information from the error queue and returns the message portion of the error:

```
errorcode, message, severity, node = errorqueue.next()
print(message)
```

Table B-1  
**Error queue commands**

Error queue command	Description
<code>errorqueue.clear()</code>	Clear error queue of all errors.
<code>errorqueue.count</code>	Number of messages in the error/event queue.
<code>errorqueue.next()</code>	Request error message.

Table B-2  
**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
-440	RECOVERABLE	Query Underminated After Indefinite Response
-430	RECOVERABLE	Query Deadlocked
-420	RECOVERABLE	Query Underminated
-410	RECOVERABLE	Query Interrupted
-363	RECOVERABLE	Input Buffer Over-run
-362	RECOVERABLE	Framing
-361	RECOVERABLE	Parity
-360	RECOVERABLE	Communications
-350	RECOVERABLE	Queue Overflow
-330	RECOVERABLE	Self Test Failed
-315	RECOVERABLE	Configuration Memory Lost
-314	RECOVERABLE	Save/ Recall Memory Lost
-292	RECOVERABLE	Referenced name does not exist
-286	RECOVERABLE	TSP Syntax error, TSP Runtime error, TSP Double error, TSP File error, or TSP Unrecognized error
-285	RECOVERABLE	Program Syntax
-284	RECOVERABLE	Program Currently Running
-282	RECOVERABLE	Illegal Program Name
-281	RECOVERABLE	Cannot Create Program
-260	RECOVERABLE	Expression
-241	RECOVERABLE	Hardware Missing
-230	RECOVERABLE	Data Corrupt or Stale
-225	RECOVERABLE	Out of Memory or TSP Memory allocation error
-224	RECOVERABLE	Illegal Parameter Value
-223	RECOVERABLE	Too Much Data
-222	RECOVERABLE	Parameter Data Out of Range
-221	RECOVERABLE	Settings Conflict
-220	RECOVERABLE	Parameter
-215	RECOVERABLE	Arm Deadlock
-214	RECOVERABLE	Trigger Deadlock
-213	RECOVERABLE	Init Ignored
-212	RECOVERABLE	Arm Ignored

Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
-211	RECOVERABLE	Trigger Ignored
-210	RECOVERABLE	Trigger
-203	RECOVERABLE	Command protected
-202	RECOVERABLE	Settings Lost Due to RTL
-201	RECOVERABLE	Invalid While in Local
-200	RECOVERABLE	Execution
-178	RECOVERABLE	Expression Data Not Allowed
-171	RECOVERABLE	Invalid Expression
-170	RECOVERABLE	Expression
-168	RECOVERABLE	Block Data Not Allowed
-161	RECOVERABLE	Invalid Block Data
-160	RECOVERABLE	Block Data
-158	RECOVERABLE	String Data Not Allowed
-154	RECOVERABLE	String Too Long
-151	RECOVERABLE	Invalid String Data
-150	RECOVERABLE	String Data
-148	RECOVERABLE	Character Data Not Allowed
-144	RECOVERABLE	Character Data Too Long
-141	RECOVERABLE	Invalid Character Data
-140	RECOVERABLE	Character Data
-128	RECOVERABLE	Numeric Data Not Allowed
-124	RECOVERABLE	Too Many Digits
-123	RECOVERABLE	Exponent Too Large
-121	RECOVERABLE	Invalid Character In Number
-120	RECOVERABLE	Numeric Data
-114	RECOVERABLE	Header Suffix Out Of Range
-113	RECOVERABLE	Undefined Header
-112	RECOVERABLE	Program Mnemonic Too Long
-111	RECOVERABLE	Header Separator
-110	RECOVERABLE	Command Header
-109	RECOVERABLE	Missing Parameter

Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
-108	RECOVERABLE	Parameter Not Allowed
-105	RECOVERABLE	Trigger Not Allowed
-104	RECOVERABLE	Data Type
-103	RECOVERABLE	Invalid Separator
-102	RECOVERABLE	Syntax
-101	RECOVERABLE	Invalid Character
-100	RECOVERABLE	Command
0	INFORMATIONAL	No Error
100	INFORMATIONAL	Limit 1 Failed
101	INFORMATIONAL	Low Limit 2 Failed
102	INFORMATIONAL	Low Limit 2 Failed
103	INFORMATIONAL	Low Limit 3 Failed
104	INFORMATIONAL	High Limit 3 Failed
105	INFORMATIONAL	Active Limit Tests
106	INFORMATIONAL	Reading Available
107	INFORMATIONAL	Reading Overflow
108	INFORMATIONAL	Buffer Available
109	INFORMATIONAL	Buffer Full
110	INFORMATIONAL	Limit 4 Failed
111	INFORMATIONAL	Output Interlock Asserted
112	INFORMATIONAL	Temperature Limit Exceeded
113	INFORMATIONAL	Voltage Limit Exceeded
114	INFORMATIONAL	Source In Compliance
200	INFORMATIONAL	Operation Complete
300	INFORMATIONAL	Device Calibration
303	INFORMATIONAL	Device Sweeping
305	INFORMATIONAL	Waiting in Trigger Layer
306	INFORMATIONAL	Waiting in Arm Layer
310	INFORMATIONAL	Entering Idle Layer
408	INFORMATIONAL	Question Calibration
414	INFORMATIONAL	Command Warning

Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
500	RECOVERABLE	Date of Calibration Not Set
501	RECOVERABLE	Next Date of Calibration Not Set
502	RECOVERABLE	Calibration Data Invalid
503	RECOVERABLE	Calibration Overflow
504	RECOVERABLE	Dac Calibration Underflow
505	RECOVERABLE	Source Offset Data Invalid
506	RECOVERABLE	Source Gain Data Invalid
507	RECOVERABLE	Measurement Offset Data Invalid
508	RECOVERABLE	Measurement Gain Data Invalid
509	RECOVERABLE	Not Permitted With Cal Locked
510	RECOVERABLE	Not Permitted With Cal Unlocked
601	RECOVERABLE	Reading Buffer Data Lost
602	RECOVERABLE	GPIB Address Lost
603	RECOVERABLE	Power On State Lost
604	RECOVERABLE	DC Calibration Data Lost
605	RECOVERABLE	Calibration Dates Lost
606	RECOVERABLE	GPIB Communication Language Lost
700	RECOVERABLE	Invalid System Communication
701	RECOVERABLE	ASCII Only With RS232
702	FATAL	Unresponsive digital FPGA
800	RECOVERABLE	Illegal With Storage Active
801	RECOVERABLE	Insufficient Vector Data
802	RECOVERABLE	Output Blocked By Interlock
803	RECOVERABLE	Not Permitted With Output Off
804	RECOVERABLE	Expression List Full
805	RECOVERABLE	Undefined Expression Exists
806	RECOVERABLE	Expression Not Found
807	RECOVERABLE	Definition Not Allowed
808	RECOVERABLE	Expression Cannot Be Deleted
809	RECOVERABLE	Source Memory Location Revised
810	RECOVERABLE	Output Blocked By Over Temperature

Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
811	RECOVERABLE	Not An Operator Or Number
812	RECOVERABLE	Mismatched Parentheses
813	RECOVERABLE	Not a Number of Data Handle
814	RECOVERABLE	Mismatched Brackets
815	RECOVERABLE	Too Many Parentheses
816	RECOVERABLE	Entire Expression Not Parsed
817	RECOVERABLE	Unknown Token
818	RECOVERABLE	Parsing Mantissa
819	RECOVERABLE	Parsing Exponent
820	RECOVERABLE	Parsing Value
821	RECOVERABLE	Invalid Data Handle Index
822	RECOVERABLE	Too Small For Sense Range
823	RECOVERABLE	Invalid With Source Read Back On
824	RECOVERABLE	Cannot Exceed Compliance Range
825	RECOVERABLE	Invalid With Auto Ohms On
826	RECOVERABLE	Attempt to Exceed Power Limit
827	RECOVERABLE	Invalid With Ohms Guard On
828	RECOVERABLE	Invalid On 1 Amp Range
829	RECOVERABLE	Invalid On 1KV Range
830	RECOVERABLE	Invalid With Infinite Arm Count
831	RECOVERABLE	Invalid In Pulse Mode
900	FATAL	Internal System
1100	RECOVERABLE	Command Unavailable
1101	RECOVERABLE	Parameter Too Big
1102	RECOVERABLE	Parameter Too Small
1103	RECOVERABLE	Max Greater Than Min
1104	RECOVERABLE	Too many digits for param type
1105	RECOVERABLE	Too Many Parameters
1106	RECOVERABLE	Battery Not Present
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist

Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
1109	RECOVERABLE	Menu name already exists
1200	RECOVERABLE	TSPlink initialization failed
1201	RECOVERABLE	TSPlink initialization failed
1202	RECOVERABLE	TSPlink initialization failed
1203	RECOVERABLE	TSPlink initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSPlink initialization failed
1205	RECOVERABLE	TSPlink initialization failed (no remote nodes found)
1206	RECOVERABLE	TSPlink initialization failed
1207	RECOVERABLE	TSPlink initialization failed
1208	RECOVERABLE	TSPlink initialization failed
1209	RECOVERABLE	TSPlink initialization failed
1210	RECOVERABLE	TSPlink initialization failed (node ID conflict)
1211	RECOVERABLE	Node %u is inaccessible
1212	RECOVERABLE	Invalid node ID
1400	RECOVERABLE	Expected at least %d parameters
1401	RECOVERABLE	Parameter %d is invalid
1402	RECOVERABLE	User scripts lost
1403	RECOVERABLE	Factory scripts lost
1404	RECOVERABLE	Invalid byte order
1405	RECOVERABLE	Invalid ASCII precision
1406	RECOVERABLE	Invalid data format
1500	RECOVERABLE	Invalid baud rate setting
1501	RECOVERABLE	Invalid parity setting
1502	RECOVERABLE	Invalid terminator setting
1503	RECOVERABLE	Invalid bits setting
1504	RECOVERABLE	Invalid flow control setting
1505	RECOVERABLE	Unknown error
1600	RECOVERABLE	Maximum GPIB message length exceeded
1800	RECOVERABLE	Invalid Digital Trigger Mode
1801	RECOVERABLE	Invalid Digital I/O Line
1802	RECOVERABLE	Digital Bit in Parameter Write Protected



Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
2000	RECOVERABLE	Flash download error
2001	RECOVERABLE	Cannot flash with error in queue
4900	RECOVERABLE	Reading buffer index %s is invalid
4901	RECOVERABLE	The maximum index for this buffer is %d
4902	RECOVERABLE	Reading buffers must be able to contain at least one element
4903	RECOVERABLE	Reading buffer expired
4904	SERIOUS	ICX parameter count mismatch, %s (Line #%d)
4905	SERIOUS	ICX parameter invalid value, %s (Line #%d)
4906	SERIOUS	ICX invalid function id, %s (Line #%d)
5000	FATAL	Bad SMU AFPGA Image size
5001	FATAL	SMU is unresponsive
5002	FATAL	Communication Timeout with DFPGA
5003	SERIOUS	Saved calibration constants corrupted
5004	SERIOUS	Operation conflicts with CALA sense mode
5005	SERIOUS	Value too big for range
5006	SERIOUS	Value too small for range
5007	SERIOUS	Operation would exceed safe operating area of the instrument
5008	SERIOUS	Operation not permitted while output is on
5009	SERIOUS	Unknown sourcing function
5010	SERIOUS	No such SMU function
5011	SERIOUS	Operation not permitted while cal is locked
5012	SERIOUS	Cal data not saved - save or restore before lock
5013	SERIOUS	Cannot save cal data - unlock before save
5014	SERIOUS	Cannot restore cal data - unlock before restore
5015	SERIOUS	Save to cal set disallowed
5016	SERIOUS	Cannot change cal date - unlock before operation
5017	SERIOUS	Cannot change cal constants - unlock before operation
5018	SERIOUS	Cal version inconsistency
5019	SERIOUS	Cannot unlock - invalid password
5020	SERIOUS	Error saving cal data to non-volatile storage
5021	SERIOUS	Cannot restore default calset. Using previous calset

Table B-2 (cont.)

**Error summary**

<b>Error number</b>	<b>Error level</b>	<b>Error Message</b>
5022	SERIOUS	Cannot restore previous calset. Using factory calset
5023	SERIOUS	Cannot restore factory calset. Using nominal calset
5024	SERIOUS	Cannot restore nominal calset. Using firmware defaults
5025	SERIOUS	Cannot set filtercount > 1 when measure.count > 1
5026	SERIOUS	Cannot set measure.count > 1 when filtercount > 1
5027	SERIOUS	Unlock cal data with factory password
5028	SERIOUS	Cannot perform requested operation while source autorange is enabled
5029	SERIOUS	Cannot save without changing cal date and cal due values
5030	SERIOUS	Not implemented yet
5031	RECOVERABLE	Cannot modify built-in reading buffers
5032	RECOVERABLE	Cannot change this setting unless buffer is cleared
5033	RECOVERABLE	Reading buffer not found within device
5034	RECOVERABLE	No readings exist within buffer
5035	RECOVERABLE	Table not found within buffer
5036	RECOVERABLE	Attribute not found
5037	RECOVERABLE	Cannot delete NVRAM Reading Buffer
5038	RECOVERABLE	Index exceeds maximum reading
5039	RECOVERABLE	Measure count exceeds buffer capacity
5040	RECOVERABLE	Cannot use same reading buffer for multiple overlapped measurements
5041	SERIOUS	Output Enable not asserted
5042	SERIOUS	Invalid while overlapped measure
5043	SERIOUS	Cannot perform requested operation while voltage measure autorange is enabled
5044	SERIOUS	Cannot perform requested operation while current measure autorange is enabled
5045	SERIOUS	Cannot perform requested operation while filter is enabled
5046	SERIOUS	SMU too hot
5047	RECOVERABLE	Minimum timestamp resolution is 1 $\mu$ s.

# C Common Commands

---

## **Appendix C topics**

**Introduction**, page C-2

**Common commands**, page C-2

Command summary, page C-2

Script command equivalents, page C-3

Command reference, page C-4

# Introduction

This appendix includes information on common commands and their script command equivalents.

## Common commands

### Command summary

Common commands supported by the Model 260x SourceMeter are summarized in [Table C-1](#). Although commands are shown in upper-case, common commands are not case sensitive, and either upper or lower case can be used. Note that although these commands are essentially the same as those defined by the IEEE-488.2 standard, the Model 260x does not strictly adhere to that standard.

Table C-1

**Common commands**

Mnemonic	Name	Description
*CLS	Clear status	Clears all event registers and Error Queue. <sup>1</sup>
*ESE <mask>	Event enable command	Program the Standard Event Enable Register. <sup>1</sup>
*ESE?	Event enable query	Read the Standard Event Enable Register. <sup>1</sup>
*ESR?	Event status register query	Read/clear the Standard Event Enable Register. <sup>1</sup>
*IDN?	Identification query	Returns the manufacturer, model number, serial number, and firmware revision levels of the unit.
*OPC	Operation complete command	Set the Operation Complete bit in the Standard Event Register after all pending commands have completed.
*OPC?	Operation complete query	Places an ASCII "1" into the Output Queue when all selected device operations have completed.
*RST	Reset command	Returns the SourceMeter to default conditions.
*SRE <mask>	Service request enable command	Programs the Service Request Enable Register. <sup>1</sup>
*SRE?	Service request enable query	Reads the Service Request Enable Register. <sup>1</sup>
*STB?	Status byte query	Reads the Status Byte Register. <sup>1</sup>
*TRG	Trigger command	Sends a remote trigger to the SourceMeter.
*TST?	Self-test query	Returns a 0.
*WAI	Wait-to-continue command	Waits until all previous commands have completed.

1. Status commands are covered in [Appendix D](#).

## Script command equivalents

Script command equivalents for the common commands in [Table C-1](#) are summarized in [Table C-2](#). See Section 12 for details on script commands.

Table C-2

### Script command equivalents

Common command	Script command equivalent
*CLS	status.reset()
*ESE?	print(tostring(status.standard.enable))
*ESE <mask>	status.standard.enable = <mask>
*ESR?	print(tostring(status.standard.event))
*IDN?	print([[Keithley Instruments Inc., Model]].localnode.model.. [[, ]].localnode.serialno..[[, ]].localnode.revision)
*OPC?	waitcomplete() print([[1]])
*OPC	opc()
*RST	reset()
*SRE?	print(tostring(status.request_enable))
*SRE <mask>	status.request_enable = <mask>
*STB?	print(tostring(status.condition))
*TRG	N/A
*TST?	print([[0]])
*WAI	waitcomplete()

## Command reference

Details on all common commands except those associated with the status model are covered below. See [Appendix D](#) for information on using status commands.

**\*IDN? — identification query**                      **Reads ID information**

The identification string includes the manufacturer, model number, serial number, and firmware revision levels and is sent in the following format:

Keithley Instruments Inc., Model nnnn, xxxxxxx, yyyy

Where:    nnnn is the model number (Model 2601 or Model 2602).  
          xxxxxxx is the serial number.  
          yyyyy is the firmware revision level.

**\*OPC — operation complete**                      **Sets OPC bit**

**\*OPC? — operation complete query**              **Places a “1” in output queue**

When \*OPC is sent, the OPC bit in the Standard Event Register (see [Appendix D](#)) will set when all overlapped commands complete. An ASCII “1” is also placed in the Output Queue to be read by the \*OPC? query when overlapped commands complete.

**\*RST — reset**    **Return SourceMeter to defaults**

When the \*RST command is sent, the SourceMeter returns to the default conditions (see [Table 1-3 on page 1-24](#)).

**\*TRG — trigger**    **Send remote trigger to SourceMeter**

Use the \*TRG command to issue a GPIB or RS-232 trigger to the SourceMeter. It has the same effect as a group execute trigger (GET).

**\*TST? — self-test query**                              **Return 0**

This command always places a 0 in the Output Queue. It is included for common command compatibility, but the Model 260x does not actually perform a self-test.

**\*WAI — wait-to-continue****Wait until commands are completed**

Two types of device commands exist:

- Sequential commands — A command whose operations are allowed to finish before the next command is executed.
- Overlapped commands — A command that allows the execution of subsequent commands while device operations of the overlapped command are still in progress.

The \*WAI command is used to suspend the execution of subsequent commands until the device operations of all previous overlapped commands are finished. The \*WAI command is not needed for sequential commands.

# D

# Status Model

---

## Appendix D topics

### Overview, page D-2

Status byte and SRQ, page D-2

Status register sets, page D-2

Queues, page D-2

Status function summary, page D-8

### Clearing registers and queues, page D-9

### Programming and reading registers, page D-10

Programming enable and transition registers, page D-10

Reading registers, page D-11

### Status byte and service request (SRQ),

page D-11

Status byte register, page D-11

Service request enable register, page D-13

Serial polling and SRQ, page D-14

SPE, SPD (serial polling), page D-14

Status byte and service request commands, page D-14

Enable and transition registers, page D-15

Controlling node and SRQ enable registers, page D-15

### Status register sets, page D-17

System Summary Event Registers, page D-17

Standard Event Register, page D-19

Operation Event Registers, page D-21

Questionable Event Registers, page D-24

Measurement Event Registers, page D-26

Register programming example, page D-29

### Queues, page D-29

Output queue, page D-29

Error queue, page D-29



# Overview

The SourceMeter provides a number of status registers and queues allowing the operator to monitor and manipulate the various instrument events. The status model is shown in [Figure D-1](#) through [Figure D-5](#). The heart of the status model is the Status Byte Register. This register can be read by the user's test program to determine if a service request (SRQ) has occurred, and what event caused it.

## Status byte and SRQ

The Status Byte Register receives the summary bits of five status register sets and two queues. The register sets and queues monitor the various instrument events. When an enabled event occurs, it sets a summary bit in the Status Byte Register. When a summary bit of the Status Byte is set and its corresponding enable bit is set (as programmed by the user), the RQS/MSS bit will set to indicate that an SRQ has occurred, and the GPIB SRQ line will be asserted.

## Status register sets

A typical status register set is made up of a condition register, an event register and an event enable register. (Many also have negative and positive transition registers.) A condition register is a read-only register that constantly updates to reflect the present operating conditions of the instrument.

When an event occurs, the appropriate event register bit sets to 1. The bit remains latched to 1 until the register is reset. When an event register bit is set and its corresponding enable bit is set (as programmed by the user), the output (summary) of the register will set to 1. This in turn sets another bit in a lower-level register, and ultimately sets the summary bit of the Status Byte Register.

## Queues

The SourceMeter uses an Output Queue and an Error Queue. The response messages, such as requested readings, are placed in the Output Queue. As various programming errors and status messages occur, they are placed in the Error Queue. When a queue contains data, it sets the appropriate summary bit of the Status Byte Register (EAV for the Error Queue; MAV for the Output Queue).

Figure D-1  
Status model overview

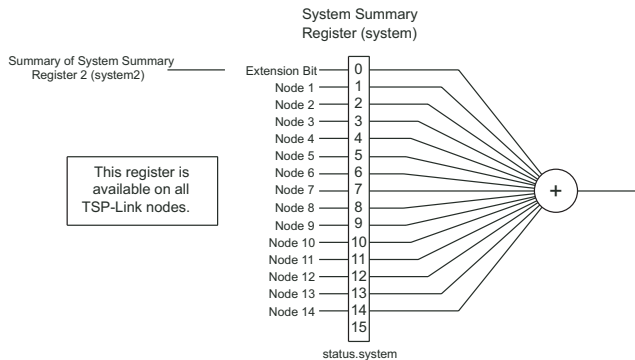
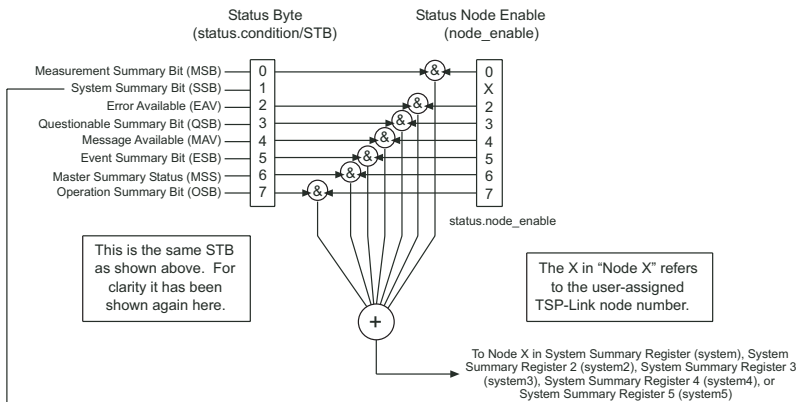
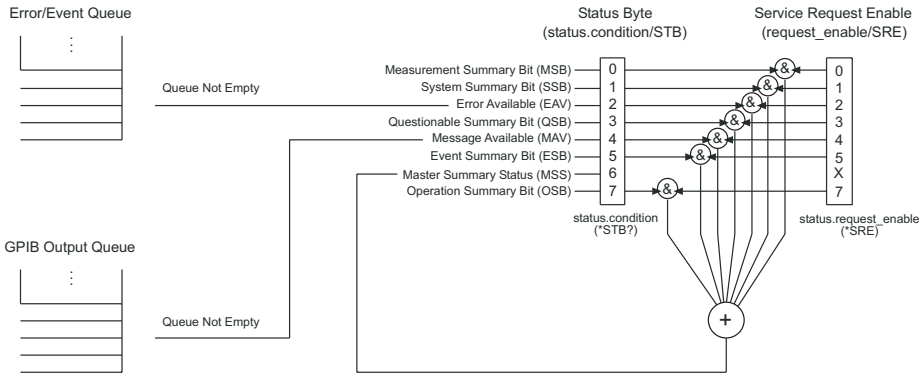


Figure D-2  
**Status model (system summary and standard event registers)**

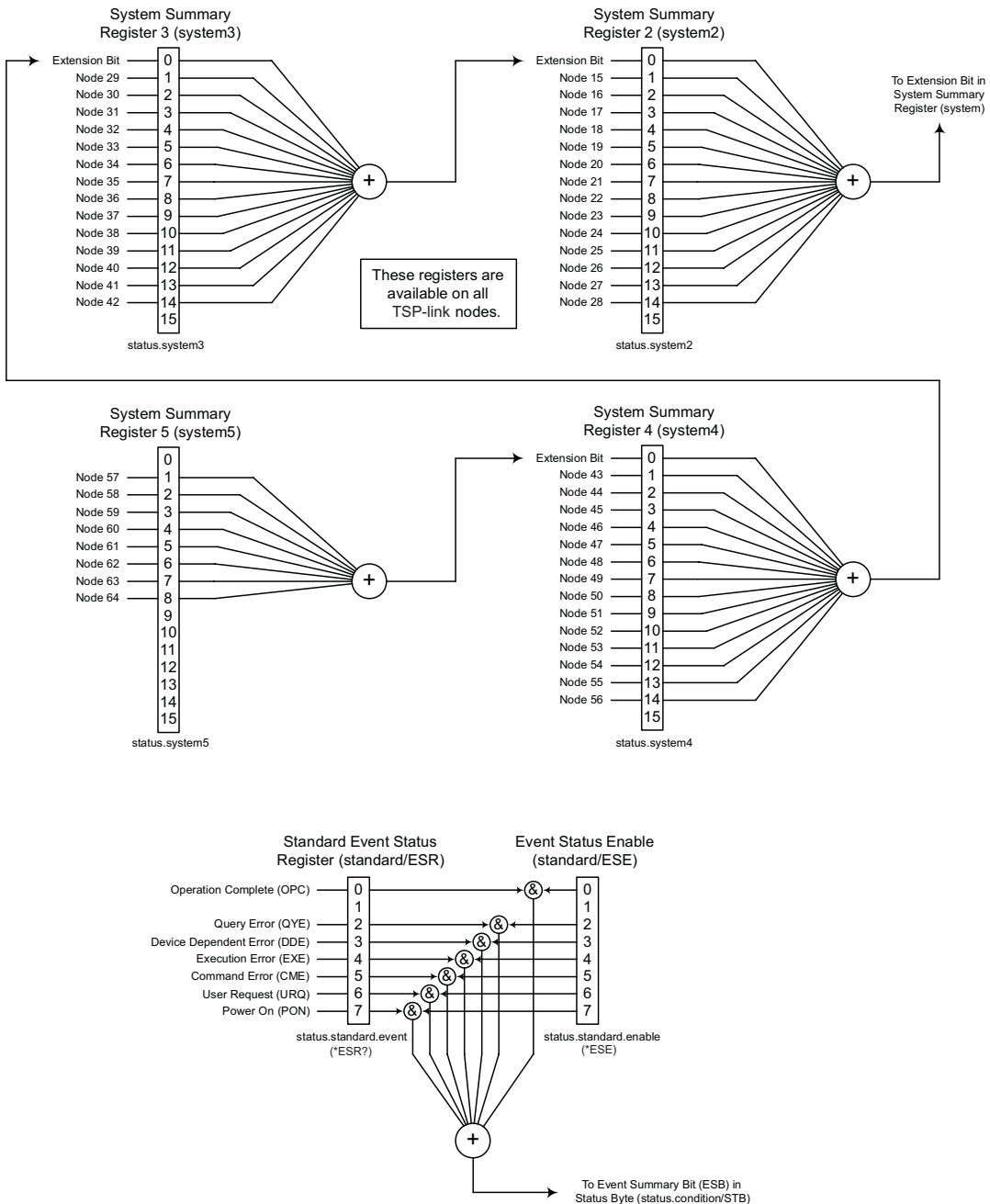


Figure D-3  
**Status model (operation event registers)**

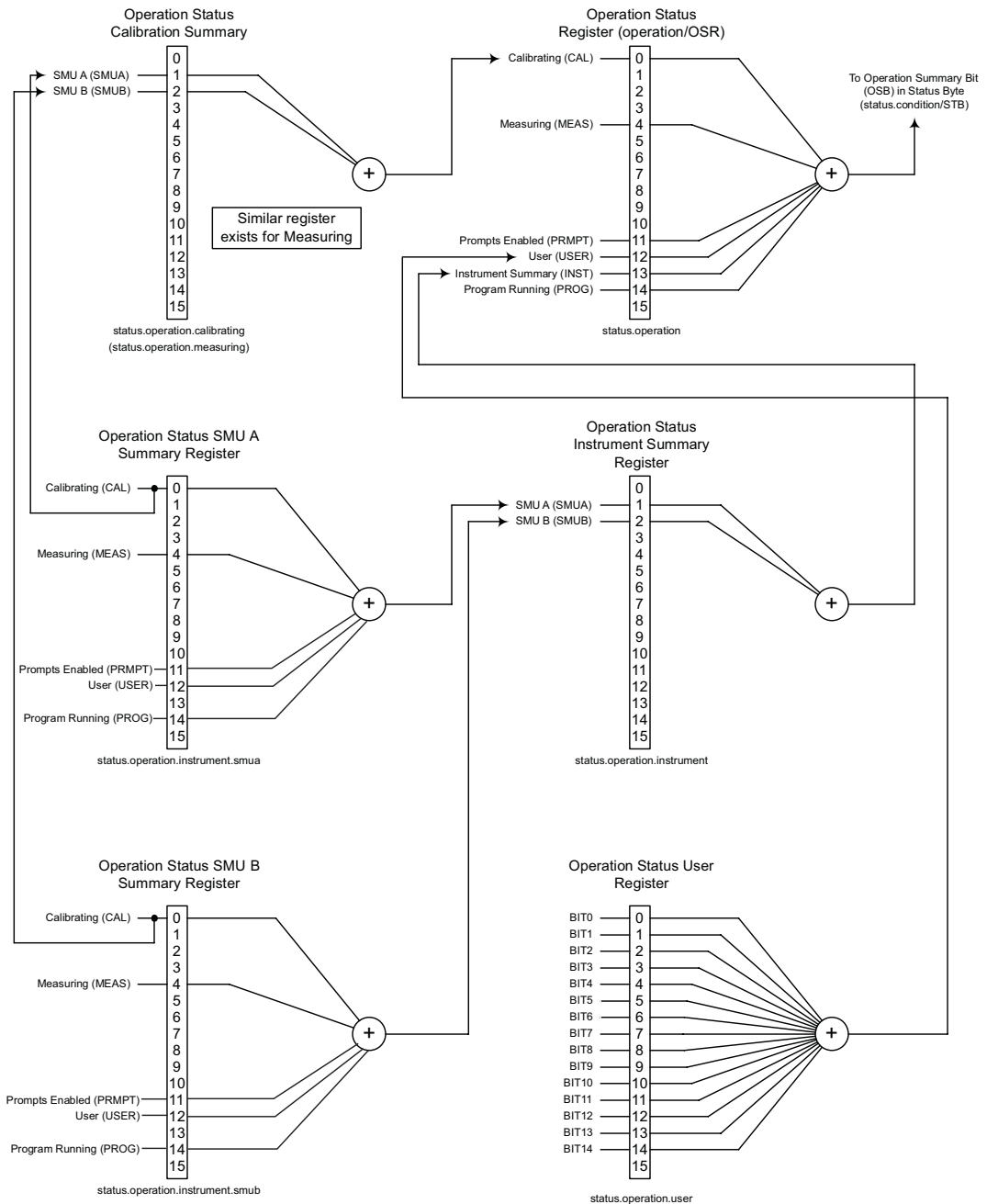


Figure D-4  
**Status model (questionable event registers)**

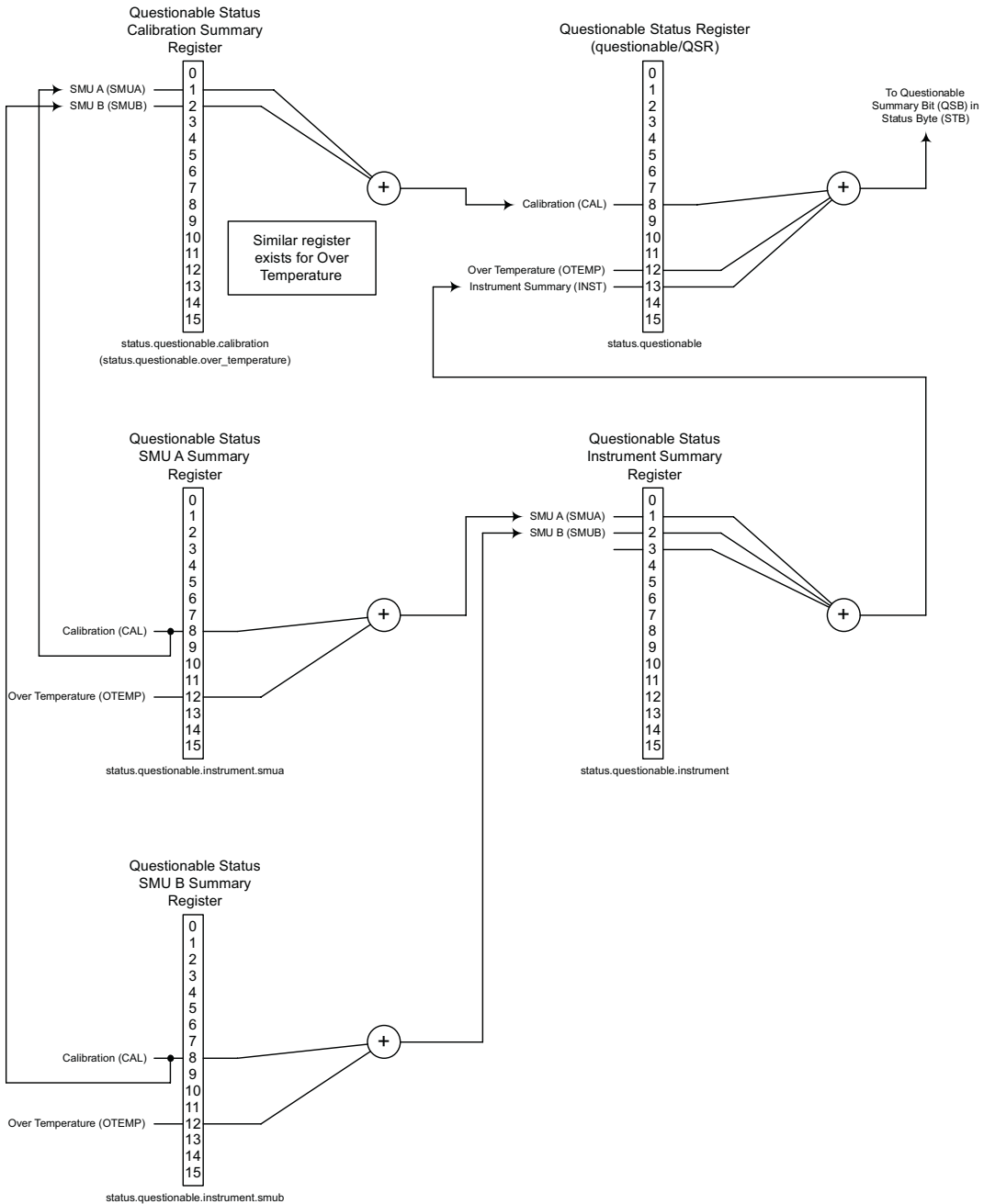
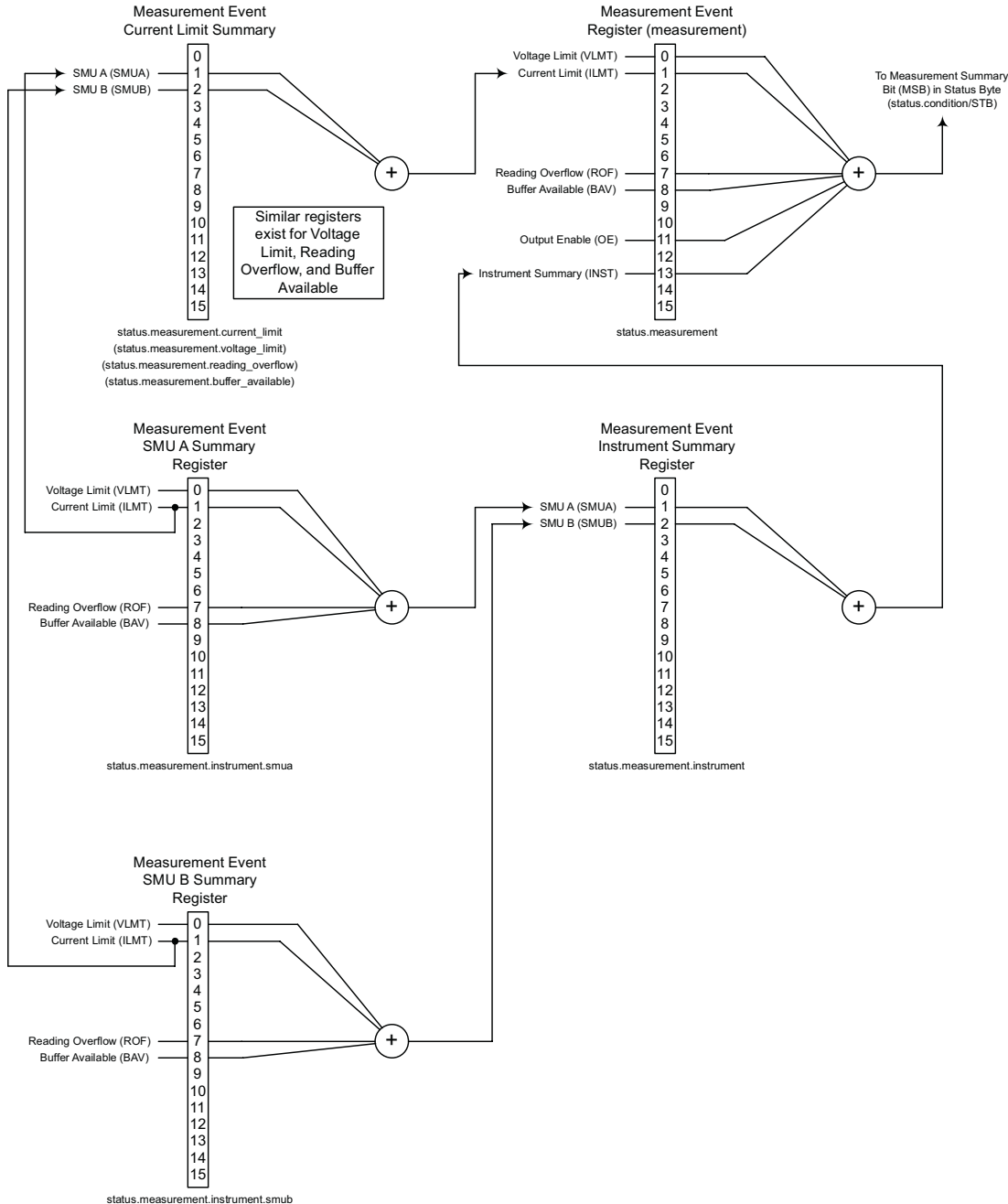


Figure D-5  
Status model (measurement event registers)



## Status function summary

The following functions control and read the various registers. ([Table D-1](#)). Common status commands are listed in [Appendix C](#). Additional information is included later in the section in command listings for the various register sets.

Table D-1

### Status functions and registers

Type	Function <sup>1</sup>	Reference
System summary	status.reset status.node_enable status.request_enable status.node_event status.request_event status.condition	<a href="#">Page D-15</a>
Measurement event	status.measurement.* status.measurement.instrument.smuX.* status.measurement.instrument.* status.measurement.voltage_limit.* status.measurement.current_limit.* status.measurement.buffer_available.* status.measurement.reading_overflow.*	<a href="#">Page D-26</a>
Operation event	status.operation.* status.operation.instrument.smuX.* status.operation.instrument.* status.operation.calibrating.* status.operation.measuring.* status.operation.user.*	<a href="#">Page D-21</a>
Questionable event	status.questionable.* status.questionable.instrument.smuX.* status.questionable.instrument.* status.questionable.calibration.* status.questionable.over_temperature.*	<a href="#">Page D-24</a>
Standard event	status.standard.enable	<a href="#">Page D-19</a>

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

Table D-1 (cont.)

**Status functions and registers**

Type	Function <sup>1</sup>	Reference
System events	status.system.enable status.system2.enable status.system3.enable status.system4.enable status.system5.enable	<a href="#">Page D-17</a>

## Clearing registers and queues

When the SourceMeter is turned on, various register status elements will be set as follows:

- The PON bit in the `status.condition` register will be set.
- Bits such as the output enable and over-temperature bits will be set appropriately.
- All enable registers will be set to 0.
- All NTR registers will be set to 0.
- All used PTR register bits will be set to 1.
- The two queues will be empty.

Commands to reset the status registers and the Error Queue are listed in [Table D-2](#). In addition to these commands, any programmable register can be reset by sending the 0 parameter value with the individual command to program the register.

Table D-2

**Commands to reset registers and clear queues**

Commands	Description
<b>To Reset Registers:</b> *CLS or status.reset() <b>To Clear Error Queue:</b> errorqueue.clear()	Reset bits of status registers to 0.  Clear all messages from Error Queue.



# Programming and reading registers

## Programming enable and transition registers

The only registers that can be programmed by the user are the enable and transition registers. All other registers in the status structure are read-only registers. The following explains how to ascertain the parameter values for the various commands used to program enable registers. The actual commands are summarized in [Appendix C](#) and [Table D-1 on page D-8](#).

A command to program an event enable or transition register is sent with a parameter value that determines the desired state (0 or 1) of each bit in the appropriate register. The bit positions of the register ([Figure D-6](#)) indicate the binary parameter value and decimal equivalent. To program one of the registers, send the decimal value for the bit(s) to be set. The registers are discussed further in [“Enable and transition registers” on page D-15](#), while specific command parameters to program these registers are outlined later in this section.

Figure D-6  
**16-bit status register**

Bit Position	B7	B6	B5	B4	B3	B2	B1	B0
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 <sup>7</sup> )	(2 <sup>6</sup> )	(2 <sup>5</sup> )	(2 <sup>4</sup> )	(2 <sup>3</sup> )	(2 <sup>2</sup> )	(2 <sup>1</sup> )	(2 <sup>0</sup> )

### A. Bits 0 through 7

Bit Position	B15	B14	B13	B12	B11	B10	B9	B8
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32768	16384	8192	4096	2048	1024	512	256
Weights	(2 <sup>15</sup> )	(2 <sup>14</sup> )	(2 <sup>13</sup> )	(2 <sup>12</sup> )	(2 <sup>11</sup> )	(2 <sup>10</sup> )	(2 <sup>9</sup> )	(2 <sup>8</sup> )

### B. Bits 8 through 15

When using a numeric parameter, registers are programmed by including the appropriate `<mask>` value, for example:

```
*ese <mask>
status.standard.enable = <mask>
```

To convert from decimal to binary, use the information shown in [Figure D-6](#). For example, to set bits B0, B4, B7, and B10, a decimal value of 1169 would be used for the mask parameter ( $1169 = 1 + 16 + 128 + 1024$ ).

## Reading registers

Any register in the status structure can be read either by sending the common command query (where applicable), or by including the script command for that register in either the `print()` or `print(tostring())` command. The `print()` command returns a numeric value, while the `print(tostring())` command returns the string equivalent. For example, any of the following commands requests the Service Request Enable register value:

```
*SRE?  
print(tostring(status.request_enable))  
print(status.request_enable)
```

The response message will be a decimal value that indicates which bits in the register are set. That value can be converted to its binary equivalent using [Figure D-6](#). For example, for a decimal value of 37 (binary value of 100101), bits B5, B2, and B0 are set.

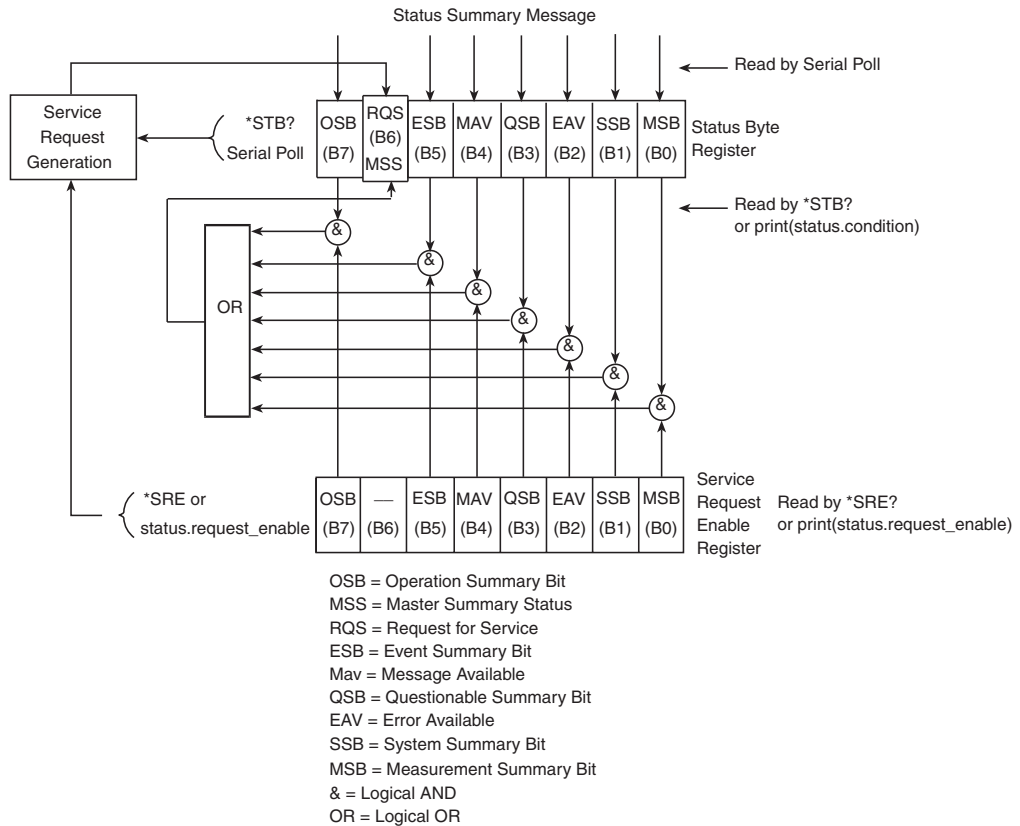
## Status byte and service request (SRQ)

Service request is controlled by two 8-bit registers; the Status Byte Register and the Service Request Enable Register. [Figure D-7](#) shows the structure of these registers.

### Status byte register

The summary messages from the status registers and queues are used to set or clear the appropriate bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are 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.

Figure D-7  
**Status byte and service request (SRQ)**



The bits of the Status Byte Register are described as follows:

- **Bit B0, Measurement Summary Bit (MSB)** — Set summary bit indicates that an enabled measurement event has occurred.
- **Bit B1, System Summary Bit (SSB)** — Set summary bit indicates that an enabled system event has occurred.
- **Bit B2, Error Available (EAV)** — Set bit indicates that an error or status message is present in the Error Queue.
- **Bit B3, Questionable Summary Bit (QSB)** — Set summary bit indicates that an enabled questionable event has occurred.
- **Bit B4, Message Available (MAV)** — Set bit indicates that a response message is present in the Output Queue.
- **Bit B5, Event Summary Bit (ESB)** — Set summary bit indicates that an enabled standard event has occurred.
- **Bit B6, Request Service (RQS)/Master Summary Status (MSS)** — Set bit indicates that an enabled summary bit of the Status Byte Register is set. Depending on how it is used, Bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit:
  - When using the GPIB serial poll sequence of the SourceMeter to obtain the status byte (serial poll byte), B6 is the RQS bit. See [“Serial polling and SRQ”](#) for details on using the serial poll sequence.
  - When using the `*STB?` common command or `status.condition` ([Table D-3 on page D-15](#)) to read the status byte, B6 is the MSS bit.
- **Bit B7, Operation Summary (OSB)** — Set summary bit indicates that an enabled operation event has occurred.

## Service request enable register

The generation of a service request is controlled by the 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, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in [Figure D-7](#), the summary bits are logically ANDed (&) with the corresponding enable bits of the Service Request Enable Register. When a set (1) summary bit is ANDed with an enabled (1) bit of the enable register, the logic “1” output is applied to the input of the 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 its script equivalent. To read the Service Request Enable Register, use the `*SRE?` query or script equivalent. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with the `*SRE` command (i.e. `*SRE 0`). The commands to program and read the SRQ Enable Register are listed in [Table D-3 on page D-15](#).

## Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 will set bit B6 and generate an SRQ (service request). 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 SourceMeter. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register will remain cleared, and the program will simply proceed normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register will set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence generated by other event types.

For common and script commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear MSS. The MSS bit stays set until all Status Byte summary bits are reset.

## SPE, SPD (serial polling)

For the GPIB interface only, the SPE, SPD General Bus Command sequence is used to serial poll the SourceMeter (see [“General bus commands” on page 11-7](#)). Serial polling obtains the serial poll byte (status byte). Typically, serial polling is used by the controller to determine which of several instruments has requested service with the SRQ line.

## Status byte and service request commands

The commands to program and read the Status Byte Register and Service Request Enable Register are listed in [Table D-3](#). Note that the table includes both common commands and their script command equivalents. For details on programming and reading registers, see [“Programming enable and transition registers” on page D-10](#), and [“Reading registers” on page D-11](#).

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).

Table D-3  
**Status Byte and Service Request Enable Register commands**

Command	Description
*STB? or print(status.condition)	Read Status Byte Register.
*SRE <mask> or status.request_enable = <mask>	Program the Service Request Enable Register:  <mask> = 0 to 255
*SRE? or print(status.request_enable)	Read the Service Request Enable Register.

## Enable and transition registers

In general, there are three types of user-writable registers that are used to configure which bits feed the register summary and when it occurs. The registers are identified in the command table footnotes as follows:

- Enable register (identified as “enable” in the table footnotes): allows various associated events to be included in the summary bit for the register.
- Negative-transition register (NTR; identified as “ntr” in the table footnotes): a particular bit in the event register will be set when the corresponding bit in the NTR is set, and the corresponding bit in the condition register transitions from 1 to 0.
- Positive-transition register (PTR; identified as “ptr” in the table footnotes): a particular bit in the event register will be set when the corresponding bit in the PTR is set, and the corresponding bit in the condition register transitions from 0 to 1.

## Controlling node and SRQ enable registers

Attributes to control system node and SRQ enable bits and read associated registers are summarized in [Table D-4](#). For example, either of the following will set the system node MSB enable bit:

```
status.node_enable = status.MSB
status.node_enable = 1
```

Table D-4

**System node and SRQ enable register bit attributes**

Attribute	Description	Bit
<b>To set system node enable register bits:</b>		
status.node_enable = status.MEASUREMENT_SUMMARY_BIT	Enable MSB.	B0
status.node_enable = status.MSB	Enable MSB.	B0
status.node_enable = status.ERROR_AVAILABLE	Enable EAV bit.	B2
status.node_enable = status.EAV	Enable EAV bit.	B2
status.node_enable = status.QUESTIONABLE_SUMMARY_BIT	Enable QSB.	B3
status.node_enable = status.QSB	Enable QSB.	B3
status.node_enable = status.MESSAGE_AVAILABLE	Enable MAV bit.	B4
status.node_enable = status.MAV	Enable MAV bit.	B4
status.node_enable = status.EVENT_SUMMARY_BIT	Enable ESB.	B5
status.node_enable = status.ESB	Enable ESB.	B5
status.node_enable = status.MASTER_SUMMARY_STATUS	Enable MSS bit.	B6
status.node_enable = status.MSS	Enable MSS bit.	B6
status.node_enable = status.OPERATION_SUMMARY_BIT	Enable OSB.	B7
status.node_enable = status.OSB	Enable OSB.	B7
<b>To set service request enable register bits:</b>		
status.request_enable = status.MEASUREMENT_SUMMARY_BIT	Enable MSB.	B0
status.request_enable = status.MSB	Enable MSB.	B0
status.request_enable = status.SYSTEM_SUMMARY_BIT	Enable SSB.	B1
status.request_enable = status.SSB	Enable SSB.	B1
status.request_enable = status.ERROR_AVAILABLE	Enable EAV bit.	B2
status.request_enable = status.EAV	Enable EAV bit.	B2
status.request_enable = status.QUESTIONABLE_SUMMARY_BIT	Enable QSB.	B3
status.request_enable = status.QSB	Enable QSB.	B3
status.request_enable = status.MESSAGE_AVAILABLE	Enable MAV bit.	B4
status.request_enable = status.MAV	Enable MAV bit.	B4
status.request_enable = status.EVENT_SUMMARY_BIT	Enable ESB.	B5
status.request_enable = status.ESB	Enable ESB.	B5
status.request_enable = status.OPERATION_SUMMARY_BIT	Enable OSB.	B7
status.request_enable = status.OSB	Enable OSB.	B7
<b>To read registers:</b>		
print(status.node_enable)	Request system enable register. Request SRQ enable register. Request status byte register.	
print(status.request_enable)		
print(status.condition)		

# Status register sets

As shown in [Figure D-1](#) through [Figure D-5](#), there are five status register sets in the status structure of the SourceMeter; System Summary Event Status, Standard Event Status, Operation Event Status, Measurement Event Status, and Questionable Event Status.

## System Summary Event Registers

As shown in [Figure D-1 on page D-3](#) and [Figure D-2 on page D-4](#), there are five register sets associated with System Event Status. These registers summarize system status for various nodes connected to the TSP-Link (see Section 9). Note that all nodes on the TSP-Link share a copy of the system summary registers once the TSP-Link has been initialized. This feature allows all nodes to access the status models of other nodes, including SRQ.

Commands are summarized in [Table D-5](#).

For example, either of the following commands will set the EXT enable bit:

```
status.system.enable = status.system.EXT
status.system.enable = 1
```

The following command will request the system enable register value:

```
print(status.system.enable)
```

The used bits of the System Event Registers are described as follows:

- **EXT** — Set bit indicates that an extension bit from a another system status register is set.
- **NODEn** — Indicates a bit on TSP-Link node n has been set (n =1 to 64).

Table D-5

### System summary event commands

Command <sup>1</sup>	Bit <sup>1</sup>
<b>To set register bits:</b> status.system.enable = status.system.EXTENSION_BIT status.system.enable = status.system.EXT status.system.enable = status.system.NODEn <b>To read registers:</b> print(status.system.enable) print(status.system.condition) print(status.system.event)	B0 B0 Bn

1. n = 1 through 14 for status.system; 15 through 28 for status.system2; 29 through 42 for status.system3; 43 through 56 for status.system4; 57 through 64 for status.system5.



Table D-5 (cont.)

**System summary event commands**

Command <sup>1</sup>	Bit <sup>1</sup>
<b>To set register bits:</b> status.system2.enable = status.system2.EXTENSION_BIT status.system2.enable = status.system2.EXT status.system2.enable = status.system2.NODEn <b>To read registers:</b> print(status.system2.enable) print(status.system2.condition) print(status.system2.event)	B0 B0 Bn
<b>To set register bits:</b> status.system3.enable = status.system3.EXTENSION_BIT status.system3.enable = status.system3.EXT status.system3.enable = status.system3.NODEn <b>To read registers:</b> print(status.system3.enable) print(status.system3.condition) print(status.system3.event)	B0 B0 Bn
<b>To set register bits:</b> status.system4.enable = status.system4.EXTENSION_BIT status.system4.enable = status.system4.EXT status.system4.enable = status.system4.NODEn <b>To read registers:</b> print(status.system4.enable) print(status.system4.condition) print(status.system4.event)	B0 B0 Bn
<b>To set register bits:</b> status.system5.enable = status.system5.EXTENSION_BIT status.system5.enable = status.system5.EXT status.system5.enable = status.system5.NODEn <b>To read registers:</b> print(status.system5.enable) print(status.system5.condition) print(status.system5.event)	B0 B0 Bn

1. n = 1 through 14 for status.system; 15 through 28 for status.system2; 29 through 42 for status.system3; 43 through 56 for status.system4; 57 through 64 for status.system5.

## Standard Event Register

The bits used in the Standard Event Register (shown in [Figure D-8](#)) are described as follows:

- **Bit B0, Operation Complete** — Set bit indicates that all pending selected device operations are completed and the SourceMeter is ready to accept new commands. The bit is set in response to an \*OPC command. The ICL function `opc()` can be used in place of the \*OPC command. See [Appendix C](#) for details on \*OPC.
- **Bit B1** — Not used.
- **Bit B2, Query Error (QYE)** — Set bit indicates that you attempted to read data from an empty Output Queue.
- **Bit B3, Device-Dependent Error (DDE)** — Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, Execution Error (EXE)** — Set bit indicates that the SourceMeter detected an error while trying to execute a command.
- **Bit B5, Command Error (CME)** — Set bit indicates that a command error has occurred. Command errors include:
  - IEEE-488.2 syntax error — SourceMeter received a message that does not follow the defined syntax of the IEEE-488.2 standard.
  - Semantic error — SourceMeter received a command that was misspelled or received an optional IEEE-488.2 command that is not implemented.
  - The instrument received a Group Execute Trigger (GET) inside a program message.
- **Bit B6, User Request (URQ)** — Set bit indicates that the LOCAL key on the SourceMeter front panel was pressed.
- **Bit B7, Power ON (PON)** — Set bit indicates that the SourceMeter has been turned off and turned back on since the last time this register has been read.

Commands to program and read the register are summarized in [Table D-6](#), and bits are summarized in [Table D-7](#).

Figure D-8  
Standard event register

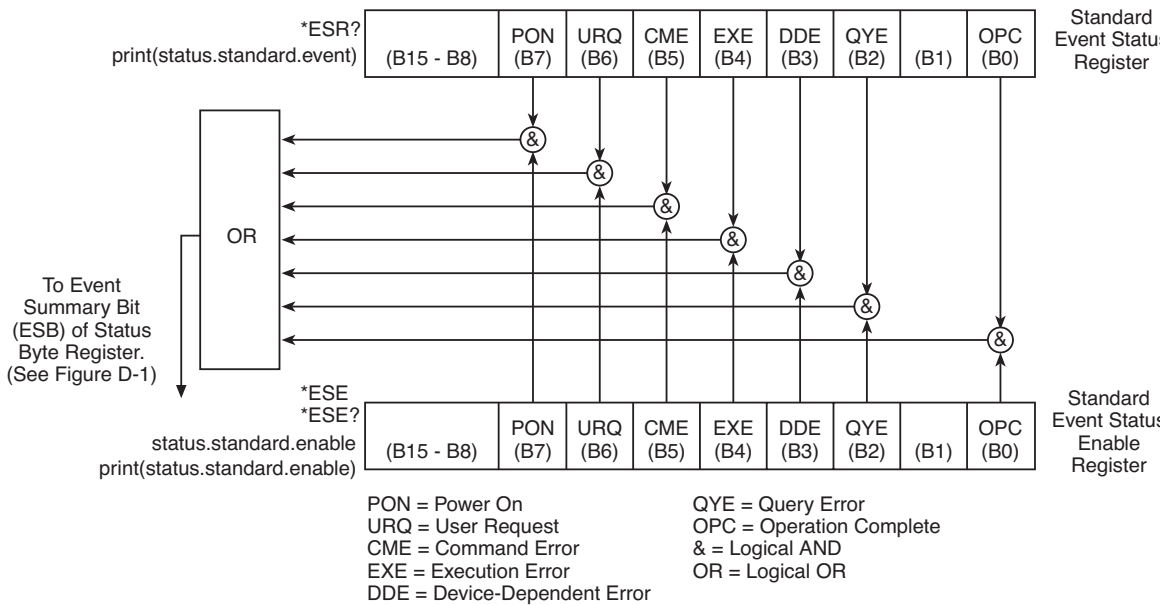


Table D-6  
Standard event commands

Command	Description
*ESR? or print(status.standard.event)	Read Standard Event Status Register.
*ESE <mask> or status.standard.enable = <mask>	Program the Event Status Enable Register: <mask> = 0 to 255 See <a href="#">Table D-7</a> .
*ESE? or print(status.standard.enable)	Read Event Status Enable Register.

Table D-7  
**Status event status registers and bits**

Command	Bit
<b>To set register bits:</b>	
status.standard.enable = status.standard.OPERATION_COMPLETE	B0
status.standard.enable = status.standard.OPC	B0
status.standard.enable = status.standard.QUERY_ERROR	B2
status.standard.enable = status.standard.QYE	B2
status.standard.enable = status.standard.DEVICE_DEPENDENT_ERROR	B3
status.standard.enable = status.standard.DDE	B3
status.standard.enable = status.standard.EXECUTION_ERROR	B4
status.standard.enable = status.standard.EXE	B4
status.standard.enable = status.standard.COMMAND_ERROR	B5
status.standard.enable = status.standard.CME	B5
status.standard.enable = status.standard.USER_REQUEST	B6
status.standard.enable = status.standard.URQ	B6
status.standard.enable = status.standard.POWER_ON	B7
status.standard.enable = status.standard.PON	B7
<b>To read registers:</b>	
print(status.standard.enable)	
print(status.standard.condition)	
print(status.standard.event)	

## Operation Event Registers

As shown in [Figure D-3 on page D-5](#), there are seven register sets associated with Operation Event Status. Commands are summarized in [Table D-8](#). For example, either of the following commands will set the CAL enable bit:

```
status.operation.enable = status.operation.CAL
status.operation.enable = 1
```

The bits used in the Operation Event Registers are described as follows:

- **CAL** — Set bit indicates that one or more channels are calibrating.
- **MEAS** — Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement.
- **PRMPT** — Set bit indicates that command prompts are enabled.
- **USER** — Set bit indicates that an enabled bit in the operation status user register is set.
- **INST** — Set bit indicates that an enabled bit in the operation status instrument summary register is set.
- **PROG** — Set bit indicates that a program is running.

Table D-8

**Operation event commands**

Command <sup>1</sup>	Bit
<p><b>To set register bits:</b>  status.operation.* = status.operation.CALIBRATING  status.operation.* = status.operation.CAL  status.operation.* = status.operation.MEASURING  status.operation.* = status.operation.MEAS  status.operation.* = status.operation.PROMPTS  status.operation.* = status.operation.PRMPPTS  status.operation.* = status.operation.USER  status.operation.* = status.operation.INSTRUMENT_SUMMARY  status.operation.* = status.operation.INST  status.operation.* = status.operation.PROGRAM_RUNNING  status.operation.* = status.operation.PROG</p> <p><b>To read registers:</b>  print(status.operation.*)  print(status.operation.condition)  print(status.operation.event)</p>	B0 B0 B4 B4 B11 B11 B12 B13 B13 B14 B14
<p><b>To set register bits:</b>  status.operation.instrument.smuX.* = status.operation.CALIBRATING  status.operation.instrument.smuX.* = status.operation.CAL  status.operation.instrument.smuX.* = status.operation.MEASURING  status.operation.instrument.smuX.* = status.operation.MEAS  status.operation.instrument.smuX.* = status.operation.PROMPTS  status.operation.instrument.smuX.* = status.operation.PRMPPTS  status.operation.instrument.smuX.* = status.operation.USER  status.operation.instrument.smuX.* = status.operation.PROGRAM_RUNNING  status.operation.instrument.smuX.* = status.operation.PROG</p> <p><b>To read registers:</b>  print(status.operation.instrument.smuX.*)  print(status.operation.instrument.smuX.condition)  print(status.operation.instrument.smuX.event)</p>	B0 B0 B4 B4 B11 B11 B12 B14 B14
<p><b>To set register bits:</b>  status.operation.instrument.* = status.operation.instrument.SMUA  status.operation.instrument.* = status.operation.instrument.SMUB</p> <p><b>To read registers:</b>  print(status.operation.instrument.*)  print(status.operation.instrument.condition)  print(status.operation.instrument.event)</p>	B1 B2

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

Table D-8 (cont.)

**Operation event commands**

<b>Command<sup>1</sup></b>	<b>Bit</b>
<b>To set register bits:</b> status.operation.calibrating.* = status.operation.calibrating.SMUA status.operation.calibrating.* = status.operation.calibrating.SMUB <b>To read registers:</b> print(status.operation.calibrating.*) print(status.operation.calibrating.condition) print(status.operation.calibrating.event)	B1 B2
<b>To set register bits:</b> status.operation.measuring.* = status.operation.measuring.SMUA status.operation.measuring.* = status.operation.measuring.SMUB <b>To read registers:</b> print(status.operation.measuring.*) print(status.operation.measuring.condition) print(status.operation.measuring.event)	B1 B2
<b>To set register bits:</b> status.operation.user.* = status.operation.user.BIT0 status.operation.user.* = status.operation.user.BIT1 status.operation.user.* = status.operation.user.BIT2 status.operation.user.* = status.operation.user.BIT3 status.operation.user.* = status.operation.user.BIT4 status.operation.user.* = status.operation.user.BIT5 status.operation.user.* = status.operation.user.BIT6 status.operation.user.* = status.operation.user.BIT7 status.operation.user.* = status.operation.user.BIT8 status.operation.user.* = status.operation.user.BIT9 status.operation.user.* = status.operation.user.BIT10 status.operation.user.* = status.operation.user.BIT11 status.operation.user.* = status.operation.user.BIT12 status.operation.user.* = status.operation.user.BIT13 status.operation.user.* = status.operation.user.BIT14 <b>To read registers:</b> print(status.operation.user.*) print(status.operation.user.condition) print(status.operation.user.enable)	B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

## Questionable Event Registers

As shown in [Figure D-4 on page D-6](#), there are six register sets associated with Questionable Event Status. Commands are summarized in [Table D-9](#).

For example, either of the following commands will set the CAL enable bit:

```
status.questionable.enable = status.questionable.CAL
status.questionable.enable = 256
```

The following command will request the questionable enable register value:

```
print(status.questionable.enable)
```

The bits used in the Questionable Event Registers are described as follows:

- **CAL** — Set bit indicates that calibration is questionable.
- **OTEMP** — Set bit indicates that an over temperature condition was detected.
- **INST** — Set bit indicates that a bit in the questionable instrument summary register is set.

Table D-9

### Questionable event commands

Command <sup>1</sup>	Bit
<b>To set register bits:</b> status.questionable.* = status.questionable.CALIBRATION status.questionable.* = status.questionable.CAL status.questionable.* = status.questionable.OVER_TEMPERATURE status.questionable.* = status.questionable.OTEMP status.questionable.* = status.questionable.INSTRUMENT_SUMMARY status.questionable.* = status.questionable.INST <b>To read registers:</b> print(status.questionable.*) print(status.questionable.condition) print(status.questionable.event)	B8 B8 B12 B12 B13 B13
<b>To set register bits:</b> status.questionable.instrument.smuX.* = status.questionable.CALIBRATION status.questionable.instrument.smuX.* = status.questionable.CAL status.questionable.instrument.smuX.* = status.questionable.OVER_TEMPERATURE status.questionable.instrument.smuX.* = status.questionable.OTEMP <b>To read registers:</b> print(status.questionable.instrument.smuX.*) print(status.questionable.instrument.smuX.condition) print(status.questionable.instrument.smuX.event)	B8 B8 B12 B12

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

Table D-9 (cont.)

**Questionable event commands**

Command <sup>1</sup>	Bit
<b>To set register bits:</b> status.questionable.instrument.* = status.questionable.instrument.SMUA status.questionable.instrument.* = status.questionable.instrument.SMUB <b>To read registers:</b> print(status.questionable.instrument.*) print(status.questionable.instrument.condition) print(status.questionable.instrument.event)	B1 B2
<b>To set register bits:</b> status.questionable.calibration.* = status.questionable.calibration.SMUA status.questionable.calibration.* = status.questionable.calibration.SMUB <b>To read registers:</b> print(status.questionable.calibration.*) print(status.questionable.calibration.condition) print(status.questionable.calibration.event)	B1 B2
<b>To set register bits:</b> status.questionable.over_temperature.* = status.questionable.over_temperature.SMUA status.questionable.over_temperature.* = status.questionable.over_temperature.SMUB <b>To read registers:</b> print(status.questionable.over_temperature.*) print(status.questionable.over_temperature.condition) print(status.questionable.over_temperature.event)	B1 B2

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.



## Measurement Event Registers

As shown in [Figure D-5 on page D-7](#), there are seven register sets associated with Measurement Event Status. Commands are summarized in [Table D-10](#).

For example, either of the following commands will set the `VOLTAGE_LIMIT` enable bit:

```
status.measurement.enable = status.measurement.VOLTAGE_LIMIT
status.measurement.enable = 1
```

The bits used in the Measurement Event Registers are described as follows:

- **VLMT** — Set bit indicates that the voltage limit was exceeded. This bit will be updated only when (1) a measurement is taken or (2) the `smuX.source.compliance` command is invoked.
- **ILMT** — Set bit indicates that the current limit was exceeded. This bit will be updated only when (1) a measurement is taken or (2) the `smuX.source.compliance` command is invoked.
- **ROF** — Set bit indicates that an overflow reading has been detected.
- **BAV** — Set bit indicates that there is at least one reading stored in either or both of the non-volatile reading buffers.
- **OE** — Set bit indicates that output enable has been asserted.
- **INST** — Set bit indicates that a bit in the measurement instrument summary register is set.

Table D-10

**Measurement event commands**

Command <sup>1</sup>	Bit
<p><b>To set register bits:</b>  status.measurement.* = status.measurement.VOLTAGE_LIMIT  status.measurement.* = status.measurement.VLMT  status.measurement.* = status.measurement.CURRENT_LIMIT  status.measurement.* = status.measurement.ILMT  status.measurement.* = status.measurement.READING_OVERFLOW  status.measurement.* = status.measurement.ROF  status.measurement.* = status.measurement.BUFFER_AVAILABLE  status.measurement.* = status.measurement.BAV  status.measurement.* = status.measurement.OUTPUT_ENABLE  status.measurement.* = status.measurement.OE  status.measurement.* = status.measurement.INSTRUMENT_SUMMARY  status.measurement.* = status.measurement.INST</p> <p><b>To read registers:</b>  print(status.measurement.*)  print(status.measurement.condition)  print(status.measurement.event)</p>	B0 B0 B1 B1 B7 B7 B8 B8 B11 B11 B13 B13
<p><b>To set register bits:</b>  status.measurement.instrument.smuX.* = status.measurement.VOLTAGE_LIMIT  status.measurement.instrument.smuX.* = status.measurement.VLMT  status.measurement.instrument.smuX.* = status.measurement.CURRENT_LIMIT  status.measurement.instrument.smuX.* = status.measurement.ILMT  status.measurement.instrument.smuX.* = status.measurement.READING_OVERFLOW  status.measurement.instrument.smuX.* = status.measurement.ROF  status.measurement.instrument.smuX.* = status.measurement.BUFFER_AVAILABLE  status.measurement.instrument.smuX.* = status.measurement.BAV</p> <p><b>To read registers:</b>  print(status.measurement.instrument.smuX.*)  print(status.measurement.instrument.smuX.condition)  print(status.measurement.instrument.smuX.event)</p>	B0 B0 B1 B1 B7 B7 B8 B8
<p><b>To set register bits:</b>  status.measurement.instrument.* = status.measurement.instrument.SMUA  status.measurement.instrument.* = status.measurement.instrument.SMUB</p> <p><b>To read registers:</b>  print(status.measurement.instrument.*)  print(status.measurement.instrument.condition)  print(status.measurement.instrument.event)</p>	B1 B2

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

Table D-10 (cont.)

**Measurement event commands**

Command <sup>1</sup>	Bit
<b>To set register bits:</b> status.measurement.voltage_limit.* = status.measurement.voltage_limit.SMUA status.measurement.voltage_limit.* = status.measurement.voltage_limit.SMUB <b>To read registers:</b> print(status.measurement.voltage_limit.*) print(status.measurement.voltage_limit.condition) print(status.measurement.voltage_limit.event)	B1 B2
<b>To set register bits:</b> status.measurement.current_limit.* = status.measurement.current_limit.SMUA status.measurement.current_limit.* = status.measurement.current_limit.SMUB <b>To read registers:</b> print(status.measurement.current_limit.*) print(status.measurement.current_limit.condition) print(status.measurement.current_limit.event)	B1 B2
<b>To set register bits:</b> status.measurement.buffer_available.* = status.measurement.buffer_available.SMUA status.measurement.buffer_available.* = status.measurement.buffer_available.SMUB <b>To read registers:</b> print(status.measurement.buffer_available.*) print(status.measurement.buffer_available.condition) print(status.measurement.buffer_available.event)	B1 B2
<b>To set register bits:</b> status.measurement.reading_overflow.* = status.measurement.reading_overflow.SMUA status.measurement.reading_overflow.* = status.measurement.reading_overflow.SMUB <b>To read registers:</b> print(status.measurement.reading_overflow.*) print(status.measurement.reading_overflow.condition) print(status.measurement.reading_overflow.event)	B1 B2

1. \* = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

## Register programming example

The command sequence below programs the instrument to generate an SRQ and set the system summary bit to TSP-Link nodes when the current limit on Channel A is exceeded.

```
status.reset()                --Clear all registers.
status.measurement.current_limit.enable --Enable current limit bit in
= status.measurement.current_limit.SMUA  current limit register.
status.measurement.enable =           --Enable status measure
status.measurement.ILMT              current limit bit.
status.node_enable = status.MSB       --Set system summary;
                                     enable MSB.
status.node_enable = status.MSB       --Enable status SRQ MSB.
```

## Queues

The SourceMeter uses two queues, which are first-in, first-out (FIFO) queues:

- Output Queue — Used to hold response messages.
- Error Queue — Used to hold error and status messages. (See [Table B-2 on page B-3.](#))

The SourceMeter status model ([Figure D-1 on page D-3](#)) shows how the two queues are structured with the other registers.

### Output queue

The output queue holds data that pertains to the normal operation of the instrument. For example, when a `print` command is sent, the response message is placed in the Output Queue.

When data is placed in the Output Queue, the Message Available (MAV) bit in the Status Byte Register sets. A response message is cleared from the Output Queue when it is read. The Output Queue is considered cleared when it is empty. An empty Output Queue clears the MAV bit in the Status Byte Register.

A message is read from the Output Queue by addressing the SourceMeter to talk.

### Error queue

The Error Queue holds error and status messages. When an error or status event occurs, a message that defines the error or status is placed in the Error Queue.

When a message is placed in the Error Queue, the Error Available (EAV) bit in the Status Byte Register is set. An error or status message is cleared from the Error

Queue when it is read. The Error Queue is considered cleared when it is empty. An empty Error Queue clears the EAV bit in the Status Byte Register.

The commands to control the Error Queue are listed in [Table D-11](#). When you read a single message in the Error Queue, the “oldest” message is read and then removed from the queue. On power-up, the Error Queue is initially empty. If there are problems detected during power-on, entries will be placed in the queue. When empty, the error number 0 and “No Error” is placed in the queue.

Messages in the Error Queue include a code number, message text, severity, and TSP-Link node number. The messages are listed in [Table B-2 on page B-3](#).

Table D-11  
**Error queue commands**

Error queue command	Description
errorqueue.clear()	Clear error queue of all errors.
errorqueue.count	Number of messages in the error/event queue.
errorcode, message, severity, node = errorqueue.next()	Request error code, text message, severity, and TSP-Link node number.

# Speed Specification Test Conditions

---

## Appendix E topics

**Introduction**, page E-2

**Test system used**, page E-2

**Overview**, page E-2

Sweep Operation Rates, page E-2

Single Measurement Rates, page E-3

Function and Range Change Rates, page E-3

Command Processing, page E-4

**Sweep Operation Rates**, page E-4

Measure to Memory, page E-5

Measure to GPIB, page E-5

Source Measure to Memory, page E-6

Source Measure to GPIB, page E-6

Source Measure Pass/Fail to Memory, page E-6

Source Measure Pass/Fail to GPIB, page E-6

**Single Measurement Rates**, page E-6

Measure to GPIB, page E-7

Source Measure to GPIB, page E-7

Source Measure Pass/Fail to GPIB, page E-7

**Function/ Range Change Rates**, page E-8

Source Range Change Rate, page E-8

Measure Range Change Rate, page E-8

Function Change Rate, page E-8

**Command Processing**, page E-9

## Introduction

The purpose of this appendix is to provide a general procedure for obtaining speed results similar to those listed in the Series 2600 Specifications in [Appendix A](#). Tests were performed using Visual Basic and VISA calls.

## Test system used

<b>PC Hardware:</b>	Pentium IV 2.4 GHz, 512MB RAM, National Instruments PCI-GPIB
<b>Model 260x Unit:</b>	Model 2602 (test runs on smuA only)
<b>Software:</b>	Microsoft Windows 2000, Microsoft Visual Basic 6.0, VISA version 3.1
<b>Drivers:</b>	(PCI-GPIB) NI-488.2 version 2.2

## Overview

Speed tests are separated into four categories: Sweep Operation Rates, Single Operation Rates, Function/Range Change Rates, and Command Processing. More detail on the tests is provided in the following paragraphs.

Sweep Operation Rate tests use a script to set up the instrument to record a large number of measurements. The script sends a single print command at the end of the test to signal the test program when it is done. Single Operation Rate tests also use a script to send out measure commands. Samples are received one at a time from the test program instead of collected inside a script. Function/Range Change Rate tests measure the time it takes to change either a function or range. Command Processing tests measure the time it takes to receive and process a command.

### Sweep Operation Rates

The procedure for Sweep Operation Rates test is listed below:

1. Short pins 1 and 2 of the digital I/O connector and power on the unit. Tests involving the digital I/O include time to receive a trigger from pin 2 to pin 1.
2. A script puts the Model 260x into a known test state, maximizing performance of the instrument. It is sent to the instrument but not yet executed. A large number of samples are taken once the script is run. This is necessary to minimize error due to resolution of the Timer() function in Visual Basic.

3. Unit is placed in sync with the test program to guarantee that the timing results will not include execution time from previous commands. A simple way to sync the unit is to issue a print command.

```
Private Sub Sync_Unit()  
    ReceiveBuffer = ""  
    TransmitBuffer = "print('done') & CRLF  
    VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)  
    VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)  
End Sub
```

4. A snapshot of the timer is taken, after which the script mentioned in step 2 is sent to the unit. The test program collects all data returned from the instrument. In the case of "X to memory" tests, a single result is sent to the test program notifying that the test is done. In the case of "X to gpib" speed tests, the program receives all measured data. Finally, another snapshot of the timer is taken. The difference between the start and end time is the speed result.

## Single Measurement Rates

The procedure for a **Single Measurement Rate** test is listed below:

1. A script puts the Model 260x into a known test state maximizing performance of the instrument. It sets up the unit to return one measurement at a time.
2. The unit is placed in sync with the test program to guarantee that the timing will not include an execution time from a previous command.
3. A snapshot of the timer is taken after which the script is sent to the unit. A loop in the test program sends a series of measure commands and obtains results one at a time. When the loop is completed, another snapshot of the timer is taken. The difference between the start and end time is the speed result.

## Function and Range Change Rates

The procedure for a **Function and Range Change Rate** test is listed below:

1. A script puts the Model 260x into a known test state, maximizing performance of the instrument. It is sent to the instrument but not executed. It configures the unit to take a large number of samples that either make range or function changes.
2. The unit is placed in sync with the test program to guarantee that the timing will not include an execution time from previous commands.
3. A snapshot of the timer is taken after which the script is sent to the unit. The test program gets a signal from the unit notifying that the test is complete. Another snapshot of the timer is taken. The difference between the start and end time is the speed result.



## Command Processing

The procedure for a **Command Processing** test is listed below:

1. Using an oscilloscope, connect a probe to GPIB ATN line. Connect another probe to the output of smua. Set up the scope to display both inputs on the appropriate scales.
2. Again, a script is used to put the Model 260x into a known test state. It is sent to the instrument but not executed. The script configures the unit to take a large number of samples.
3. The script is executed causing the output of the smu to change levels.
4. Time is recorded on the scope, measuring the period between the time the GPIB ATN line stops moving, and when the output of the smu begins to change.

## Sweep Operation Rates

All tests in this section use a setup script with the following procedure if internal handshaking is involved. All tests use smua.

1. Set the source range to 1V, turning autorange off.
2. Measure range is set to 1V, turning autorange off.
3. Measurement filter is off.
4. Set the source function to volts.
5. The NPLC is set.
6. Turn output ON.
7. Autozero is set to smua.AUTOZERO\_ONCE.
8. A voltage measurement is taken to get a background reference reading.
9. Autozero is turned off.
10. A for-loop takes the desired number of samples.
11. Measurements are taken, data is saved to a buffer, and an array is created in Lua to store the samples. If the test is a “to GPIB” test, data is returned using the printnumber() function.
12. Turn output OFF.
13. A print command is issued to the test program when the test is complete if the test is “to MEMORY”.

For Digital I/O handshaking:

1. Set the source range to 1V, turning autorange off.
2. Measure range is set to 1V, turning autorange off.
3. Measurement filter is off.

4. Set the trigger mode of line 1 to FALLING, `digio.trigger[1].mode = digio.TRIG_FALLING`.
5. Set the pulse width of line 2, `digio.trigger[2].pulsewidth = 1E-6`
6. Sets the source function to volts.
7. The NPLC is set.
8. Turn output ON.
9. Autozero is set to `smua.AUTOZERO_ONCE`.
10. A measurement is taken internally to get a background reference reading.
11. Autozero is turned off.
12. A for-loop takes the desired number of samples.
13. A loop generates a series of measurements. Before each measurement, `digio.trigger[1].wait` is issued. A voltage measurement is then stored to a buffer. If the test is a "to GPIB" test, data is returned with the `printnumber()` function. Finally, `digio.trigger[2].wait` is sent.
14. Turn output OFF.
15. A print command is issued to the test program when the test is complete if the test is "to MEMORY".

## Measure to Memory

The script uses `smua.measure.v(nvbuffer1)` to take measurements. The measure count is set to a large number such as 1,000 instead of taking data in a for-loop. Results are saved to `nvbuffer1` instead of an array created in Lua.

The following shows how the speed results are obtained in Visual Basic. `TestScript()` is the setup script.

```
tstart = Timer ' Starts timer
  TransmitBuffer = "TestScript()" & CRLF
  ReceiveBuffer = ""
  VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
  VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
tstop = Timer 'Stops timer
```

```
Result = NumPoints/ (tstop - start) ' NumPoints is the number of samples taken
```

## Measure to GPIB

This test is similar to **Measure to Memory**. Instead of storing data to a buffer, data is returned to the test program one at a time. It also sets the return data format to SREAL instead of ASCII. The command `printnumber()` returns the data.

The following shows how the results are obtained in Visual Basic. `TestScript()` is the setup script.

```
tstart = Timer
  TransmitBuffer = "TestScript()" & CRLF
  VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
  For i = 1 To NumPoints ' Number of samples to take
    VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
  Next i
tstop = Timer

Result = NumPoints / (tstop - start) ' NumPoints is the number of samples taken
```

## Source Measure to Memory

The setup for this test is similar to **Measure to Memory**.

The `smua.measurevandstep(Levelv)` function is used instead of `smua.measure.v()` (refer to command documentation in Section 12). `Levelv` increments throughout the test. Data is stored to an array created in Lua.

## Source Measure to GPIB

The setup for this test is similar to **Measure to GPIB** except the `smua.measurevandstep(Levelv)` function is used instead of `smua.measure.v()`. `Levelv` increments throughout the test.

## Source Measure Pass/Fail to Memory

This test is similar to the **Source Measure to Memory** test, and it adds some addition logic. Each measurement is compared to a value, and a flag is set to 1 or 0 depending on whether the measurement is above a threshold or not.

## Source Measure Pass/Fail to GPIB

The setup for this test is similar to **Source Measure to GPIB** and adds some addition logic. Each measurement is compared to a value and a flag is set to 1 or 0 depending on whether the measurement is above a threshold or not.

# Single Measurement Rates

Each test in this section uses a setup script that performs the following:

1. Set the source range to 1V, turning autorange off.
2. Measure range is set to 1V, turning autorange off.
3. Measurement filter is off.
4. Set the source function to volts.
5. The NPLC is set.

6. Turn output ON.
7. Autozero is set to `smua.AUTOZERO_ONCE`.
8. A measurement is taken internally to get a background reference reading.
9. Autozero is turned off.
10. The `format.data` command is set to `format.SREAL`.
11. A function is issued to take a single measurement.
12. Data is returned using `printnumber()`.
13. Turn output OFF.

## Measure to GPIB

Single voltage measurements are recorded using `smua.measure.v()`.

The following shows how the results are obtained in Visual Basic. `TestScript()` is the setup script.

```
TransmitBuffer = "printnumber(smua.measure.v())" & CRLF
tlen = Len(TransmitBuffer)

tstart = Timer
For i = 1 To NumPoints ' NumPoints is the number of samples taken
    VisaStatus = viWrite(VisaSession, TransmitBuffer, tlen, ReturnCount)
    VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
Next i
tstop = Timer

Result = NumPoints/ (tstop - start)
```

## Source Measure to GPIB

This test is similar to **Measure to GPIB**. The `smua.measurevandstep(Levelv)` function is used instead of `smua.measure.v()`. `Levelv` increments are used throughout the test.

## Source Measure Pass/Fail to GPIB

The setup for this test is similar to **Source Measure to GPIB** and adds some addition logic. Each measurement is compared to a value and a flag is set to 1 or 0 depending on whether the measurement is above the threshold or not.

## Function/ Range Change Rates

The tests in this section have the following setup:

1. Set the source range to 1V, turning autorange off.
2. Measure range is set to 1V, turning autorange off.
3. Measurement filter is off.
4. Set the source function to volts.
5. The NPLC is set.
6. Turn output ON.
7. Range or function is altered.
8. Turn output OFF.

### Source Range Change Rate

This test uses the previous setup to alternate between two source ranges.

The following shows how the results are obtained in Visual Basic. TestScript() is the setup script.

```
tstart = Timer
TransmitBuffer = "TestScript()" & CRLF
ReceiveBuffer = ""
VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
tstop = Timer
```

Result = NumPoints/ (tstop – start) ' NumPoints is the number of samples taken

### Measure Range Change Rate

This test method is the same as the **Source Range Change Rate** test, but this one alternates between two measurement ranges.

### Function Change Rate

This test method is the same as the **Source Range Change Rate** test, but this one alternates between voltages and current.

# Command Processing

This test has the following setup:

1. Set the source range to 1V, turning autorange off.
2. Set the source function to volts.
3. Turn output ON.
4. A series of `smua.source.levelv()` commands are sent out to alternate the smu voltage level between 1V and 0V.
5. Time is recorded on the scope measuring the period between the time the GPIB ATN line stops moving and when the output of the smu begins to change.
6. Turn output OFF.

---

# Index

## A

- Accessories 1-5
- Adapters 1-5
- Attributes 12-3, 12-9
- Auto ohms measurements 4-18
- Auto range 6-3
- Auto zero 4-7
  - Front panel 4-8
  - Front panel operation 4-8
  - NPLC caching 4-7
  - Remote command 4-8
  - Remote operation 4-8

## B

- Baud rate 11-12
- Beeper 1-14
- Branching 2-57
- Buffer
  - Commands 7-5
  - Configuration 7-2
  - Dynamically allocated 7-11
  - Front panel 7-2
  - Location number 7-4
  - Overview 7-2
  - Programming examples 7-12
  - Reading attributes 7-8
  - Readings 7-4
  - Remote 7-5
  - Status 7-10
  - Storage attributes 7-6
  - Timestamp 7-4

## C

- Calibration
  - Commands 16-6
  - Considerations 16-3
  - Cycle 16-3
  - Errors 16-5
  - Procedure 16-8
  - Recommended equipment 16-4
  - Steps 16-5
- Capabilities 1-2
  - Source-measure 4-2
- Chassis ground 1-11, 3-3
- Chunk 2-6

- Circuit configurations 4-5, 8-16
  - Basic 4-5
  - Measure only (V or I) 8-18
  - Source I 8-16
  - Source V 8-17
- Command programming 12-2
  - Attributes 12-3, 12-9
  - Conventions 12-2
  - Functions 12-3, 12-9
  - Logical instruments 12-5
  - Reading buffers 12-6
  - Syntax rules 12-4
  - Time and date values 12-8
  - TSP-Link nodes 12-5
- Commands
  - Requesting settings 1-25
- Common commands C-2, E-9
  - \*IDN? C-4
  - \*OPC C-4
  - \*OPC? C-4
  - \*RST C-4
  - \*TRG C-4
  - \*TST? C-4
  - \*WAI C-5
  - Summary C-2
- Compliance
  - Considerations 15-7
  - Limit 4-3, 8-2
  - Maximum 8-2
  - Principles 8-3
  - Setting front panel compliance limit 4-4
  - Setting remote compliance limit 4-5
- Compliance limit
  - Front panel operation 4-4
  - Remote operation 4-5
  - Setting 4-4
- Concatenation 2-56
- Connection
  - GPIB 11-3
- Connectors 1-5
  - Digital I/O 1-11
  - IEEE-488 1-11
  - Output 1-11
  - Power module 1-11
  - RS-232 1-11
  - TSP-Link 1-11
- Contact information 1-3
- Cooling vents 1-12
- Current accuracy 15-9
  - Output 15-9
- Current measurement accuracy 15-11
- Current measurement accuracy limits 15-12

---

## D

- Data store
  - Commands 7-5
  - Front panel 7-2
  - Overview 7-2
  - Programming examples 7-12
  - Remote 7-5
- DCL (device clear) 11-8
- Default settings 1-22
- Default setups 1-23
- Differences, remote vs. local operation 2-49
- Digital I/O port 10-2
  - +5V output 10-2
  - Bit weighting 10-5
  - Commands 10-6
  - Configuration 10-2, 10-4
  - Controlling I/O lines 10-4
  - Front panel control 10-5, 10-6
  - Lines 10-2
  - Output enable 10-3, 10-8
  - Programming examples 10-8
  - Remote operation 10-6
- Digits 6-6, 6-7, 7-12, 7-13, 7-14
  - Remote programming 6-7
- Display
  - Annunciators 1-9
  - Resolution 6-7
- Display modes 1-15
- Display operations
  - Adding menu entries 14-13
  - Character codes 14-7
  - Clearing 14-5
  - Cursor position 14-5
  - Deleting menu entries 14-14
  - Functions and attributes 14-2
  - Input prompting 14-8
  - Keycodes 14-16
  - Load test menu 14-13
  - LOCAL lockout 14-12
  - Measurement functions 14-3
  - Menu 14-8
  - Messages 14-4
  - Resolution 14-4
  - Running a test 14-14
  - Text messages 14-6
  - Triggering 14-15
  - User screen 14-3
- DISPLAY PATTERNS test 17-4
- Duty cycle 8-24

## E

- Editing 1-16
  - Compliance 1-16
  - Controls 1-16
  - Source 1-16
- Environmental conditions 15-2, 16-2
  - Line power 16-2
  - Temperature and relative humidity 16-2
  - Warm-up period 16-2
- Error
  - and status messages 11-9
- Error and status messages 1-21
- Error messages
  - Summary B-2
- Examples
  - Ohms programming example 4-21, 4-23
  - Source-measure programming example 4-16

## F

- Factory scripts 2-48, 13-2
  - Firmware upgrade 13-13
  - Modifying 2-48
  - Running 2-48
- Fall time 8-23
- Features 1-2
- Filter
  - Front panel control 6-12
- Filters 6-12
  - Commands 6-15
  - Configuration menu 6-13
  - Configuring 6-12
  - Enabling 6-13
  - Front panel control 6-12
  - Programming example 6-15
  - Remote programming 6-15
  - Response time 6-12
  - Types of 6-12
- Floating a SMU 3-13
- Flow control (signal handshaking) 11-12
- Front panel
  - GPIB operation 11-9
  - Tests 17-3
- Front panel calibration 16-5
- Front panel operation
  - Source-measure procedure 4-13
- Front panel summaries 1-6
- Functions 2-4, 2-54, 12-3
- Fuse
  - Line, replacement 17-2
- Fuse replacement 1-14



## G

- GET (group execute trigger) 11-8
- GPIB
  - Connections 11-3
  - Error and status messages 11-9
  - Front panel operation 11-9
  - Operation 11-3
  - Primary address 11-5
  - Standards 11-3
  - Status indicators 11-9
  - Terminator 11-6
- GPIB adapter 1-5
- GPIB cables 1-5
- GTL (go to local) 11-8
- Guard 8-19
  - Connections 8-20
  - Overview 8-20
- Guarding 3-8

## I

- IFC (interface clear) 11-7
- Input/output connections 1-11
- Input/output terminal blocks 3-2
- Inspection 1-4
- Interactive script 2-40
- Interface
  - Selecting an 11-2
  - Selection 1-21
- Interface selection
  - GPIB 1-21
  - RS-232 1-21
- Interlock
  - Safety 10-8

## K

- Keys
  - Function 1-8
  - Output control 1-9
  - Range 1-9
  - Special 1-8
- KEYS test 17-3

## L

- Line frequency 1-13
- Line fuse replacement 17-2
- Line power 15-3
- Line power connection 1-13
- LLO (local lockout) 11-8

- LOCAL key 11-10
- Logical operators 2-55
- Loop control 2-58
- Low range limits 6-4
- LSTN 11-9

## M

- Math library functions 2-61
- Measure
  - V or I 8-18
- Measure only 4-16
- Measure voltage or current 4-16
- Menus
  - Configuration 1-18
  - Main 1-18
  - Navigation 1-17

## N

- Noise shield 3-9
- Non-volatile memory 2-8
- NPLC caching 4-7

## O

- Ohms
  - Calculations 4-18
  - Measurement procedure 4-18
  - Measurements 4-18
  - Programming example 4-21
  - Remote programming 4-21
  - Sense selection 4-20
  - Sensing 4-19
- Operating boundaries 8-7
  - I -Source 8-8
  - Source or sink 8-7
  - V-Source 8-12
- Operation
  - Considerations 4-7
  - Overview 4-2
- Options 1-5
- Output current accuracy limits 15-10
- Output enable 10-8
  - Control 10-9
  - Operation 10-8
- Output voltage accuracy 15-7
- Output voltage accuracy limits 15-8
- Output-off states 3-15
- Overheating protection 8-6

---

## P

- Phone number 1-3
- Power
  - Calculations 4-22, 8-6
  - Equations 8-6
  - Measurement procedure 4-22
  - Measurements 4-22
  - Programming example 4-23
  - Remote programming 4-23
- Power module 1-11
- Power switch 1-8
- Power-on setup 1-22, 1-23
- Power-up 1-13
  - Sequence 1-14
- Precedence 2-55
- PRESet default setup 1-23
- Primary address 11-5
- Pulse
  - Concepts 8-23
  - Duty cycle 8-24
  - Period 8-23
  - Rise and fall times 8-23
  - Sweeps 5-6
- Pulse characteristics
  - Pulse duty cycle 8-24
- Pulse energy limitations 8-22
- Pulse sweeps 8-5

## Q

- Queries 2-3
- Queues D-2, D-29
  - Error D-29
  - Output D-29

## R

- Range 6-2
  - Auto 6-3
  - Auto range limits 6-4
  - Available ranges 6-2
  - Commands 6-5
  - Considerations 6-4
  - Limitations 6-3
  - Low limits 6-4
  - Manual 6-3
  - Programming 6-5
  - Programming example 6-6
- Reading buffers 12-6

## Readings

- Maximum 6-3
- Recalling 7-4
- Requesting 1-25, 4-16
- Storing 7-3
- Rear panel summaries 1-6
- Recalling readings 7-4
- Recommended test equipment 15-3
- Recommended verification equipment 15-4
- Reference manual 1-6
- Registers
  - Enable and transition D-15
  - Measurement event D-26
  - Operation event D-21
  - Programming example D-29
  - Questionable event D-24
  - Reading D-11
  - Serial polling and SRQ D-14
  - Service request enable D-13
  - Standard event D-19
  - Status sets D-17
  - System summary event D-17
- Rel 6-10
  - Defining a value 6-10
  - Enabling and disabling 6-10
  - Front panel 6-10
  - Remote programming 6-11
- REM 11-9
- Remote calibration 16-5
- Remote operation
  - Source-measure procedure 4-14
- Remote programming 1-25
  - Ohms 4-21
  - Power 4-23
- REN (remote enable) 11-7
- Restoring factory defaults 15-5
- Rise time 8-23
- Rotary knob 1-9
- RS-232 cable 1-5
- RS-232 Interface
  - Terminator 11-11
- RS-232 interface
  - Baud rate 11-12
  - Connections 11-13
  - Data bits 11-12
  - Flow control 11-12
  - Operation 11-10
  - Parity 11-12
  - Sending and receiving data 11-11
- Run-time environment 2-3, 2-7
  - Memory considerations 2-51

---

## S

- Safety
  - See also "Interlock"
- Safety shield 3-10
- Safety symbols and terms 1-3
- Script management 2-46
- Scripting 2-2
- Scripting Language 2-2
- Scripts 2-3, 2-6
  - Creating 2-17
  - Creating functions 2-5
  - Factory scripts 2-48
  - Importing 2-27
  - Launch configuration 2-22
  - Launching 2-25
  - Modifying 2-17
  - Programming model 2-9
  - Retrieving from Model 260x 2-26
  - Saving 2-19
  - User 2-39
- Scripts, named 2-4
- SDC (selective device clear) 11-8
- Selecting an interface 11-2
- Sensing 3-5
  - 2-wire local 3-5
  - 4-wire remote 3-5
  - Mode selection 3-6
  - Ohms 4-19
- Serial polling D-14
- Setting the measurement range 15-6
- Setting the source range and output value 15-6
- Setups
  - Front panel 1-22
  - Power-on 1-22
  - Restoring 1-22
  - User 1-22
- Shielding
  - Noise 3-9
  - Safety 3-10
- Sink 8-7
- Sink operation 4-17
- SMU connections 3-7
- Source 8-7
- Source I measure I 8-16
- Source V measure V 8-16
- Source-measure capabilities 4-2
- Source-measure procedure
  - Front panel operation 4-13
  - Programming example 4-16
  - Remote operation 4-14
- SPE, SPD (serial polling) 11-8

- Speed 6-8
  - Command 6-9
  - Configuration menu 6-9
  - Programming example 6-10
  - Remote programming 6-9
  - Setting 6-9
- SRQ (Service Request) 11-9, D-11
- Staircase sweeps 8-4
- Standard libraries 2-59
- Status byte and service request (SRQ) D-11
- Status byte and service request commands D-14
- Status model
  - Clearing registers and queues D-9
  - Commands D-8
  - Programming registers and queues D-10
  - Status byte and SRQ D-2, D-11
- Status register sets D-2, D-17
- Storing readings 7-3
- String library functions 2-60
- Sweep
  - Characteristics 5-3
  - Custom 5-7
  - Custom functions 5-10
  - Functions 5-8
  - Linear staircase 5-3
  - List 5-7
  - Logarithmic staircase 5-4
  - Measurement storage 5-8
  - Overview 5-2
  - Programming examples 5-11
  - Pulse 5-6, 8-5
  - Staircase functions 5-9
  - Waveforms 8-4
- Syntax rules 12-4
- System connections 2-10
- System Expansion 9-1

## T

- Tables/arrays 2-53
- TALK 11-9
- Terminator 11-6, 11-11
- Test considerations 15-5
- Test fixture 3-12
- Test Script Builder 2-12
  - Instrument Console 2-28
  - Opening communications 2-15
  - Starting 2-14
- Test Script Processor 2-2
- Tests
  - Front panel 17-3
- Timestamp 7-4

---

Triggering 4-9  
    Front panel 4-11  
    Measurement 4-10  
    Remote 4-12  
    Types of 4-9  
TSP software 2-10  
TSP-Link  
    Abort 9-7  
    Accessing nodes 9-6  
    Connections 9-3  
    Initialization 9-3  
    Master 9-2  
    Node numbers 9-3  
    PC-based system 2-50, 9-2  
    Reset 9-4  
    reset() command 9-7  
    Slaves 9-2  
    Stand-alone system 2-50, 9-2

## U

Unpacking 1-4  
User script 2-39  
    Creating 2-42  
    Modifying 2-46  
    Running 2-43  
    Saving 2-43  
User setups 1-22, 1-23

## V

Variables 2-52  
Vents  
    Cooling 1-11  
Verification 15-2  
    Limits 15-4  
    Test procedures 15-5  
    Test requirements 15-2  
    Test summary 15-5  
Voltage (measure) 4-16  
Voltage accuracy 15-6  
    limit 15-8  
Voltage measurement accuracy 15-8  
Voltage measurement accuracy limits 15-9

## W

Warm-up 4-7  
Warm-up period 15-3  
Warranty information 1-3



# Service Form

**Model No.** \_\_\_\_\_ **Serial No.** \_\_\_\_\_ **Date** \_\_\_\_\_

**Name and Telephone No.** \_\_\_\_\_

**Company** \_\_\_\_\_

List all control settings, describe problem and check boxes that apply to problem. \_\_\_\_\_

Intermittent                       Analog output follows display                       Particular range or function bad; specify \_\_\_\_\_

IEEE failure                       Obvious problem on power-up                       Batteries and fuses are OK

Front panel operational     All ranges or functions are bad                       Checked all cables

Display or output (check one)

Drifts                                       Unable to zero                                       Unstable

Overload                                       Will not read applied input

Calibration only                       Certificate of calibration required                       Data required

(attach any additional sheets as necessary)

Show a block diagram of your measurement including all instruments connected (whether power is turned on or not). Also, describe signal source.

Where is the measurement being performed? (factory, controlled laboratory, out-of-doors, etc.) \_\_\_\_\_

What power line voltage is used? \_\_\_\_\_ Ambient temperature? \_\_\_\_\_ °F

Relative humidity? \_\_\_\_\_ Other? \_\_\_\_\_

Any additional information. (If special modifications have been made by the user, please describe.)

Be sure to include your name and phone number on this service form.

Specifications are subject to change without notice.  
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.  
All other trademarks and trade names are the property of their respective companies.

---

**KEITHLEY**

A G R E A T E R M E A S U R E O F C O N F I D E N C E

**Keithley Instruments, Inc.**

**Corporate Headquarters** • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY (534-8453) • [www.keithley.com](http://www.keithley.com)